NI-DAQmx Python API Documentation

Release 0.5.0

National Instruments

May 01, 2017

API Reference:

Info	Contains a Python API for interacting with NI-DAQmx. See GitHub for the latest source.
Author	National Instruments

CHAPTER 1

About

The **nidaqmx** package contains an API (Application Programming Interface) for interacting with the NI-DAQmx driver. The package is implemented in Python. This package was created and is supported by NI. The package is implemented as a complex, highly object-oriented wrapper around the NI-DAQmx C API using the ctypes Python library.

nidaqmx 0.5 supports all versions of the NI-DAQmx driver that ships with the C API. The C API is included in any version of the driver that supports it. The **nidaqmx** package does not require installation of the C header files.

Some functions in the **nidaqmx** package may be unavailable with earlier versions of the NI-DAQmx driver. Visit the ni.com/downloads to upgrade your version of NI-DAQmx.

nidaqmx supports only the Windows operating system.

nidaqmx supports CPython 2.7, 3.4+, PyPy2, and PyPy3.

CHAPTER 2

Features

The following represents a non-exhaustive list of supported features for **nidaqmx**:

- Fully-object oriented
- Fully-featured Task class
- Fully-featured Scale class
- Fully-featured System sub-package with System, Device, PhysicalChannel, WatchdogTask, etc. classes
- NI-DAQmx Events
- NI-DAQmx Streams
- Enums support in both Python 2 and 3
- Exceptions support
- Warnings support
- Collections that emulate Python container types
- Single, dynamic read and write methods (see *Usage*)
- Performant, NumPy-based reader and writer classes
- Optional parameters
- Implicitly verified properties
- Context managers

The following features are not yet supported by the **nidaqmx** package:

- Calibration methods
- Real-time methods

CHAPTER 3

Installation

Running **nidaqmx** requires NI-DAQmx or NI-DAQmx Runtime. Visit the ni.com/downloads to download the latest version of NI-DAQmx.

nidaqmx can be installed with pip:

```
$ python -m pip install nidaqmx
```

Or **easy_install** from setuptools:

```
$ python -m easy_install nidaqmx
```

You also can download the project source and run:

```
$ python setup.py install
```

CHAPTER 4

Usage

The following is a basic example of using an `nidaqmx.task.Task` object. This example illustrates how the single, dynamic `nidaqmx.task.Task.read()` method returns the appropriate data type.

```
>>> import nidaqmx
>>> with nidaqmx.Task() as task:
...     task.ai_channels.add_ai_voltage_chan("Dev1/ai0")
...     task.read()
...
-0.07476920729381246
>>> with nidaqmx.Task() as task:
...     task.ai_channels.add_ai_voltage_chan("Dev1/ai0")
...     task.read(number_of_samples_per_channel=2)
...
[0.26001373311970705, 0.37796597238117036]
>>> from nidaqmx.constants import LineGrouping
>>> with nidaqmx.Task() as task:
...     task.di_channels.add_di_chan(
...         "cDAQ2Mod4/port0/line0:1", line_grouping=LineGrouping.CHAN_PER_LINE)
...     task.read(number_of_samples_per_channel=2)
...
[[False, True], [True, True]]
```

A single, dynamic `nidaqmx.task.Task.write()` method also exists.

```
>>> import nidaqmx
>>> from nidaqmx.types import CtrTime
>>> with nidaqmx.Task() as task:
...     task.co_channels.add_co_pulse_chan_time("Dev1/ctr0")
...     sample = CtrTime(high_time=0.001, low_time=0.001)
...     task.write(sample)
...
1
>>> with nidaqmx.Task() as task:
...     task.ao_channels.add_ao_voltage_chan("Dev1/ao0")
...     task.write([1.1, 2.2, 3.3, 4.4, 5.5], auto_start=True)
```

```
...
5
```

Consider using the *nidaqmx.stream_readers* and *nidaqmx.stream_writers* classes to increase the performance of your application, which accept pre-allocated NumPy arrays.

Following is an example of using an `nidaqmx.system.System` object.

```
>>> import nidaqmx.system
>>> system = nidaqmx.system.System.local()
>>> system.driver_version
DriverVersion(major_version=16L, minor_version=0L, update_version=0L)
>>> for device in system.devices:
...     print(device)
...
Device(name=Dev1)
Device(name=Dev2)
Device(name=cDAQ1)
>>> import collections
>>> isinstance(system.devices, collections.Sequence)
True
>>> device = system.devices['Dev1']
>>> device == nidaqmx.system.Device('Dev1')
True
>>> isinstance(device.ai_physical_chans, collections.Sequence)
True
>>> phys_chan = device.ai_physical_chans['ai0']
>>> phys_chan
PhysicalChannel(name=Dev1/ai0)
>>> phys_chan == nidaqmx.system.PhysicalChannel('Dev1/ai0')
True
>>> phys_chan.ai_term_cfgs
[<TerminalConfiguration.RSE: 10083>, <TerminalConfiguration.NRSE: 10078>,
↪<TerminalConfiguration.DIFFERENTIAL: 10106>]
>>> from enum import Enum
>>> isinstance(phys_chan.ai_term_cfgs[0], Enum)
True
```

CHAPTER 5

Support / Feedback

The **nidaqmx** package is supported by NI. For support for **nidaqmx**, open a request through the NI support portal at ni.com.

CHAPTER 6

Bugs / Feature Requests

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

Information to Include When Asking for Help

Please include **all** of the following information when opening an issue:

- Detailed steps on how to reproduce the problem and full traceback, if applicable.
- The python version used:

```
$ python -c "import sys; print(sys.version)"
```

- The versions of the **nidaqmx**, numpy, six and enum34 packages used:

```
$ python -m pip list
```

- The version of the NI-DAQmx driver used. Follow this KB article to determine the version of NI-DAQmx you have installed.
- The operating system and version, for example Windows 7, CentOS 7.2, ...

CHAPTER 7

Additional Documentation

Refer to the NI-DAQmx Help for API-agnostic information about NI-DAQmx or measurement concepts.

NI-DAQmx Help installs only with the full version of NI-DAQmx.

CHAPTER 8

License

nidaqmx 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.

nidaqmx.constants

class `nidaqmx.constants.`**`ACExcitWireMode`**

Bases: `enum.Enum`

`FIVE_WIRE` **= 5**

5-wire.

`FOUR_WIRE` **= 4**

4-wire.

`SIX_WIRE` **= 6**

6-wire.

class `nidaqmx.constants.`**`ADCTimingMode`**

Bases: `enum.Enum`

`AUTOMATIC` **= 16097**

Uses the most appropriate supported timing mode based on the Sample Clock Rate.

`BEST_50_HZ_REJECTION` **= 14713**

Improves 50 Hz noise rejection while decreasing noise rejection at other frequencies.

`BEST_60_HZ_REJECTION` **= 14714**

Improves 60 Hz noise rejection while decreasing noise rejection at other frequencies.

`CUSTOM` **= 10137**

Use **ai_adc_custom_timing_mode** to specify a custom value controlling the tradeoff between speed and resolution.

`HIGH_RESOLUTION` **= 10195**

Increases resolution and noise rejection while decreasing conversion rate.

HIGH_SPEED = 14712
Increases conversion rate while decreasing resolution.

class nidaqmx.constants.**AOIdleOutputBehavior**
Bases: enum.Enum

HIGH_IMPEDANCE = 12527
Set the channel to high-impedance, effectively disconnecting the analog output circuitry from the I/O connector.

MAINTAIN_EXISTING_VALUE = 12528
Continue generating the current value.

ZERO_VOLTS = 12526
Generate 0 V.

class nidaqmx.constants.**AOPowerUpOutputBehavior**
Bases: enum.Enum

CURRENT = 10134
Current output.

HIGH_IMPEDANCE = 12527
High-impedance state.

VOLTAGE = 10322
Voltage output.

class nidaqmx.constants.**AccelChargeSensitivityUnits**
Bases: enum.Enum

PICO_COULOMBS_PER_G = 16099
PicoCoulombs per g.

PICO_COULOMBS_PER_INCHES_PER_SECOND_SQUARED = 16101
PicoCoulombs per in/s^2.

PICO_COULOMBS_PER_METERS_PER_SECOND_SQUARED = 16100
PicoCoulombs per m/s^2.

class nidaqmx.constants.**AccelSensitivityUnits**
Bases: enum.Enum

M_VOLTS_PER_G = 12509
mVolts/g.

VOLTS_PER_G = 12510
Volts/g.

class nidaqmx.constants.**AccelUnits**
Bases: enum.Enum

FROM_CUSTOM_SCALE = 10065
Units a custom scale specifies. If you select this value, you must specify a custom scale name.

G = 10186
1 g is approximately equal to 9.81 m/s/s.

INCHES_PER_SECOND_SQUARED = 12471
Inches per second per second.

METERS_PER_SECOND_SQUARED = 12470
Meters per second per second.

class `nidaqmx.constants.`**`AcquisitionType`**
Bases: `enum.Enum`

`CONTINUOUS` = 10123
Acquire or generate samples until you stop the task.

`FINITE` = 10178
Acquire or generate a finite number of samples.

`HW_TIMED_SINGLE_POINT` = 12522
Acquire or generate samples continuously using hardware timing without a buffer. Hardware timed single point sample mode is supported only for the sample clock and change detection timing types.

class `nidaqmx.constants.`**`Action`**
Bases: `enum.Enum`

`CANCEL` = 1
Cancel

`COMMIT` = 0
Commit

class `nidaqmx.constants.`**`ActiveLevel`**
Bases: `enum.Enum`

`ABOVE` = 10093
Pause the measurement or generation while the signal is above the threshold.

`BELOW` = 10107
Pause the measurement or generation while the signal is below the threshold.

class `nidaqmx.constants.`**`ActiveOrInactiveEdgeSelection`**
Bases: `enum.Enum`

`ACTIVE` = 14617
Active edges.

`INACTIVE` = 14618
Inactive edges.

class `nidaqmx.constants.`**`AngleUnits`**
Bases: `enum.Enum`

`DEGREES` = 10146
Degrees.

`FROM_CUSTOM_SCALE` = 10065
Units a custom scale specifies. If you select this value, you must specify a custom scale name.

`RADIANS` = 10273
Radians.

`TICKS` = 10304
Ticks.

class `nidaqmx.constants.`**`AngularVelocityUnits`**
Bases: `enum.Enum`

`DEGREES_PER_SECOND` = 16082
Degrees per second.

`FROM_CUSTOM_SCALE` = 10065
Units a custom scale specifies. If you select this value, you must specify a custom scale name.

RADIANS_PER_SECOND = 16081
Radians per second.

RPM = 16080
Revolutions per minute.

class nidaqmx.constants.**AutoZeroType**
Bases: enum.Enum

EVERY_SAMPLE = 10164
Perform an auto zero at every sample of the acquisition.

NONE = 10230
Do not perform an autozero.

ONCE = 10244
Perform an auto zero at the beginning of the acquisition. This auto zero task might not run if you have used DAQmx Control Task previously in your task.

class nidaqmx.constants.**BreakMode**
Bases: enum.Enum

BREAK_BEFORE_MAKE = 10110
When advancing to the next entry in the scan list, disconnect all previous connections before making any new connections.

NO_ACTION = 10227
When advancing to the next entry in the scan list, leave all previous connections intact.

class nidaqmx.constants.**BridgeConfiguration**
Bases: enum.Enum

FULL_BRIDGE = 10182
Sensor is a full bridge. If you set **ai_excit_use_for_scaling** to True, NI-DAQmx divides the measurement by the excitation value. Many sensors scale data to native units using scaling of volts per excitation.

HALF_BRIDGE = 10187
Sensor is a half bridge. If you set **ai_excit_use_for_scaling** to True, NI-DAQmx divides the measurement by the excitation value. Many sensors scale data to native units using scaling of volts per excitation.

NO_BRIDGE = 10228
Sensor is not a Wheatstone bridge.

QUARTER_BRIDGE = 10270
Sensor is a quarter bridge. If you set **ai_excit_use_for_scaling** to True, NI-DAQmx divides the measurement by the excitation value. Many sensors scale data to native units using scaling of volts per excitation.

class nidaqmx.constants.**BridgeElectricalUnits**
Bases: enum.Enum

M_VOLTS_PER_VOLT = 15897
Millivolts per volt.

VOLTS_PER_VOLT = 15896
Volts per volt.

class nidaqmx.constants.**BridgePhysicalUnits**
Bases: enum.Enum

BAR = 15880
Bar.

FOOT_POUNDS = 15884
Pound-feet.

INCH_OUNCES = 15882
Ounce-inches.

INCH_POUNDS = 15883
Pound-inches.

KILOGRAM_FORCE = 15877
kilograms-force.

NEWTONS = 15875
Newtons.

NEWTON_METERS = 15881
Newton metres.

PASCALS = 10081
Pascals.

POUNDS = 15876
Pounds.

POUNDS_PER_SQ_INCH = 15879
Pounds per square inch.

class `nidaqmx.constants.`**`BridgeShuntCalSource`**
Bases: `enum.Enum`

BUILT_IN = 10200
Use the internal shunt.

USER_PROVIDED = 10167
Use an external shunt.

class `nidaqmx.constants.`**`BridgeUnits`**
Bases: `enum.Enum`

FROM_CUSTOM_SCALE = 10065
Units a custom scale specifies. If you select this value, you must specify a custom scale name.

FROM_TEDS = 12516
Units defined by TEDS information associated with the channel.

M_VOLTS_PER_VOLT = 15897
Millivolts per volt.

VOLTS_PER_VOLTS = 15896
Volts per volt.

class `nidaqmx.constants.`**`BusType`**
Bases: `enum.Enum`

COMPACT_DAQ = 14637
CompactDAQ.

PCI = 12582
PCI.

PCIE = 13612
PCI Express.

PC_CARD = 12585
PC Card/PCMCIA.

PXI = 12583
PXI.

PXIE = 14706
PXI Express.

SCC = 14707
SCC.

SCXI = 12584
SCXI.

SWITCH_BLOCK = 15870
SwitchBlock.

TCPIP = 14828
TCP/IP.

UNKNOWN = 12588
Unknown bus type.

USB = 12586
USB.

class `nidaqmx.constants.`**`CJCSource`**
Bases: `enum.Enum`

BUILT_IN = 10200
Use a cold-junction compensation channel built into the terminal block.

CONSTANT_USER_VALUE = 10116
You must specify the cold-junction temperature.

SCANNABLE_CHANNEL = 10113
Use a channel for cold-junction compensation.

class `nidaqmx.constants.`**`CalibrationMode2`**
Bases: `enum.Enum`

CHARGE = 16105
Charge

VOLTAGE = 10322
Voltage

class `nidaqmx.constants.`**`CalibrationTerminalConfig`**
Bases: `enum.Enum`

DIFF = 10106
Differential

PSEUDO_DIFF = 12529
Pseudodifferential

class `nidaqmx.constants.`**`ChannelType`**
Bases: `enum.Enum`

ANALOG_INPUT = 10100
Analog input channel.

ANALOG_OUTPUT = 10102
Analog output channel.

COUNTER_INPUT = 10131
Counter input channel.

COUNTER_OUTPUT = 10132
Counter output channel.

DIGITAL_INPUT = 10151
Digital input channel.

DIGITAL_OUTPUT = 10153
Digital output channel.

class nidaqmx.constants.**ChargeUnits**
Bases: enum.Enum

COULOMBS = 16102
Coulombs.

FROM_CUSTOM_SCALE = 10065
Units a custom scale specifies. If you select this value, you must specify a custom scale name.

PICO_COULOMBS = 16103
PicoCoulombs.

class nidaqmx.constants.**ConstrainedGenMode**
Bases: enum.Enum

FIXED_50_PERCENT_DUTY_CYCLE = 14711
Pulse duty cycle must be 50 percent. The frequency can change while the task runs.

FIXED_HIGH_FREQ = 14709
Pulse frequency must be above 7.63 Hz and cannot change while the task runs. In this mode, the duty cycle has 8 bits of resolution.

FIXED_LOW_FREQ = 14710
Pulse frequency must be below 366.21 Hz and cannot change while the task runs. In this mode, the duty cycle has 16 bits of resolution.

UNCONSTRAINED = 14708
Counter has no restrictions on pulse generation.

class nidaqmx.constants.**CountDirection**
Bases: enum.Enum

COUNT_DOWN = 10124
Decrement counter.

COUNT_UP = 10128
Increment counter.

EXTERNAL_SOURCE = 10326
The state of a digital line controls the count direction. Each counter has a default count direction terminal.

class nidaqmx.constants.**CounterFrequencyMethod**
Bases: enum.Enum

DYNAMIC_AVERAGING = 16065
Uses one counter and automatically configures the counter settings based on the range of frequencies to be measured. During the acquisition, the counter dynamically adjusts the number of periods that are averaged to balance measurement accuracy and latency.

HIGH_FREQUENCY_2_COUNTERS = 10157
Use two counters, one of which counts pulses of the signal to measure during the specified measurement time.

LARGE_RANGE_2_COUNTERS = 10205
Use one counter to divide the frequency of the input signal to create a lower-frequency signal that the second counter can more easily measure.

LOW_FREQUENCY_1_COUNTER = 10105
Use one counter that uses a constant timebase to measure the input signal.

class nidaqmx.constants.**Coupling**
Bases: enum.Enum

AC = 10045
Remove the DC offset from the signal.

DC = 10050
Allow NI-DAQmx to measure all of the signal.

GND = 10066
Remove the signal from the measurement and measure only ground.

class nidaqmx.constants.**CurrentShuntResistorLocation**
Bases: enum.Enum

EXTERNAL = 10167
Use a shunt resistor external to the device. You must specify the value of the shunt resistor by using **ai_current_shunt_resistance**.

INTERNAL = 10200
Use the built-in shunt resistor of the device.

LET_DRIVER_CHOOSE = -1

class nidaqmx.constants.**CurrentUnits**
Bases: enum.Enum

AMPS = 10342
Amperes.

FROM_CUSTOM_SCALE = 10065
Units a custom scale specifies. If you select this value, you must specify a custom scale name.

FROM_TEDS = 12516
Units defined by TEDS information associated with the channel.

class nidaqmx.constants.**DataJustification**
Bases: enum.Enum

LEFT = 10209
Samples occupy the higher bits of the integer.

RIGHT = 10279
Samples occupy the lower bits of the integer.

class nidaqmx.constants.**DataTransferActiveTransferMode**
Bases: enum.Enum

DMA = 10054
Direct Memory Access. Data transfers take place independently from the application.

INTERRUPT = 10204
Data transfers take place independently from the application. Using interrupts increases CPU usage because the CPU must service interrupt requests. Typically, you should use interrupts if the device is out of DMA channels.

POLLED = 10264
Data transfers take place when you call DAQmx Read or DAQmx Write.

USB_BULK = 12590
Data transfers take place independently from the application using a USB bulk pipe.

class `nidaqmx.constants.`**`DeassertCondition`**
Bases: `enum.Enum`

ONBOARD_MEMORY_CUSTOM_THRESHOLD = 12577
Deassert the signal when the amount of space available in the onboard memory is below the value specified with **rdy_for_xfer_event_deassert_cond_custom_threshold**.

ON_BOARD_MEMORY_FULL = 10236
Deassert the signal when the onboard memory fills.

ON_BOARD_MEMORY_MORE_THAN_HALF_FULL = 10237
Deassert the signal when more than half of the onboard memory of the device fills.

class `nidaqmx.constants.`**`DigitalDriveType`**
Bases: `enum.Enum`

ACTIVE_DRIVE = 12573
Drive the output pin to approximately 0 V for logic low and +3.3 V or +5 V, depending on the device, for logic high.

OPEN_COLLECTOR = 12574
Drive the output pin to 0 V for logic low. For logic high, the output driver assumes a high-impedance state and does not drive a voltage.

class `nidaqmx.constants.`**`DigitalPatternCondition`**
Bases: `enum.Enum`

PATTERN_DOES_NOT_MATCH = 10253
Trigger when the physical channels do not match the specified pattern.

PATTERN_MATCHES = 10254
Trigger when the physical channels match the specified pattern.

class `nidaqmx.constants.`**`DigitalWidthUnits`**
Bases: `enum.Enum`

SAMPLE_CLOCK_PERIODS = 10286
Complete periods of the Sample Clock.

SECONDS = 10364
Seconds.

TICKS = 10304
Timebase ticks.

class `nidaqmx.constants.`**`EddyCurrentProxProbeSensitivityUnits`**
Bases: `enum.Enum`

IL = 14837
Volts/mil.

ILLIMETER = 14839
Volts/mMeter.

MICRON = 14840
mVolts/micron.

MIL = 14836
mVolts/mil.

MILLIMETER = 14838
mVolts/mMeter.

class `nidaqmx.constants.`**`Edge`**
Bases: `enum.Enum`

FALLING = 10171
Falling edge(s).

RISING = 10280
Rising edge(s).

class `nidaqmx.constants.`**`EncoderType`**
Bases: `enum.Enum`

TWO_PULSE_COUNTING = 10313
Two pulse counting.

X_1 = 10090
If signal A leads signal B, count the rising edges of signal A. If signal B leads signal A, count the falling edges of signal A.

X_2 = 10091
Count the rising and falling edges of signal A.

X_4 = 10092
Count the rising and falling edges of signal A and signal B.

class `nidaqmx.constants.`**`EncoderZIndexPhase`**
Bases: `enum.Enum`

AHIGH_BHIGH = 10040
Reset the measurement when signal A and signal B are high.

AHIGH_BLOW = 10041
Reset the measurement when signal A is high and signal B is low.

ALOW_BHIGH = 10042
Reset the measurement when signal A is low and signal B high.

ALOW_BLOW = 10043
Reset the measurement when signal A and signal B are low.

class `nidaqmx.constants.`**`EveryNSamplesEventType`**
Bases: `enum.Enum`

ACQUIRED_INTO_BUFFER = 1
Acquired Into Buffer

TRANSFERRED_FROM_BUFFER = 2
Transferred From Buffer

class `nidaqmx.constants.`**`ExcitationDCorAC`**
Bases: `enum.Enum`

USE_AC = 10045
AC excitation.

USE_DC = 10050
DC excitation.

class nidaqmx.constants.**ExcitationIdleOutputBehavior**
Bases: enum.Enum

MAINTAIN_EXISTING_VALUE = 12528
Continue generating the current value.

ZERO_VOLTS_OR_AMPERES = 12526
Drive excitation output to zero.

class nidaqmx.constants.**ExcitationSource**
Bases: enum.Enum

EXTERNAL = 10167
Use an excitation source other than the built-in excitation source of the device. If you select this value, you must specify the amount of excitation.

INTERNAL = 10200
Use the built-in excitation source of the device. If you select this value, you must specify the amount of excitation.

NONE = 10230
Supply no excitation to the channel.

class nidaqmx.constants.**ExcitationVoltageOrCurrent**
Bases: enum.Enum

USE_CURRENT = 10134
Current excitation.

USE_VOLTAGE = 10322
Voltage excitation.

class nidaqmx.constants.**ExportAction**
Bases: enum.Enum

INTERLOCKED = 12549
Handshake Event deasserts after the Handshake Trigger asserts, plus the amount of time specified with **hshk_event_interlocked_deassert_delay**.

LEVEL = 10210
The exported Sample Clock goes high at the beginning of the sample and goes low when the last AI Convert begins.

PULSE = 10265
Send a pulse to the terminal.

TOGGLE = 10307
Toggle the state of the terminal from low to high or from high to low.

class nidaqmx.constants.**FillMode**
Bases: enum.Enum

GROUP_BY_CHANNEL = 0
Group by Channel

GROUP_BY_SCAN_NUMBER = 1
Group by Scan Number

class nidaqmx.constants.**FilterResponse**
Bases: enum.Enum

BUTTERWORTH = 16076
Butterworth filter response.

CONSTANT_GROUP_DELAY = 16075
Constant group delay filter response.

ELLIPTICAL = 16077
Elliptical filter response.

HARDWARE_DEFINED = 10191
Use the hardware-defined filter response.

class nidaqmx.constants.**FilterType**
Bases: enum.Enum

BANDPASS = 16073
Bandpass filter.

CUSTOM = 10137
Custom filter.

HIGHPASS = 16072
Highpass filter.

LOWPASS = 16071
Lowpass filter.

NOTCH = 16074
Notch filter.

class nidaqmx.constants.**ForceIEPESensorSensitivityUnits**
Bases: enum.Enum

M_VOLTS_PER_NEWTON = 15891
Millivolts per newton.

M_VOLTS_PER_POUND = 15892
Millivolts per pound.

class nidaqmx.constants.**ForceUnits**
Bases: enum.Enum

FROM_CUSTOM_SCALE = 10065
Units a custom scale specifies. If you select this value, you must specify a custom scale name.

KILOGRAM_FORCE = 15877
Kilograms-force.

NEWTONS = 15875
Newtons.

POUNDS = 15876
Pounds.

class nidaqmx.constants.**FrequencyUnits**
Bases: enum.Enum

FROM_CUSTOM_SCALE = 10065
Units a custom scale specifies. If you select this value, you must specify a custom scale name.

HZ = 10373
Hertz.

TICKS = 10304
Timebase ticks.

class `nidaqmx.constants.`**`FuncGenType`**
Bases: `enum.Enum`

SAWTOOTH = 14754
Sawtooth wave.

SINE = 14751
Sine wave.

SQUARE = 14753
Square wave.

TRIANGLE = 14752
Triangle wave.

class `nidaqmx.constants.`**`GpsSignalType`**
Bases: `enum.Enum`

IRIGB = 10070
Use the IRIG-B synchronization method. The GPS receiver sends one synchronization pulse per second, as well as information about the number of days, hours, minutes, and seconds that elapsed since the beginning of the current year.

NONE = 10230
Do not synchronize the counter to a GPS receiver. The timestamp measurement returns the number of seconds that elapsed since the device powered up unless you set **ci_timestamp_initial_seconds**.

PPS = 10080
Use the PPS synchronization method. The GPS receiver sends one synchronization pulse per second, but does not send any timing information. The timestamp measurement returns the number of seconds that elapsed since the device powered up unless you set **ci_timestamp_initial_seconds**.

class `nidaqmx.constants.`**`HandshakeStartCondition`**
Bases: `enum.Enum`

IMMEDIATE = 10198
Device is waiting for space in the FIFO (for acquisition) or waiting for samples (for generation).

WAIT_FOR_HANDSHAKE_TRIGGER_ASSERT = 12550
Device is waiting for the Handshake Trigger to assert.

WAIT_FOR_HANDSHAKE_TRIGGER_DEASSERT = 12551
Device is waiting for the Handshake Trigger to deassert.

class `nidaqmx.constants.`**`Impedance1`**
Bases: `enum.Enum`

FIFTY_OHMS = 50
50 Ohms.

ONE_M_OHM = 1000000
1 M Ohm.

SEVENTY_FIVE_OHMS = 75
75 Ohms.

TEN_G_OHMS = 10000000000
10 G Ohm.

class `nidaqmx.constants.`**`InputCalSource`**
Bases: `enum.Enum`

GROUND = 2
Ground

LOOPBACK_0 = 0
Loopback 0 degree shift

LOOPBACK_180 = 1
Loopback 180 degree shift

class `nidaqmx.constants.`**`InputDataTransferCondition`**
Bases: `enum.Enum`

ONBOARD_MEMORY_CUSTOM_THRESHOLD = 12577
Transfer data from the device when the number of samples specified with **ai_data_xfer_custom_threshold** are in the device FIFO.

ON_BOARD_MEMORY_MORE_THAN_HALF_FULL = 10237
Transfer data from the device when more than half of the onboard memory of the device fills.

ON_BOARD_MEMORY_NOT_EMPTY = 10241
Transfer data from the device when there is data in the onboard memory.

WHEN_ACQUISITION_COMPLETE = 12546
Transfer data when the acquisition is complete.

class `nidaqmx.constants.`**`LVDTSensitivityUnits`**
Bases: `enum.Enum`

M_VOLTS_PER_VOLT_PER_MILLIMETER = 12506
mVolts/Volt/mMeter.

M_VOLTS_PER_VOLT_PER_MILLI_INCH = 12505
mVolts/Volt/0.001 Inch.

class `nidaqmx.constants.`**`Language`**
Bases: `enum.Enum`

CHS = 5

DEU = 2

ENG = 0

FRA = 1

JPN = 3

KOR = 4

RAW = -1

class `nidaqmx.constants.`**`LengthUnits`**
Bases: `enum.Enum`

FROM_CUSTOM_SCALE = 10065
Units a custom scale specifies. If you select this value, you must specify a custom scale name.

INCHES = 10379
Inches.

METERS = 10219
Meters.

TICKS = 10304
Ticks.

class nidaqmx.constants.**Level**
Bases: enum.Enum

HIGH = 10192
Logic high.

LOW = 10214
Logic low.

NO_CHANGE = 10160
Do not change the state of the lines. On some devices, you can select this value only for entire ports.

TRISTATE = 10310
High-impedance state. You can select this state only on devices with bidirectional lines. You cannot select this state for dedicated digital output lines. On some devices, you can select this value only for entire ports.

class nidaqmx.constants.**LineGrouping**
Bases: enum.Enum

CHAN_FOR_ALL_LINES = 1
One Channel For All Lines

CHAN_PER_LINE = 0
One Channel For Each Line

class nidaqmx.constants.**LoggingMode**
Bases: enum.Enum

LOG = 15844
Enable logging for the task. You cannot read data using DAQmx Read when using this mode. If you require access to the data, read from the TDMS file.

LOG_AND_READ = 15842
Enable both logging and reading data for the task. You must use DAQmx Read to read samples for NI-DAQmx to stream them to disk.

OFF = 10231
Disable logging for the task.

class nidaqmx.constants.**LoggingOperation**
Bases: enum.Enum

CREATE = 15848
Create a new TDMS file. If the file already exists, NI-DAQmx returns an error.

CREATE_OR_REPLACE = 15847
Create a new TDMS file, or replace an existing TDMS file.

OPEN = 10437
Open an existing TDMS file, and append data to that file. If the file does not exist, NI-DAQmx returns an error.

OPEN_OR_CREATE = 15846
Open an existing TDMS file, and append data to that file. If the file does not exist, NI-DAQmx creates a new TDMS file.

class `nidaqmx.constants.`**`LogicFamily`**

Bases: `enum.Enum`

`FIVE_V` **= 14619**

Compatible with TTL and 5 V CMOS signals.

`THREE_POINT_THREE_V` **= 14621**

Compatible with LVTTL signals.

`TWO_POINT_FIVE_V` **= 14620**

Compatible with 2.5 V CMOS signals.

class `nidaqmx.constants.`**`LogicLvlBehavior`**

Bases: `enum.Enum`

`NONE` **= 10230**

Supply no excitation to the channel.

`PULL_UP` **= 16064**

High logic.

class `nidaqmx.constants.`**`MIOAIConvertTimebaseSource`**

Bases: `enum.Enum`

`EIGHTY_M_HZ_TIMEBASE` **= 14636**

Use the onboard 80 MHz timebase.

`EIGHT_M_HZ_TIMEBASE` **= 16023**

Use the onboard 8 MHz timebase.

`MASTER_TIMEBASE` **= 10282**

Use the same source as the Master Timebase.

`ONE_HUNDRED_M_HZ_TIMEBASE` **= 15857**

Use the onboard 100 MHz timebase.

`SAMPLE_TIMEBASE` **= 10284**

Use the same source as Sample Clock timebase.

`TWENTY_M_HZ_TIMEBASE` **= 12537**

Use the onboard 20 MHz timebase.

class `nidaqmx.constants.`**`ModulationType`**

Bases: `enum.Enum`

`AM` **= 14756**

Amplitude modulation.

`FM` **= 14757**

Frequency modulation.

`NONE` **= 10230**

No modulation.

class `nidaqmx.constants.`**`OutputDataTransferCondition`**

Bases: `enum.Enum`

`ON_BOARD_MEMORY_EMPTY` **= 10235**

Transfer data to the device only when there is no data in the onboard memory of the device.

`ON_BOARD_MEMORY_HALF_FULL_OR_LESS` **= 10239**

Transfer data to the device any time the onboard memory is less than half full.

ON_BOARD_MEMORY_LESS_THAN_FULL = 10242
Transfer data to the device any time the onboard memory of the device is not full.

class nidaqmx.constants.**OverflowBehavior**
Bases: enum.Enum

GNORE_OVERRUNS = 15863
NI-DAQmx ignores Sample Clock overruns, and the task continues to run.

TOP_TASK_AND_ERROR = 15862
Stop task and return an error.

class nidaqmx.constants.**OverwriteMode**
Bases: enum.Enum

DO_NOT_OVERWRITE_UNREAD_SAMPLES = 10159
The acquisition stops when it encounters a sample in the buffer that you have not read.

OVERWRITE_UNREAD_SAMPLES = 10252
When an acquisition encounters unread data in the buffer, the acquisition continues and overwrites the unread samples with new ones. You can read the new samples by setting **relative_to** to **ReadRelativeTo.MOST_RECENT_SAMPLE** and setting **offset** to the appropriate number of samples.

class nidaqmx.constants.**PathCapability**
Bases: enum.Enum

CHANNEL_IN_USE = 10434

CHANNEL_RESERVED_FOR_ROUTING = 10436

CHANNEL_SOURCE_CONFLICT = 10435

PATH_ALREADY_EXISTS = 10432

PATH_AVAILABLE = 10431

PATH_UNSUPPORTED = 10433

class nidaqmx.constants.**Polarity**
Bases: enum.Enum

ACTIVE_HIGH = 10095
High state is the active state.

ACTIVE_LOW = 10096
Low state is the active state.

class nidaqmx.constants.**PowerUpChannelType**
Bases: enum.Enum

CHANNEL_CURRENT = 1
Current Channel

CHANNEL_HIGH_IMPEDANCE = 2
High-Impedance Channel

CHANNEL_VOLTAGE = 0
Voltage Channel

class nidaqmx.constants.**PowerUpStates**
Bases: enum.Enum

HIGH = 10192
Logic high.

LOW = 10214
Logic low.

TRISTATE = 10310
High-impedance state. You can select this state only on devices with bidirectional lines. You cannot select this state for dedicated digital output lines. On some devices, you can select this value only for entire ports.

class `nidaqmx.constants.`**`PressureUnits`**
Bases: `enum.Enum`

BAR = 15880
Bar.

FROM_CUSTOM_SCALE = 10065
Units a custom scale specifies. If you select this value, you must specify a custom scale name.

PASCALS = 10081
Pascals.

POUNDS_PER_SQ_INCH = 15879
Pounds per square inch.

class `nidaqmx.constants.`**`ProductCategory`**
Bases: `enum.Enum`

AO_SERIES = 14647
AO Series.

B_SERIES_DAQ = 14662
B Series DAQ.

COMPACT_DAQ_CHASSIS = 14658
CompactDAQ chassis.

C_SERIES_MODULE = 14659
C Series I/O module.

DIGITAL_IO = 14648
Digital I/O.

DSA = 14649
Dynamic Signal Acquisition.

E_SERIES_DAQ = 14642
E Series DAQ.

M_SERIES_DAQ = 14643
M Series DAQ.

NETWORK_DAQ = 14829
Network DAQ.

NIELVIS = 14755
NI ELVIS.

SCC_CONNECTOR_BLOCK = 14704
SCC Connector Block.

SCC_MODULE = 14705
SCC Module.

SCXI_MODULE = 14660
SCXI module.

SC_EXPRESS = 15886
SC Express.

SC_SERIES_DAQ = 14645
SC Series DAQ.

SWITCHES = 14650
Switches.

S_SERIES_DAQ = 14644
S Series DAQ.

TIO_SERIES = 14661
TIO Series.

UNKNOWN = 12588
Unknown category.

USBDAQ = 14646
USB DAQ.

X_SERIES_DAQ = 15858
X Series DAQ.

class nidaqmx.constants.**RTDType**
Bases: enum.Enum

CUSTOM = 10137
You must use **ai_rtd_a**, **ai_rtd_b**, and **ai_rtd_c** to supply the coefficients for the Callendar-Van Dusen equation.

PT_3750 = 12481
Pt3750.

PT_3851 = 10071
Pt3851.

PT_3911 = 12482
Pt3911.

PT_3916 = 10069
Pt3916.

PT_3920 = 10053
Pt3920.

PT_3928 = 12483
Pt3928.

class nidaqmx.constants.**RVDTSensitivityUnits**
Bases: enum.Enum

M_VPER_VPER_DEGREE = 12507
mVolts/Volt/Degree.

M_VPER_VPER_RADIAN = 12508
mVolts/Volt/Radian.

class nidaqmx.constants.**RawDataCompressionType**
Bases: enum.Enum

LOSSLESS_PACKING = 12555
Remove unused bits from samples. No resolution is lost.

LOSSY_LSB_REMOVAL = 12556
Remove unused bits from samples. Then, if necessary, remove bits from samples until the samples are the size specified with **ai_lossy_lsb_removal_compressed_samp_size**. This compression type limits resolution to the specified sample size.

NONE = 10230
Do not compress samples.

class `nidaqmx.constants.`**`ReadRelativeTo`**
Bases: `enum.Enum`

CURRENT_READ_POSITION = 10425
Start reading samples relative to the last sample returned by the previous read. For the first read operation, this position is the first sample acquired or the first pretrigger sample if you configured a reference trigger for the task.

FIRST_PRETRIGGER_SAMPLE = 10427
Start reading samples relative to the first pretrigger sample. You specify the number of pretrigger samples to acquire when you configure a reference trigger.

FIRST_SAMPLE = 10424
Start reading samples relative to the first sample acquired.

MOST_RECENT_SAMPLE = 10428
Start reading samples relative to the next sample acquired. For example, use this value and set **offset** to -1 to read the last sample acquired.

REFERENCE_TRIGGER = 10426
Start reading samples relative to the first sample after the reference trigger occurred.

class `nidaqmx.constants.`**`RegenerationMode`**
Bases: `enum.Enum`

ALLOW_REGENERATION = 10097
Allow NI-DAQmx to regenerate samples that the device previously generated. When you choose this value, the write marker returns to the beginning of the buffer after the device generates all samples currently in the buffer.

DONT_ALLOW_REGENERATION = 10158
Do not allow NI-DAQmx to regenerate samples the device previously generated. When you choose this value, NI-DAQmx waits for you to write more samples to the buffer or until the timeout expires.

class `nidaqmx.constants.`**`RelayPosition`**
Bases: `enum.Enum`

CLOSED = 10438

OPEN = 10437

class `nidaqmx.constants.`**`ResistanceConfiguration`**
Bases: `enum.Enum`

FOUR_WIRE = 4
4-wire mode.

THREE_WIRE = 3
3-wire mode.

TWO_WIRE = 2
2-wire mode.

class `nidaqmx.constants.`**`ResistanceUnits`**
Bases: `enum.Enum`

FROM_CUSTOM_SCALE = 10065
Units a custom scale specifies. If you select this value, you must specify a custom scale name.

FROM_TEDS = 12516
Units defined by TEDS information associated with the channel.

OHMS = 10384
Ohms.

class `nidaqmx.constants.`**`ResistorState`**
Bases: `enum.Enum`

PULL_DOWN = 15951
pull down state for pull up pull down resistors

PULL_UP = 15950
pull up state for pull up/pull down resistors

class `nidaqmx.constants.`**`ResolutionType`**
Bases: `enum.Enum`

BITS = 10109
Bits.

class `nidaqmx.constants.`**`SCXI1124Range`**
Bases: `enum.Enum`

NEG_10_TO_10_V = 14634

NEG_1_TO_1_V = 14632

NEG_5_TO_5_V = 14633

ZERO_TO_FIVE_V = 14630

ZERO_TO_ONE_V = 14629

ZERO_TO_TEN_V = 14631

ZERO_TO_TWENTY_M_A = 14635

class `nidaqmx.constants.`**`SampClkOverrunBehavior`**
Bases: `enum.Enum`

REPEAT_LAST_SAMPLE = 16062
Repeat the last sample.

RETURN_SENTINEL_VALUE = 16063
Return the sentinel value.

class `nidaqmx.constants.`**`SampleInputDataWhen`**
Bases: `enum.Enum`

HANDSHAKE_TRIGGER_ASSERTS = 12552
Latch data when the Handshake Trigger asserts.

HANDSHAKE_TRIGGER_DEASSERTS = 12553
Latch data when the Handshake Trigger deasserts.

class `nidaqmx.constants.`**`SampleTimingType`**
Bases: `enum.Enum`

BURST_HANDSHAKE = 12548
Determine sample timing using burst handshaking between the device and a peripheral device.

CHANGE_DETECTION = 12504
Acquire samples when a change occurs in the state of one or more digital input lines. The lines must be contained within a digital input channel.

HANDSHAKE = 10389
Determine sample timing by using digital handshaking between the device and a peripheral device.

IMPLICIT = 10451
Configure only the duration of the task.

ON_DEMAND = 10390
Acquire or generate a sample on each read or write operation. This timing type is also referred to as static or software-timed.

PIPELINED_SAMPLE_CLOCK = 14668
Device acquires or generates samples on each sample clock edge, but does not respond to certain triggers until a few sample clock edges later. Pipelining allows higher data transfer rates at the cost of increased trigger response latency. Refer to the device documentation for information about which triggers pipelining affects. This timing type allows handshaking with some devices using the Pause trigger, the Ready for Transfer event, or the Data Active event. Refer to the device documentation for more information.

SAMPLE_CLOCK = 10388
Acquire or generate samples on the specified edge of the sample clock.

class `nidaqmx.constants.`**`ScaleType`**
Bases: `enum.Enum`

LINEAR = 10447
Scale values by using the equation y=mx+b, where x is a prescaled value and y is a scaled value.

MAP_RANGES = 10448
Scale values proportionally from a range of pre-scaled values to a range of scaled values.

NONE = 10230
Do not scale electrical values to physical units.

POLYNOMIAL = 10449
Scale values by using an Nth order polynomial equation.

TABLE = 10450
Map a list of pre-scaled values to a list of corresponding scaled values, with all other values scaled proportionally.

TWO_POINT_LINEAR = 15898
You provide two pairs of electrical values and their corresponding physical values. NI-DAQmx uses those values to calculate the slope and y-intercept of a linear equation and uses that equation to scale electrical values to physical values.

class `nidaqmx.constants.`**`ScanRepeatMode`**
Bases: `enum.Enum`

CONTINUOUS = 10117
The task returns to the beginning of the scan list when it reaches the end of the scan list.

FINITE = 10172
The task advances through the scan list one time only. NI-DAQmx ignores any Advance Triggers after completing the scan list.

class `nidaqmx.constants.`**`Sense`**
Bases: `enum.Enum`

LOCAL = 16095
Local.

REMOTE = 16096
Remote.

class nidaqmx.constants.**ShuntCalSelect**
Bases: enum.Enum

A = 12513
Switch A.

AAND_B = 12515
Switches A and B.

B = 12514
Switch B.

class nidaqmx.constants.**ShuntElementLocation**
Bases: enum.Enum

NONE = 10230

R_1 = 12465

R_2 = 12466

R_3 = 12467

R_4 = 14813

class nidaqmx.constants.**ShuntResistorSelect**
Bases: enum.Enum

A = 12513
A

B = 12514
B

class nidaqmx.constants.**Signal**
Bases: enum.Enum

ADVANCE_TRIGGER = 12488

ADV_CMPLT_EVENT = 12492

AI_CONVERT_CLOCK = 12484

AI_HOLD_CMPLT_EVENT = 12493

CHANGE_DETECTION_EVENT = 12511
Timed Loop executes each time the Change Detection Event occurs.

COUNTER_OUTPUT_EVENT = 12494
Timed Loop executes each time the Counter Output Event occurs.

REFERENCE_TRIGGER = 12490

SAMPLE_CLOCK = 12487
Timed Loop executes on each active edge of the Sample Clock.

SAMPLE_COMPLETE = 12530
Timed Loop executes each time the Sample Complete Event occurs.

START_TRIGGER = 12491

TEN_M_HZ_REF_CLOCK = 12536

TWENTY_M_HZ_TIMEBASE_CLOCK = 12486

WATCHDOG_TIMER_EXPIRED_EVENT = 12512

class `nidaqmx.constants.`**`SignalModifiers`**
Bases: `enum.Enum`

DO_NOT_INVERT_POLARITY = 0
Do not invert polarity

INVERT_POLARITY = 1
Invert polarity

class `nidaqmx.constants.`**`Slope`**
Bases: `enum.Enum`

FALLING = 10171
Trigger on the falling slope of the signal.

RISING = 10280
Trigger on the rising slope of the signal.

class `nidaqmx.constants.`**`SoftwareTrigger`**
Bases: `enum.Enum`

ADVANCE_TRIGGER = 12488
Place holder enum to make editting internal enum easier.

class `nidaqmx.constants.`**`SoundPressureUnits`**
Bases: `enum.Enum`

FROM_CUSTOM_SCALE = 10065
Units a custom scale specifies. If you select this value, you must specify a custom scale name.

PA = 10081
Pascals.

class `nidaqmx.constants.`**`SourceSelection`**
Bases: `enum.Enum`

EXTERNAL = 10167
External to the device.

INTERNAL = 10200
Internal to the device.

class `nidaqmx.constants.`**`StrainGageBridgeType`**
Bases: `enum.Enum`

FULL_BRIDGE_I = 10183
Four active gages with two pairs subjected to equal and opposite strains.

FULL_BRIDGE_II = 10184
Four active gages with two aligned with maximum principal strain and two Poisson gages in adjacent arms.

FULL_BRIDGE_III = 10185
Four active gages with two aligned with maximum principal strain and two Poisson gages in opposite arms.

HALF_BRIDGE_I = 10188
Two active gages with one aligned with maximum principal strain and one Poisson gage.

HALF_BRIDGE_II = 10189
Two active gages with equal and opposite strains.

QUARTER_BRIDGE_I = 10271
Single active gage.

QUARTER_BRIDGE_II = 10272
Single active gage and one dummy gage.

class nidaqmx.constants.**StrainGageRosetteMeasurementType**
Bases: enum.Enum

CARTESIAN_SHEAR_STRAIN_XY = 15976
The tensile strain coplanar to the surface of the material under stress in the XY coordinate direction.

CARTESIAN_STRAIN_X = 15974
The tensile strain coplanar to the surface of the material under stress in the X coordinate direction.

CARTESIAN_STRAIN_Y = 15975
The tensile strain coplanar to the surface of the material under stress in the Y coordinate direction.

MAX_SHEAR_STRAIN = 15977
The maximum strain coplanar to the cross section of the material under stress.

MAX_SHEAR_STRAIN_ANGLE = 15978
The angle at which the maximum shear strain of the rosette occurs.

PRINCIPAL_STRAIN_1 = 15971
The maximum tensile strain coplanar to the surface of the material under stress.

PRINCIPAL_STRAIN_2 = 15972
The minimum tensile strain coplanar to the surface of the material under stress.

PRINCIPAL_STRAIN_ANGLE = 15973
The angle at which the principal strains of the rosette occur.

class nidaqmx.constants.**StrainGageRosetteType**
Bases: enum.Enum

DELTA = 15969
A delta rosette consists of three strain gages, each separated by a 60 degree angle.

RECTANGULAR = 15968
A rectangular rosette consists of three strain gages, each separated by a 45 degree angle.

TEE = 15970
A tee rosette consists of two gages oriented at 90 degrees with respect to each other.

class nidaqmx.constants.**StrainUnits**
Bases: enum.Enum

FROM_CUSTOM_SCALE = 10065
Units a custom scale specifies. If you select this value, you must specify a custom scale name.

STRAIN = 10299
Strain.

class nidaqmx.constants.**SwitchChannelUsage**
Bases: enum.Enum

LOAD_CHANNEL = 10440
You can use the channel only as the output for a signal passing through the switch.

RESERVED_FOR_ROUTING_CHANNEL = 10441
You can use the channel only to complete routes within a switch.

SOURCE_CHANNEL = 10439
You can use the channel only as an input for a signal.

class nidaqmx.constants.**SyncType**
Bases: enum.Enum

MASTER = 15888
Device is the source for shared clocks and triggers.

NONE = 10230
Disables trigger skew correction.

SLAVE = 15889
Device uses clocks and triggers from the master device.

class nidaqmx.constants.**TEDSUnits**
Bases: enum.Enum

FROM_CUSTOM_SCALE = 10065
Units a custom scale specifies. If you select this value, you must specify a custom scale name.

FROM_TEDS = 12516
Units defined by TEDS information associated with the channel.

class nidaqmx.constants.**TaskMode**
Bases: enum.Enum

TASK_ABORT = 6
Abort

TASK_COMMIT = 3
Commit

TASK_RESERVE = 4
Reserve

TASK_START = 0
Start

TASK_STOP = 1
Stop

TASK_UNRESERVE = 5
Unreserve

TASK_VERIFY = 2
Verify

class nidaqmx.constants.**TaskStringFormat**
Bases: enum.Enum

INI = 0

JSON = 2

TAB_DELIMITED = 1

class nidaqmx.constants.**TemperatureUnits**
Bases: enum.Enum

DEG_C = 10143
Degrees Celsius.

DEG_F = 10144
Degrees Fahrenheit.

DEG_R = 10145
Degrees Rankine.

FROM_CUSTOM_SCALE = 10065
Units a custom scale specifies. If you select this value, you must specify a custom scale name.

K = 10325
Kelvins.

class `nidaqmx.constants.`**`TerminalConfiguration`**
Bases: `enum.Enum`

DEFAULT = -1
Default.

DIFFERENTIAL = 10106
Differential.

NRSE = 10078
Non-Referenced Single-Ended.

PSEUDODIFFERENTIAL = 12529
Pseudodifferential.

RSE = 10083
Referenced Single-Ended.

class `nidaqmx.constants.`**`ThermocoupleType`**
Bases: `enum.Enum`

B = 10047
B-type thermocouple.

E = 10055
E-type thermocouple.

J = 10072
J-type thermocouple.

K = 10073
K-type thermocouple.

N = 10077
N-type thermocouple.

R = 10082
R-type thermocouple.

S = 10085
S-type thermocouple.

T = 10086
T-type thermocouple.

class `nidaqmx.constants.`**`TimeUnits`**
Bases: `enum.Enum`

FROM_CUSTOM_SCALE = 10065
Units a custom scale specifies. If you select this value, you must specify a custom scale name.

SECONDS = 10364
Seconds.

TICKS = 10304
Timebase ticks.

class nidaqmx.constants.**TorqueUnits**
Bases: enum.Enum

FOOT_POUNDS = 15884
Pound-feet.

FROM_CUSTOM_SCALE = 10065
Units a custom scale specifies. If you select this value, you must specify a custom scale name.

INCH_OUNCES = 15882
Ounce-inches.

INCH_POUNDS = 15883
Pound-inches.

NEWTON_METERS = 15881
Newton meters.

class nidaqmx.constants.**TriggerType**
Bases: enum.Enum

ANALOG_EDGE = 10099
Trigger when an analog signal crosses a threshold.

ANALOG_LEVEL = 10101
Pause the measurement or generation while an analog signal is above or below a level.

ANALOG_WINDOW = 10103
Trigger when an analog signal enters or leaves a range of values.

DIGITAL_EDGE = 10150
Trigger on a rising or falling edge of a digital pulse.

DIGITAL_LEVEL = 10152
Pause the measurement or generation while a digital signal is at either a high or low state.

DIGITAL_PATTERN = 10398
Pause the measurement or generation while digital physical channels either match or do not match a digital pattern.

INTERLOCKED = 12549
Use the Handshake Trigger as a control signal for asynchronous handshaking, such as 8255 handshaking.

NONE = 10230
Disable reference triggering for the task.

SOFTWARE = 10292
Advance to the next entry in a scan list when you call DAQmx Send Software Trigger.

class nidaqmx.constants.**TriggerUsage**
Bases: enum.Enum

ADVANCE = 12488
Advance trigger.

ARM_START = 14641
Arm Start trigger.

HANDSHAKE = 10389
Handshake trigger.

PAUSE = 12489
Pause trigger.

REFERENCE = 12490
Reference trigger.

START = 12491
Start trigger.

class `nidaqmx.constants.`**`UnderflowBehavior`**
Bases: `enum.Enum`

AUSE_UNTIL_DATA_AVAILABLE = 14616
Pause the task until samples are available in the FIFO.

HALT_OUTPUT_AND_ERROR = 14615
Stop generating samples and return an error.

class `nidaqmx.constants.`**`UnitsPreScaled`**
Bases: `enum.Enum`

AMPS = 10342
Amperes.

BAR = 15880
Bar.

COULOMBS = 16102
Coulombs.

DEGREES = 10146
Degrees.

DEGREES_PER_SECOND = 16082
Degrees per second.

DEG_C = 10143
Degrees Celsius.

DEG_F = 10144
Degrees Fahrenheit.

DEG_R = 10145
Degrees Rankine.

FOOT_POUNDS = 15884
Pound-feet.

FROM_TEDS = 12516
Units defined by TEDS information associated with the channel.

G = 10186
1 g is approximately equal to 9.81 m/s/s.

HERTZ = 10373
Hertz.

INCHES = 10379
Inches.

INCHES_PER_SECOND = 15960
Inches per second.

INCHES_PER_SECOND_SQUARED = 12471
Inches per second per second.

INCH_OUNCES = 15882
Ounce-inches.

INCH_POUNDS = 15883
Pound-inches.

K = 10325
Kelvins.

KILOGRAM_FORCE = 15877
Kilograms-force.

METERS = 10219
Meters.

METERS_PER_SECOND = 15959
Meters per second.

METERS_PER_SECOND_SQUARED = 12470
Meters per second per second.

M_VOLTS_PER_VOLT = 15897
Millivolts per volt.

NEWTONS = 15875
Newtons.

NEWTON_METERS = 15881
Newton meters.

OHMS = 10384
Ohms.

PA = 10081
Pascals.

PICO_COULOMBS = 16103
PicoCoulombs.

POUNDS = 15876
Pounds.

POUNDS_PER_SQ_INCH = 15879
Pounds per square inch.

RADIANS = 10273
Radians.

RADIANS_PER_SECOND = 16081
Radians per second.

RPM = 16080
Revolutions per minute.

SECONDS = 10364
Seconds.

STRAIN = 10299
Strain.

TICKS = 10304
Ticks.

VOLTS = 10348
Volts.

VOLTS_PER_VOLT = 15896
Volts per volt.

class nidaqmx.constants.**UsageTypeAI**
Bases: enum.Enum

ACCELERATION_4_WIRE_DC_VOLTAGE = 16106
Acceleration measurement using a 4 wire DC voltage based sensor.

ACCELERATION_ACCELEROMETER_CURRENT_INPUT = 10356
Acceleration measurement using an accelerometer.

ACCELERATION_CHARGE = 16104
Acceleration measurement using a charge-based sensor.

BRIDGE = 15908
Measure voltage ratios from a Wheatstone bridge.

CHARGE = 16105
Charge measurement.

CURRENT = 10134
Current measurement.

CURRENT_ACRMS = 10351
Current RMS measurement.

FORCE_BRIDGE = 15899
Force measurement using a bridge-based sensor.

FORCE_IEPE_SENSOR = 15895
Force measurement using an IEPE Sensor.

FREQUENCY_VOLTAGE = 10181
Frequency measurement using a frequency to voltage converter.

POSITION_ANGULAR_RVDT = 10353
Position measurement using an RVDT.

POSITION_EDDY_CURRENT_PROX_PROBE = 14835
Position measurement using an eddy current proximity probe.

POSITION_LINEAR_LVDT = 10352
Position measurement using an LVDT.

PRESSURE_BRIDGE = 15902
Pressure measurement using a bridge-based sensor.

RESISTANCE = 10278
Resistance measurement.

ROSETTE_STRAIN_GAGE = 15980
Strain measurement using a rosette strain gage.

SOUND_PRESSURE_MICROPHONE = 10354
Sound pressure measurement using a microphone.

STRAIN_STRAIN_GAGE = 10300
Strain measurement.

TEDS = 12531
Measurement type defined by TEDS.

TEMPERATURE_BUILT_IN_SENSOR = 10311
Temperature measurement using a built-in sensor on a terminal block or device. On SCXI modules, for example, this could be the CJC sensor.

TEMPERATURE_RTD = 10301
Temperature measurement using an RTD.

TEMPERATURE_THERMISTOR = 10302
Temperature measurement using a thermistor.

TEMPERATURE_THERMOCOUPLE = 10303
Temperature measurement using a thermocouple.

TORQUE_BRIDGE = 15905
Torque measurement using a bridge-based sensor.

VELOCITY_IEPE_SENSOR = 15966
Velocity measurement using an IEPE Sensor.

VOLTAGE = 10322
Voltage measurement.

VOLTAGE_ACRMS = 10350
Voltage RMS measurement.

VOLTAGE_CUSTOM_WITH_EXCITATION = 10323
Voltage measurement with an excitation source. You can use this measurement type for custom sensors that require excitation, but you must use a custom scale to scale the measured voltage.

class `nidaqmx.constants.`**`UsageTypeAO`**
Bases: `enum.Enum`

CURRENT = 10134
Current generation.

FUNCTION_GENERATION = 14750
Function generation.

VOLTAGE = 10322
Voltage generation.

class `nidaqmx.constants.`**`UsageTypeCI`**
Bases: `enum.Enum`

COUNT_EDGES = 10125
Count edges of a digital signal.

DUTY_CYCLE = 16070
Measure the duty cycle of a digital signal.

FREQUENCY = 10179
Measure the frequency of a digital signal.

PERIOD = 10256
Measure the period of a digital signal.

POSITION_ANGULAR_ENCODER = 10360
Angular position measurement using an angular encoder.

POSITION_LINEAR_ENCODER = 10361
Linear position measurement using a linear encoder.

PULSE_FREQ = 15864
Pulse measurement, returning the result as frequency and duty cycle.

PULSE_TICKS = 15866
Pulse measurement, returning the result as high ticks and low ticks.

PULSE_TIME = 15865
Pulse measurement, returning the result as high time and low time.

PULSE_WIDTH_DIGITAL = 10359
Measure the width of a pulse of a digital signal.

PULSE_WIDTH_DIGITAL_SEMI_PERIOD = 10289
Measure the time between state transitions of a digital signal.

PULSE_WIDTH_DIGITAL_TWO_EDGE_SEPARATION = 10267
Measure time between edges of two digital signals.

TIME_GPS = 10362
Timestamp measurement, synchronizing the counter to a GPS receiver.

VELOCITY_ANGULAR_ENCODER = 16078
Angular velocity measurement using an angular encoder.

VELOCITY_LINEAR_ENCODER = 16079
Linear velocity measurement using a linear encoder.

class `nidaqmx.constants.`**`UsageTypeCO`**
Bases: `enum.Enum`

PULSE_FREQUENCY = 10119
Generate digital pulses defined by frequency and duty cycle.

PULSE_TICKS = 10268
Generate digital pulses defined by the number of timebase ticks that the pulse is at a low state and the number of timebase ticks that the pulse is at a high state.

PULSE_TIME = 10269
Generate pulses defined by the time the pulse is at a low state and the time the pulse is at a high state.

class `nidaqmx.constants.`**`VelocityIEPESensorSensitivityUnits`**
Bases: `enum.Enum`

M_VOLTS_PER_INCH_PER_SECOND = 15964
Millivolts per inch per second.

M_VOLTS_PER_MILLIMETER_PER_SECOND = 15963
Millivolts per millimeter per second.

class `nidaqmx.constants.`**`VelocityUnits`**
Bases: `enum.Enum`

FROM_CUSTOM_SCALE = 10065
Units a custom scale specifies. If you select this value, you must specify a custom scale name.

INCHES_PER_SECOND = 15960
Inches per second.

METERS_PER_SECOND = 15959
Meters per second.

class `nidaqmx.constants.`**`VoltageUnits`**
Bases: `enum.Enum`

`FROM_CUSTOM_SCALE` = 10065
Units a custom scale specifies. If you select this value, you must specify a custom scale name.

`FROM_TEDS` = 12516
Units defined by TEDS information associated with the channel.

`VOLTS` = 10348
Volts.

class `nidaqmx.constants.`**`WDTTaskAction`**
Bases: `enum.Enum`

`CLEAR_EXPIRATION` = 1
Clear Expiration

`RESET_TIMER` = 0
Reset Timer

class `nidaqmx.constants.`**`WaitMode`**
Bases: `enum.Enum`

`POLL` = 12524
Repeatedly check for available samples as fast as possible. This mode allows for the highest sampling rates at the expense of CPU efficiency.

`SLEEP` = 12547
Check for available samples once per the amount of time specified in **sleep_time**.

`WAIT_FOR_INTERRUPT` = 12523
Check for available samples when the system receives an interrupt service request. This mode is the most CPU efficient, but results in lower possible sampling rates.

`YIELD` = 12525
Repeatedly check for available samples, but yield control to other threads after each check. This mode offers a balance between sampling rate and CPU efficiency.

class `nidaqmx.constants.`**`WatchdogAOExpirState`**
Bases: `enum.Enum`

`CURRENT` = 10134
Current output.

`NO_CHANGE` = 10160
Expiration does not affect the port. Do not change the state of any lines in the port, and do not lock the port.

`VOLTAGE` = 10322
Voltage output.

class `nidaqmx.constants.`**`WatchdogCOExpirState`**
Bases: `enum.Enum`

`HIGH` = 10192
High logic.

`LOW` = 10214
Low logic.

NO_CHANGE = 10160
Expiration does not affect the state of the counter output. The channels retain their states at the time of the watchdog timer expiration, and no further counter generation runs.

class `nidaqmx.constants.`**`WaveformAttributes`**
Bases: `enum.Enum`

SAMPLES_AND_TIMING = 10140
Return the samples and timing information.

SAMPLES_ONLY = 10287
Return only samples.

SAMPLES_TIMING_AND_ATTRIBUTES = 10141
Return the samples, timing information, and other attributes, such as the name of the channel.

class `nidaqmx.constants.`**`WindowTriggerCondition1`**
Bases: `enum.Enum`

ENTERING_WINDOW = 10163
Trigger when the signal enters the window.

LEAVING_WINDOW = 10208
Trigger when the signal leaves the window.

class `nidaqmx.constants.`**`WindowTriggerCondition2`**
Bases: `enum.Enum`

INSIDE_WINDOW = 10199
Pause the measurement or generation while the trigger is inside the window.

OUTSIDE_WINDOW = 10251
Pause the measurement or generation while the signal is outside the window.

class `nidaqmx.constants.`**`WriteBasicTEDSOptions`**
Bases: `enum.Enum`

DO_NOT_WRITE = 12540
blah

WRITE_TO_EEPROM = 12538
blah

WRITE_TO_PROM = 12539
blah

class `nidaqmx.constants.`**`WriteRelativeTo`**
Bases: `enum.Enum`

CURRENT_WRITE_POSITION = 10430
Write samples relative to the current position in the buffer.

FIRST_SAMPLE = 10424
Write samples relative to the first sample.

nidaqmx.errors

exception `nidaqmx.errors.`**`DaqError`**(*message*, *error_code*, *task_name=u''*)
Bases: `nidaqmx.errors.Error`

Error raised by any DAQmx method.

error_code
int – Specifies the NI-DAQmx error code.

error_type
nidaqmx.error_codes.DAQmxErrors – Specifies the NI-DAQmx error type.

exception nidaqmx.errors.**DaqWarning**(*message*, *error_code*)
Bases: exceptions.Warning

Warning raised by any NI-DAQmx method.

error_code
int – Specifies the NI-DAQmx error code.

error_type
nidaqmx.error_codes.DAQmxWarnings – Specifies the NI-DAQmx error type.

nidaqmx.errors.**DaqResourceWarning**
alias of _ResourceWarning

nidaqmx.scale

class nidaqmx.scale.**Scale**(*name*)
Bases: object

Represents a DAQmx scale.

__init__(*name*)

Parameters **name** (*str*) – Specifies the name of the scale to create.

__weakref__
list of weak references to the object (if defined)

static calculate_reverse_poly_coeff(*forward_coeffs*, *min_val_x=-5.0*, *max_val_x=5.0*, *num_points_to_compute=1000*, *reverse_poly_order=-1*)
Computes a set of coefficients for a polynomial that approximates the inverse of the polynomial with the coefficients you specify with the "forward_coeffs" input. This function generates a table of x versus y values over the range of x. This function then finds a polynomial fit, using the least squares method to compute a polynomial that computes x when given a value for y.

Parameters

- **forward_coeffs** (*List[float]*) – Is the list of coefficients for the polynomial that computes y given a value of x. Each element of the list corresponds to a term of the equation.
- **min_val_x** (*Optional[float]*) – Is the minimum value of x for which you use the polynomial. This is the smallest value of x for which the function generates a y value in the table.
- **max_val_x** (*Optional[float]*) – Is the maximum value of x for which you use the polynomial. This is the largest value of x for which the function generates a y value in the table.
- **num_points_to_compute** (*Optional[int]*) – Is the number of points in the table of x versus y values. The function spaces the values evenly between "min_val_x" and "max_val_x".

- **reverse_poly_order** (*Optional[int]*) – Is the order of the reverse polynomial to compute. For example, an input of 3 indicates a 3rd order polynomial. A value of -1 indicates a reverse polynomial of the same order as the forward polynomial.

Returns Specifies the list of coefficients for the reverse polynomial. Each element of the list corresponds to a term of the equation. For example, if index three of the list is 9, the fourth term of the equation is 9y^3.

Return type List[float]

static create_lin_scale(*scale_name*, *slope*, *y_intercept=0.0*, *pre_scaled_units=<UnitsPreScaled.VOLTS: 10348>*, *scaled_units=None*)

Creates a custom scale that uses the equation y=mx+b, where x is a pre-scaled value, and y is a scaled value. The equation is identical for input and output. If the equation is in the form x=my+b, you must first solve for y in terms of x.

Parameters

- **scale_name** (*str*) – Specifies the name of the scale to create.
- **slope** (*float*) – Is the slope, m, in the equation.
- **y_intercept** (*Optional[float]*) – Is the y-intercept, b, in the equation.
- **pre_scaled_units** (*Optional[nidaqmx.constants.UnitsPreScaled]*) – Is the units of the values to scale.
- **scaled_units** (*Optional[str]*) – Is the units to use for the scaled value. You can use an arbitrary string. NI-DAQmx uses the units to label a graph or chart.

Returns Indicates an object that represents the created custom scale.

Return type *nidaqmx.scale.Scale*

static create_map_scale(*scale_name*, *prescaled_min*, *prescaled_max*, *scaled_min*, *scaled_max*, *pre_scaled_units=<UnitsPreScaled.VOLTS: 10348>*, *scaled_units=None*)

Creates a custom scale that scales values proportionally from a range of pre-scaled values to a range of scaled values.

Parameters

- **scale_name** (*str*) – Specifies the name of the scale to create.
- **prescaled_min** (*float*) – Is the smallest value in the range of pre-scaled values. NI-DAQmx maps this value to "scaled_min".
- **prescaled_max** (*float*) – Is the largest value in the range of pre-scaled values. NI-DAQmx maps this value to "scaled_max".
- **scaled_min** (*float*) – Is the smallest value in the range of scaled values. NI-DAQmx maps this value to "prescaled_min". Read operations clip samples that are smaller than this value. Write operations generate errors for samples that are smaller than this value.
- **scaled_max** (*float*) – Is the largest value in the range of scaled values. NI-DAQmx maps this value to "prescaled_max". Read operations clip samples that are larger than this value. Write operations generate errors for samples that are larger than this value.
- **pre_scaled_units** (*Optional[nidaqmx.constants.UnitsPreScaled]*) – Is the units of the values to scale.
- **scaled_units** (*Optional[str]*) – Is the units to use for the scaled value. You can use an arbitrary string. NI-DAQmx uses the units to label a graph or chart.

Returns Indicates an object that represents the created custom scale.

Return type *nidaqmx.scale.Scale*

static **create_polynomial_scale**(*scale_name*, *forward_coeffs*, *reverse_coeffs*, *pre_scaled_units=<UnitsPreScaled.VOLTS: 10348>*, *scaled_units=None*)

Creates a custom scale that uses an nth order polynomial equation. NI-DAQmx requires both a polynomial to convert pre- scaled values to scaled values (forward) and a polynomial to convert scaled values to pre-scaled values (reverse). If you only know one set of coefficients, use the DAQmx Compute Reverse Polynomial Coefficients function to generate the other set.

Parameters

- **scale_name** (`str`) – Specifies the name of the scale to create.
- **forward_coeffs** (`List[float]`) – Is an list of coefficients for the polynomial that converts pre-scaled values to scaled values. Each element of the list corresponds to a term of the equation.
- **reverse_coeffs** (`List[float]`) – Is an list of coefficients for the polynomial that converts scaled values to pre-scaled values. Each element of the list corresponds to a term of the equation.
- **pre_scaled_units** (`Optional[nidaqmx.constants.UnitsPreScaled]`) – Is the units of the values to scale.
- **scaled_units** (`Optional[str]`) – Is the units to use for the scaled value. You can use an arbitrary string. NI-DAQmx uses the units to label a graph or chart.

Returns Indicates an object that represents the created custom scale.

Return type *nidaqmx.scale.Scale*

static **create_table_scale**(*scale_name*, *prescaled_vals*, *scaled_vals*, *pre_scaled_units=<UnitsPreScaled.VOLTS: 10348>*, *scaled_units=None*)

Creates a custom scale that maps an list of pre-scaled values to an list of corresponding scaled values. NI-DAQmx applies linear interpolation to values that fall between the values in the table. Read operations clip scaled samples that are outside the maximum and minimum scaled values found in the table. Write operations generate errors for samples that are outside the minimum and maximum scaled values found in the table.

Parameters

- **scale_name** (`str`) – Specifies the name of the scale to create.
- **prescaled_vals** (`List[float]`) – Is the list of pre-scaled values that map to the values in "scaled_vals".
- **scaled_vals** (`List[float]`) – Is the list of scaled values that map to the values in "prescaled_vals".
- **pre_scaled_units** (`Optional[nidaqmx.constants.UnitsPreScaled]`) – Is the units of the values to scale.
- **scaled_units** (`Optional[str]`) – Is the units to use for the scaled value. You can use an arbitrary string. NI-DAQmx uses the units to label a graph or chart.

Returns Indicates an object that represents the created custom scale.

Return type *nidaqmx.scale.Scale*

description

str – Specifies a description for the scale.

lin_slope
float – Specifies the slope, m, in the equation y=mx+b.

lin_y_intercept
float – Specifies the y-intercept, b, in the equation y=mx+b.

map_pre_scaled_max
float – Specifies the largest value in the range of pre-scaled values. NI-DAQmx maps this value to **map_scaled_max**.

map_pre_scaled_min
float – Specifies the smallest value in the range of pre-scaled values. NI-DAQmx maps this value to **map_scaled_min**.

map_scaled_max
float – Specifies the largest value in the range of scaled values. NI-DAQmx maps this value to **map_pre_scaled_max**. Reads coerce samples that are larger than this value to match this value. Writes generate errors for samples that are larger than this value.

map_scaled_min
float – Specifies the smallest value in the range of scaled values. NI-DAQmx maps this value to **map_pre_scaled_min**. Reads coerce samples that are smaller than this value to match this value. Writes generate errors for samples that are smaller than this value.

name
str – Specifies the name of this scale.

poly_forward_coeff
List[float] – Specifies a list of coefficients for the polynomial that converts pre-scaled values to scaled values. Each element of the list corresponds to a term of the equation. For example, if index three of the list is 9, the fourth term of the equation is 9x^3.

poly_reverse_coeff
List[float] – Specifies a list of coefficients for the polynomial that converts scaled values to pre-scaled values. Each element of the list corresponds to a term of the equation. For example, if index three of the list is 9, the fourth term of the equation is 9y^3.

pre_scaled_units
`nidaqmx.constants.UnitsPreScaled` – Specifies the units of the values that you want to scale.

save(*save_as=u''*, *author=u''*, *overwrite_existing_scale=False*, *allow_interactive_editing=True*, *allow_interactive_deletion=True*)
Saves this custom scale to MAX.

Parameters

- **save_as** (`Optional[str]`) – Is the name to save the task, global channel, or custom scale as. If you do not specify a value for this input, NI-DAQmx uses the name currently assigned to the task, global channel, or custom scale.
- **author** (`Optional[str]`) – Is a name to store with the task, global channel, or custom scale.
- **options** (`Optional[int]`) – Specifies whether to allow the task, global channel, or custom scale to be deleted through MAX.
- **overwrite_existing_scale** (`Optional[bool]`) – Specifies whether to overwrite a custom scale of the same name if one is already saved in MAX. If this input is False and a custom scale of the same name is already saved in MAX, this function returns an error.

- **allow_interactive_editing** (*Optional[bool]*) – Specifies whether to allow the task, global channel, or custom scale to be edited in the DAQ Assistant. If allow_interactive_editing is True, the DAQ Assistant must support all task or global channel settings.
- **allow_interactive_deletion** (*Optional[bool]*) – Specifies whether to allow the task, global channel, or custom scale to be deleted through MAX.

scale_type
nidaqmx.constants.ScaleType – Indicates the method or equation form that the custom scale uses.

scaled_units
str – Specifies the units to use for scaled values. You can use an arbitrary string.

table_pre_scaled_vals
List[float] – Specifies a list of pre-scaled values. These values map directly to the values in **table_scaled_vals**.

table_scaled_vals
List[float] – Specifies a list of scaled values. These values map directly to the values in **table_pre_scaled_vals**.

nidaqmx.stream_readers

class nidaqmx.stream_readers.**AnalogSingleChannelReader**(*task_in_stream*)
Bases: nidaqmx.stream_readers.ChannelReaderBase

Reads samples from an analog input channel in an NI-DAQmx task.

read_many_sample(*data*, *number_of_samples_per_channel=-1*, *timeout=10.0*)
Reads one or more floating-point samples from a single analog input channel in a task.

This read method accepts a preallocated NumPy array to hold the samples requested, which can be advantageous for performance and interoperability with NumPy and SciPy.

Passing in a preallocated array is valuable in continuous acquisition scenarios, where the same array can be used repeatedly in each call to the method.

Parameters

- **data** (*numpy.ndarray*) – Specifies a preallocated 1D NumPy array of floating-point values to hold the samples requested.

  Each element in the array corresponds to a sample from the channel. The size of the array must be large enough to hold all requested samples from the channel in the task; otherwise, an error is thrown.

- **number_of_samples_per_channel** (*Optional[int]*) – Specifies the number of samples to read.

  If you set this input to nidaqmx.constants. READ_ALL_AVAILABLE, NI-DAQmx determines how many samples to read based on if the task acquires samples continuously or acquires a finite number of samples.

  If the task acquires samples continuously and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, this method reads all the samples currently available in the buffer.

If the task acquires a finite number of samples and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, the method waits for the task to acquire all requested samples, then reads those samples. If you set the "read_all_avail_samp" property to True, the method reads the samples currently available in the buffer and does not wait for the task to acquire all requested samples.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

Returns Indicates the number of samples acquired by each channel. NI-DAQmx returns a single value because this value is the same for all channels.

Return type int

read_one_sample(*timeout=10*)

Reads a single floating-point sample from a single analog input channel in a task.

Parameters **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

Returns Indicates a single floating-point sample from the task.

Return type float

verify_array_shape

bool – Indicates whether the size and shape of the user-defined NumPy arrays passed to read methods are verified. Defaults to True when this object is instantiated.

Setting this property to True may marginally adversely impact the performance of read methods.

class nidaqmx.stream_readers.**AnalogMultiChannelReader**(*task_in_stream*)

Bases: nidaqmx.stream_readers.ChannelReaderBase

Reads samples from one or more analog input channels in an NI-DAQmx task.

read_many_sample(*data*, *number_of_samples_per_channel=-1*, *timeout=10.0*)

Reads one or more floating-point samples from one or more analog input channels in a task.

This read method accepts a preallocated NumPy array to hold the samples requested, which can be advantageous for performance and interoperability with NumPy and SciPy.

Passing in a preallocated array is valuable in continuous acquisition scenarios, where the same array can be used repeatedly in each call to the method.

Parameters

- **data** (*numpy.ndarray*) – Specifies a preallocated 2D NumPy array of floating-point values to hold the samples requested. The size of the array must be large enough to hold all requested samples from all channels in the task; otherwise, an error is thrown.

  Each row corresponds to a channel in the task. Each column corresponds to a sample from each channel. The order of the channels in the array corresponds to the order in which you add the channels to the task or to the order of the channels you specify with the "channels_to_read" property.

If the size of the array is too large or the array is shaped incorrectly, the previous statement may not hold true as the samples read may not be separated into rows and columns properly. Set the "verify_array_shape" property on this channel reader object to True to validate that the NumPy array object is shaped properly. Setting this property to True may marginally adversely impact the performance of the method.

- **number_of_samples_per_channel** (*Optional[int]*) – Specifies the number of samples to read.

  If you set this input to nidaqmx.constants. READ_ALL_AVAILABLE, NI-DAQmx determines how many samples to read based on if the task acquires samples continuously or acquires a finite number of samples.

  If the task acquires samples continuously and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, this method reads all the samples currently available in the buffer.

  If the task acquires a finite number of samples and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, the method waits for the task to acquire all requested samples, then reads those samples. If you set the "read_all_avail_samp" property to True, the method reads the samples currently available in the buffer and does not wait for the task to acquire all requested samples.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

Returns Indicates the number of samples acquired by each channel. NI-DAQmx returns a single value because this value is the same for all channels.

Return type int

read_one_sample(*data*, *timeout=10*)

Reads a single floating-point sample from one or more analog input channels in a task.

This read method accepts a preallocated NumPy array to hold the samples requested, which can be advantageous for performance and interoperability with NumPy and SciPy.

Passing in a preallocated array is valuable in continuous acquisition scenarios, where the same array can be used repeatedly in each call to the method.

Parameters

- **data** (*numpy.ndarray*) – Specifies a preallocated 1D NumPy array of floating-point values to hold the samples requested.

  Each element in the array corresponds to a sample from each channel. The size of the array must be large enough to hold all requested samples from the channel in the task; otherwise, an error is thrown.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

verify_array_shape

bool – Indicates whether the size and shape of the user-defined NumPy arrays passed to read methods are verified. Defaults to True when this object is instantiated.

Setting this property to True may marginally adversely impact the performance of read methods.

class nidaqmx.stream_readers.**AnalogUnscaledReader**(*task_in_stream*)

Bases: nidaqmx.stream_readers.ChannelReaderBase

Reads unscaled samples from one or more analog input channels in an NI-DAQmx task.

read_int16(*data*, *number_of_samples_per_channel=-1*, *timeout=10.0*)

Reads one or more unscaled 16-bit integer samples from one or more analog input channels in a task.

This read method accepts a preallocated NumPy array to hold the samples requested, which can be advantageous for performance and interoperability with NumPy and SciPy.

Passing in a preallocated array is valuable in continuous acquisition scenarios, where the same array can be used repeatedly in each call to the method.

Parameters

- **data** (*numpy.ndarray*) – Specifies a preallocated 2D NumPy array of unscaled 16-bit integer values to hold the samples requested. The size of the array must be large enough to hold all requested samples from all channels in the task; otherwise, an error is thrown.

  Each row corresponds to a channel in the task. Each column corresponds to a sample from each channel. The order of the channels in the array corresponds to the order in which you add the channels to the task or to the order of the channels you specify with the "channels_to_read" property.

  If the size of the array is too large or the array is shaped incorrectly, the previous statement may not hold true as the samples read may not be separated into rows and columns properly. Set the "verify_array_shape" property on this channel reader object to True to validate that the NumPy array object is shaped properly. Setting this property may marginally adversely impact the performance of the method.

- **number_of_samples_per_channel** (*Optional[int]*) – Specifies the number of samples to read.

  If you set this input to nidaqmx.constants. READ_ALL_AVAILABLE, NI-DAQmx determines how many samples to read based on if the task acquires samples continuously or acquires a finite number of samples.

  If the task acquires samples continuously and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, this method reads all the samples currently available in the buffer.

  If the task acquires a finite number of samples and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, the method waits for the task to acquire all requested samples, then reads those samples. If you set the "read_all_avail_samp" property to True, the method reads the samples currently available in the buffer and does not wait for the task to acquire all requested samples.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

Returns Indicates the number of samples acquired by each channel. NI-DAQmx returns a single value because this value is the same for all channels.

Return type int

read_int32(*data*, *number_of_samples_per_channel=-1*, *timeout=10.0*)

Reads one or more unscaled 32-bit integer samples from one or more analog input channels in a task.

This read method accepts a preallocated NumPy array to hold the samples requested, which can be advantageous for performance and interoperability with NumPy and SciPy.

Passing in a preallocated array is valuable in continuous acquisition scenarios, where the same array can be used repeatedly in each call to the method.

Parameters

- **data** (*numpy.ndarray*) – Specifies a preallocated 2D NumPy array of unscaled 32-bit integer values to hold the samples requested. The size of the array must be large enough to hold all requested samples from all channels in the task; otherwise, an error is thrown.

  Each row corresponds to a channel in the task. Each column corresponds to a sample from each channel. The order of the channels in the array corresponds to the order in which you add the channels to the task or to the order of the channels you specify with the "channels_to_read" property.

  If the size of the array is too large or the array is shaped incorrectly, the previous statement may not hold true as the samples read may not be separated into rows and columns properly. Set the "verify_array_shape" property on this channel reader object to True to validate that the NumPy array object is shaped properly. Setting this property may marginally adversely impact the performance of the method.

- **number_of_samples_per_channel** (*Optional[int]*) – Specifies the number of samples to read.

  If you set this input to nidaqmx.constants. READ_ALL_AVAILABLE, NI-DAQmx determines how many samples to read based on if the task acquires samples continuously or acquires a finite number of samples.

  If the task acquires samples continuously and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, this method reads all the samples currently available in the buffer.

  If the task acquires a finite number of samples and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, the method waits for the task to acquire all requested samples, then reads those samples. If you set the "read_all_avail_samp" property to True, the method reads the samples currently available in the buffer and does not wait for the task to acquire all requested samples.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

Returns Indicates the number of samples acquired by each channel. NI-DAQmx returns a single value because this value is the same for all channels.

Return type int

read_uint16(*data*, *number_of_samples_per_channel=-1*, *timeout=10.0*)

Reads one or more unscaled 16-bit unsigned integer samples from one or more analog input channels in a task.

This read method accepts a preallocated NumPy array to hold the samples requested, which can be advantageous for performance and interoperability with NumPy and SciPy.

Passing in a preallocated array is valuable in continuous acquisition scenarios, where the same array can be used repeatedly in each call to the method.

Parameters

- **data** (*numpy.ndarray*) – Specifies a preallocated 2D NumPy array of unscaled 16-bit unsigned integer values to hold the samples requested. The size of the array must be large enough to hold all requested samples from all channels in the task; otherwise, an error is thrown.

  Each row corresponds to a channel in the task. Each column corresponds to a sample from each channel. The order of the channels in the array corresponds to the order in which you add the channels to the task or to the order of the channels you specify with the "channels_to_read" property.

  If the size of the array is too large or the array is shaped incorrectly, the previous statement may not hold true as the samples read may not be separated into rows and columns properly. Set the "verify_array_shape" property on this channel reader object to True to validate that the NumPy array object is shaped properly. Setting this property may marginally adversely impact the performance of the method.

- **number_of_samples_per_channel** (*Optional[int]*) – Specifies the number of samples to read.

  If you set this input to nidaqmx.constants. READ_ALL_AVAILABLE, NI-DAQmx determines how many samples to read based on if the task acquires samples continuously or acquires a finite number of samples.

  If the task acquires samples continuously and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, this method reads all the samples currently available in the buffer.

  If the task acquires a finite number of samples and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, the method waits for the task to acquire all requested samples, then reads those samples. If you set the "read_all_avail_samp" property to True, the method reads the samples currently available in the buffer and does not wait for the task to acquire all requested samples.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

Returns Indicates the number of samples acquired by each channel. NI-DAQmx returns a single value because this value is the same for all channels.

Return type int

read_uint32(*data*, *number_of_samples_per_channel=-1*, *timeout=10.0*)

Reads one or more unscaled unsigned 32-bit integer samples from one or more analog input channels in a task.

This read method accepts a preallocated NumPy array to hold the samples requested, which can be advantageous for performance and interoperability with NumPy and SciPy.

Passing in a preallocated array is valuable in continuous acquisition scenarios, where the same array can be used repeatedly in each call to the method.

Parameters

- **`data`** (*`numpy.ndarray`*) – Specifies a preallocated 2D NumPy array of unscaled 32-bit unsigned integer values to hold the samples requested. The size of the array must be large enough to hold all requested samples from all channels in the task; otherwise, an error is thrown.

  Each row corresponds to a channel in the task. Each column corresponds to a sample from each channel. The order of the channels in the array corresponds to the order in which you add the channels to the task or to the order of the channels you specify with the "channels_to_read" property.

  If the size of the array is too large or the array is shaped incorrectly, the previous statement may not hold true as the samples read may not be separated into rows and columns properly. Set the "verify_array_shape" property on this channel reader object to True to validate that the NumPy array object is shaped properly. Setting this property may marginally adversely impact the performance of the method.

- **`number_of_samples_per_channel`** (*`Optional[int]`*) – Specifies the number of samples to read.

  If you set this input to nidaqmx.constants. READ_ALL_AVAILABLE, NI-DAQmx determines how many samples to read based on if the task acquires samples continuously or acquires a finite number of samples.

  If the task acquires samples continuously and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, this method reads all the samples currently available in the buffer.

  If the task acquires a finite number of samples and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, the method waits for the task to acquire all requested samples, then reads those samples. If you set the "read_all_avail_samp" property to True, the method reads the samples currently available in the buffer and does not wait for the task to acquire all requested samples.

- **`timeout`** (*`Optional[float]`*) – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

Returns Indicates the number of samples acquired by each channel. NI-DAQmx returns a single value because this value is the same for all channels.

Return type int

`verify_array_shape`

bool – Indicates whether the size and shape of the user-defined NumPy arrays passed to read methods are verified. Defaults to True when this object is instantiated.

Setting this property to True may marginally adversely impact the performance of read methods.

class `nidaqmx.stream_readers.`**`CounterReader`**(*task_in_stream*)

Bases: `nidaqmx.stream_readers.ChannelReaderBase`

Reads samples from a counter input channel in an NI-DAQmx task.

read_many_sample_double (*data*, *number_of_samples_per_channel=-1*, *timeout=10.0*)

Reads one or more floating-point samples from a single counter input channel in a task.

This read method accepts a preallocated NumPy array to hold the samples requested, which can be advantageous for performance and interoperability with NumPy and SciPy.

Passing in a preallocated array is valuable in continuous acquisition scenarios, where the same array can be used repeatedly in each call to the method.

Parameters

- **data** (*numpy.ndarray*) – Specifies a preallocated 1D NumPy array of floating-point values to hold the samples requested.

  Each element in the array corresponds to a sample from the channel. The size of the array must be large enough to hold all requested samples from the channel in the task; otherwise, an error is thrown.

- **number_of_samples_per_channel** (*Optional[int]*) – Specifies the number of samples to read.

  If you set this input to nidaqmx.constants. READ_ALL_AVAILABLE, NI-DAQmx determines how many samples to read based on if the task acquires samples continuously or acquires a finite number of samples.

  If the task acquires samples continuously and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, this method reads all the samples currently available in the buffer.

  If the task acquires a finite number of samples and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, the method waits for the task to acquire all requested samples, then reads those samples. If you set the "read_all_avail_samp" property to True, the method reads the samples currently available in the buffer and does not wait for the task to acquire all requested samples.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

Returns Indicates the number of samples acquired by each channel. NI-DAQmx returns a single value because this value is the same for all channels.

Return type int

read_many_sample_pulse_frequency (*frequencies*, *duty_cycles*, *number_of_samples_per_channel=-1*, *timeout=10.0*)

Reads one or more pulse samples in terms of frequency from a single counter input channel in a task.

This read method accepts preallocated NumPy arrays to hold the samples requested, which can be advantageous for performance and interoperability with NumPy and SciPy.

Passing in preallocated arrays is valuable in continuous acquisition scenarios, where the same array can be used repeatedly in each call to the method.

Parameters

- **frequencies** (*numpy.ndarray*) – Specifies a preallocated 1D NumPy array of floating-point values to hold the frequency portion of the pulse samples requested.

Each element in the array corresponds to a sample from the channel. The size of the array must be large enough to hold all requested samples from the channel in the task; otherwise, an error is thrown.

- **duty_cycles** (*numpy.ndarray*) – Specifies a preallocated 1D NumPy array of floating-point values to hold the duty cycle portion of the pulse samples requested.

  Each element in the array corresponds to a sample from the channel. The size of the array must be large enough to hold all requested samples from the channel in the task; otherwise, an error is thrown.

- **number_of_samples_per_channel** (*Optional[int]*) – Specifies the number of samples to read.

  If you set this input to nidaqmx.constants. READ_ALL_AVAILABLE, NI-DAQmx determines how many samples to read based on if the task acquires samples continuously or acquires a finite number of samples.

  If the task acquires samples continuously and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, this method reads all the samples currently available in the buffer.

  If the task acquires a finite number of samples and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, the method waits for the task to acquire all requested samples, then reads those samples. If you set the "read_all_avail_samp" property to True, the method reads the samples currently available in the buffer and does not wait for the task to acquire all requested samples.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

Returns Indicates the number of samples acquired by each channel. NI-DAQmx returns a single value because this value is the same for all channels.

Return type int

read_many_sample_pulse_ticks(*high_ticks*, *low_ticks*, *number_of_samples_per_channel=-1*, *timeout=10.0*)

Reads one or more pulse samples in terms of ticks from a single counter input channel in a task.

This read method accepts preallocated NumPy arrays to hold the samples requested, which can be advantageous for performance and interoperability with NumPy and SciPy.

Passing in preallocated arrays is valuable in continuous acquisition scenarios, where the same array can be used repeatedly in each call to the method.

Parameters

- **high_ticks** (*numpy.ndarray*) – Specifies a preallocated 1D NumPy array of 32-bit unsigned integer values to hold the high ticks portion of the pulse samples requested.

  Each element in the array corresponds to a sample from the channel. The size of the array must be large enough to hold all requested samples from the channel in the task; otherwise, an error is thrown.

- **low_ticks** (*numpy.ndarray*) – Specifies a preallocated 1D NumPy array of 32-bit unsigned integer values to hold the low ticks portion of the pulse samples requested.

Each element in the array corresponds to a sample from the channel. The size of the array must be large enough to hold all requested samples from the channel in the task; otherwise, an error is thrown.

- **number_of_samples_per_channel** (*Optional[int]*) – Specifies the number of samples to read.

  If you set this input to nidaqmx.constants. READ_ALL_AVAILABLE, NI-DAQmx determines how many samples to read based on if the task acquires samples continuously or acquires a finite number of samples.

  If the task acquires samples continuously and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, this method reads all the samples currently available in the buffer.

  If the task acquires a finite number of samples and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, the method waits for the task to acquire all requested samples, then reads those samples. If you set the "read_all_avail_samp" property to True, the method reads the samples currently available in the buffer and does not wait for the task to acquire all requested samples.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

Returns Indicates the number of samples acquired by each channel. NI-DAQmx returns a single value because this value is the same for all channels.

Return type int

read_many_sample_pulse_time(*high_times*, *low_times*, *number_of_samples_per_channel=-1*, *timeout=10.0*)

Reads one or more pulse samples in terms of time from a single counter input channel in a task.

This read method accepts preallocated NumPy arrays to hold the samples requested, which can be advantageous for performance and interoperability with NumPy and SciPy.

Passing in preallocated arrays is valuable in continuous acquisition scenarios, where the same array can be used repeatedly in each call to the method.

Parameters

- **high_times** (*numpy.ndarray*) – Specifies a preallocated 1D NumPy array of floating-point values to hold the high time portion of the pulse samples requested.

  Each element in the array corresponds to a sample from the channel. The size of the array must be large enough to hold all requested samples from the channel in the task; otherwise, an error is thrown.

- **low_times** (*numpy.ndarray*) – Specifies a preallocated 1D NumPy array of floating-point values to hold the low time portion of the pulse samples requested.

  Each element in the array corresponds to a sample from the channel. The size of the array must be large enough to hold all requested samples from the channel in the task; otherwise, an error is thrown.

- **number_of_samples_per_channel** (*Optional[int]*) – Specifies the number of samples to read.

If you set this input to nidaqmx.constants. READ_ALL_AVAILABLE, NI-DAQmx determines how many samples to read based on if the task acquires samples continuously or acquires a finite number of samples.

If the task acquires samples continuously and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, this method reads all the samples currently available in the buffer.

If the task acquires a finite number of samples and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, the method waits for the task to acquire all requested samples, then reads those samples. If you set the "read_all_avail_samp" property to True, the method reads the samples currently available in the buffer and does not wait for the task to acquire all requested samples.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

Returns Indicates the number of samples acquired by each channel. NI-DAQmx returns a single value because this value is the same for all channels.

Return type int

read_many_sample_uint32 (*data*, *number_of_samples_per_channel=-1*, *timeout=10.0*)

Reads one or more 32-bit unsigned integer samples from a single counter input channel in a task.

This read method accepts a preallocated NumPy array to hold the samples requested, which can be advantageous for performance and interoperability with NumPy and SciPy.

Passing in a preallocated array is valuable in continuous acquisition scenarios, where the same array can be used repeatedly in each call to the method.

Parameters

- **data** (*numpy.ndarray*) – Specifies a preallocated 1D NumPy array of 32-bit unsigned integer values to hold the samples requested.

  Each element in the array corresponds to a sample from the channel. The size of the array must be large enough to hold all requested samples from the channel in the task; otherwise, an error is thrown.

- **number_of_samples_per_channel** (*Optional[int]*) – Specifies the number of samples to read.

  If you set this input to nidaqmx.constants. READ_ALL_AVAILABLE, NI-DAQmx determines how many samples to read based on if the task acquires samples continuously or acquires a finite number of samples.

  If the task acquires samples continuously and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, this method reads all the samples currently available in the buffer.

  If the task acquires a finite number of samples and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, the method waits for the task to acquire all requested samples, then reads those samples. If you set the "read_all_avail_samp" property to True, the method reads the samples currently available in the buffer and does not wait for the task to acquire all requested samples.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

Returns Indicates the number of samples acquired by each channel. NI-DAQmx returns a single value because this value is the same for all channels.

Return type int

read_one_sample_double(*timeout=10*)

Reads a single floating-point sample from a single counter input channel in a task.

Parameters **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

Returns

Indicates a single floating-point sample from the task.

Return type float

read_one_sample_pulse_frequency(*timeout=10*)

Reads a pulse sample in terms of frequency from a single counter input channel in a task.

Parameters **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

Returns Indicates a pulse sample in terms of frequency from the task.

Return type *nidaqmx.types.CtrFreq*

read_one_sample_pulse_ticks(*timeout=10*)

Reads a pulse sample in terms of ticks from a single counter input channel in a task.

Parameters **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

Returns Indicates a pulse sample in terms of ticks from the task.

Return type *nidaqmx.types.CtrTick*

read_one_sample_pulse_time(*timeout=10*)

Reads a pulse sample in terms of time from a single counter input channel in a task.

Parameters **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

Returns Indicates a pulse sample in terms of time from the task.

Return type *nidaqmx.types.CtrTime*

read_one_sample_uint32(*timeout=10*)

Reads a single 32-bit unsigned integer sample from a single counter input channel in a task.

Parameters **timeout** (`Optional[float]`) – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

Returns Indicates a single 32-bit unsigned integer sample from the task.

Return type int

verify_array_shape

bool – Indicates whether the size and shape of the user-defined NumPy arrays passed to read methods are verified. Defaults to True when this object is instantiated.

Setting this property to True may marginally adversely impact the performance of read methods.

class `nidaqmx.stream_readers.`**DigitalSingleChannelReader**(*task_in_stream*)

Bases: `nidaqmx.stream_readers.ChannelReaderBase`

Reads samples from a digital input channel in an NI-DAQmx task.

read_many_sample_port_byte(*data*, *number_of_samples_per_channel=-1*, *timeout=10.0*)

Reads one or more 8-bit unsigned integer samples from a single digital input channel in a task.

Use this method for devices with up to 8 lines per port.

This read method accepts a preallocated NumPy array to hold the samples requested, which can be advantageous for performance and interoperability with NumPy and SciPy.

Passing in a preallocated array is valuable in continuous acquisition scenarios, where the same array can be used repeatedly in each call to the method.

Parameters

- **data** (`numpy.ndarray`) – Specifies a preallocated 1D NumPy array of 8-bit unsigned integer values to hold the samples requested.

  Each element in the array corresponds to a sample from the channel. The size of the array must be large enough to hold all requested samples from the channel in the task; otherwise, an error is thrown.

- **number_of_samples_per_channel** (`Optional[int]`) – Specifies the number of samples to read.

  If you set this input to nidaqmx.constants. READ_ALL_AVAILABLE, NI-DAQmx determines how many samples to read based on if the task acquires samples continuously or acquires a finite number of samples.

  If the task acquires samples continuously and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, this method reads all the samples currently available in the buffer.

If the task acquires a finite number of samples and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, the method waits for the task to acquire all requested samples, then reads those samples. If you set the "read_all_avail_samp" property to True, the method reads the samples currently available in the buffer and does not wait for the task to acquire all requested samples.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

Returns Indicates the number of samples acquired by each channel. NI-DAQmx returns a single value because this value is the same for all channels.

Return type int

read_many_sample_port_uint16(*data*, *number_of_samples_per_channel=-1*, *timeout=10.0*)

Reads one or more 16-bit unsigned integer samples from a single digital input channel in a task.

Use this method for devices with up to 16 lines per port.

This read method accepts a preallocated NumPy array to hold the samples requested, which can be advantageous for performance and interoperability with NumPy and SciPy.

Passing in a preallocated array is valuable in continuous acquisition scenarios, where the same array can be used repeatedly in each call to the method.

Parameters

- **data** (*numpy.ndarray*) – Specifies a preallocated 1D NumPy array of 16-bit unsigned integer values to hold the samples requested.

  Each element in the array corresponds to a sample from the channel. The size of the array must be large enough to hold all requested samples from the channel in the task; otherwise, an error is thrown.

- **number_of_samples_per_channel** (*Optional[int]*) – Specifies the number of samples to read.

  If you set this input to nidaqmx.constants. READ_ALL_AVAILABLE, NI-DAQmx determines how many samples to read based on if the task acquires samples continuously or acquires a finite number of samples.

  If the task acquires samples continuously and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, this method reads all the samples currently available in the buffer.

  If the task acquires a finite number of samples and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, the method waits for the task to acquire all requested samples, then reads those samples. If you set the "read_all_avail_samp" property to True, the method reads the samples currently available in the buffer and does not wait for the task to acquire all requested samples.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

Returns Indicates the number of samples acquired by each channel. NI-DAQmx returns a single value because this value is the same for all channels.

Return type int

read_many_sample_port_uint32 (*data*, *number_of_samples_per_channel=-1*, *timeout=10.0*)

Reads one or more 32-bit unsigned integer samples from a single digital input channel in a task.

Use this method for devices with up to 32 lines per port.

This read method accepts a preallocated NumPy array to hold the samples requested, which can be advantageous for performance and interoperability with NumPy and SciPy.

Passing in a preallocated array is valuable in continuous acquisition scenarios, where the same array can be used repeatedly in each call to the method.

Parameters

- **data** (*numpy.ndarray*) – Specifies a preallocated 1D NumPy array of 32-bit unsigned integer values to hold the samples requested.

  Each element in the array corresponds to a sample from the channel. The size of the array must be large enough to hold all requested samples from the channel in the task; otherwise, an error is thrown.

- **number_of_samples_per_channel** (*Optional[int]*) – Specifies the number of samples to read.

  If you set this input to nidaqmx.constants. READ_ALL_AVAILABLE, NI-DAQmx determines how many samples to read based on if the task acquires samples continuously or acquires a finite number of samples.

  If the task acquires samples continuously and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, this method reads all the samples currently available in the buffer.

  If the task acquires a finite number of samples and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, the method waits for the task to acquire all requested samples, then reads those samples. If you set the "read_all_avail_samp" property to True, the method reads the samples currently available in the buffer and does not wait for the task to acquire all requested samples.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

Returns Indicates the number of samples acquired by each channel. NI-DAQmx returns a single value because this value is the same for all channels.

Return type int

read_one_sample_multi_line (*data*, *timeout=10*)

Reads a single boolean sample from a single digital input channel in a task. The channel can contain multiple digital lines.

This read method accepts a preallocated NumPy array to hold the samples requested, which can be advantageous for performance and interoperability with NumPy and SciPy.

Passing in a preallocated array is valuable in continuous acquisition scenarios, where the same array can be used repeatedly in each call to the method.

Parameters

- **data** (*numpy.ndarray*) – Specifies a preallocated 1D NumPy array of boolean values to hold the samples requested.

  Each element in the array corresponds to a sample from a line in the channel. The size of the array must be large enough to hold all requested samples from the channel in the task; otherwise, an error is thrown.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

read_one_sample_one_line(*timeout=10*)

Reads a single boolean sample from a single digital input channel in a task. The channel can contain only one digital line.

Parameters **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

Returns Indicates a single boolean sample from the task.

Return type bool

read_one_sample_port_byte(*timeout=10*)

Reads a single 8-bit unsigned integer sample from a single digital input channel in a task.

Use this method for devices with up to 8 lines per port.

Parameters **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

Returns Indicates a single 8-bit unsigned integer sample from the task.

Return type int

read_one_sample_port_uint16(*timeout=10*)

Reads a single 16-bit unsigned integer sample from a single digital input channel in a task.

Use this method for devices with up to 16 lines per port.

Parameters **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

Returns Indicates a single 16-bit unsigned integer sample from the task.

Return type int

read_one_sample_port_uint32(*timeout=10*)

Reads a single 32-bit unsigned integer sample from a single digital input channel in a task.

Use this method for devices with up to 32 lines per port.

Parameters **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

Returns Indicates a single 32-bit unsigned integer sample from the task.

Return type int

verify_array_shape

bool – Indicates whether the size and shape of the user-defined NumPy arrays passed to read methods are verified. Defaults to True when this object is instantiated.

Setting this property to True may marginally adversely impact the performance of read methods.

class nidaqmx.stream_readers.**DigitalMultiChannelReader**(*task_in_stream*)

Bases: nidaqmx.stream_readers.ChannelReaderBase

Reads samples from one or more digital input channels in an NI-DAQmx task.

read_many_sample_port_byte(*data*, *number_of_samples_per_channel=-1*, *timeout=10.0*)

Reads one or more 8-bit unsigned integer samples from one or more digital input channel in a task.

Use this method for devices with up to 8 lines per port.

This read method accepts a preallocated NumPy array to hold the samples requested, which can be advantageous for performance and interoperability with NumPy and SciPy.

Passing in a preallocated array is valuable in continuous acquisition scenarios, where the same array can be used repeatedly in each call to the method.

Parameters

- **data** (*numpy.ndarray*) – Specifies a preallocated 2D NumPy array of 8-bit unsigned integer values to hold the samples requested. The size of the array must be large enough to hold all requested samples from all channels in the task; otherwise, an error is thrown.

  Each row corresponds to a channel in the task. Each column corresponds to a sample from each channel. The order of the channels in the array corresponds to the order in which you add the channels to the task or to the order of the channels you specify with the "channels_to_read" property.

  If the size of the array is too large or the array is shaped incorrectly, the previous statement may not hold true as the samples read may not be separated into rows and columns properly. Set the "verify_array_shape" property on this channel reader object to True to validate that the NumPy array object is shaped properly. Setting this property may marginally adversely impact the performance of the method.

- **number_of_samples_per_channel** (*Optional[int]*) – Specifies the number of samples to read.

  If you set this input to nidaqmx.constants. READ_ALL_AVAILABLE, NI-DAQmx determines how many samples to read based on if the task acquires samples continuously or acquires a finite number of samples.

If the task acquires samples continuously and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, this method reads all the samples currently available in the buffer.

If the task acquires a finite number of samples and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, the method waits for the task to acquire all requested samples, then reads those samples. If you set the "read_all_avail_samp" property to True, the method reads the samples currently available in the buffer and does not wait for the task to acquire all requested samples.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

Returns Indicates the number of samples acquired by each channel. NI-DAQmx returns a single value because this value is the same for all channels.

Return type int

read_many_sample_port_uint16(*data*, *number_of_samples_per_channel=-1*, *timeout=10.0*)

Reads one or more 16-bit unsigned integer samples from one or more digital input channels in a task.

Use this method for devices with up to 16 lines per port.

This read method accepts a preallocated NumPy array to hold the samples requested, which can be advantageous for performance and interoperability with NumPy and SciPy.

Passing in a preallocated array is valuable in continuous acquisition scenarios, where the same array can be used repeatedly in each call to the method.

Parameters

- **data** (*numpy.ndarray*) – Specifies a preallocated 2D NumPy array of 16-bit unsigned integer values to hold the samples requested. The size of the array must be large enough to hold all requested samples from all channels in the task; otherwise, an error is thrown.

  Each row corresponds to a channel in the task. Each column corresponds to a sample from each channel. The order of the channels in the array corresponds to the order in which you add the channels to the task or to the order of the channels you specify with the "channels_to_read" property.

  If the size of the array is too large or the array is shaped incorrectly, the previous statement may not hold true as the samples read may not be separated into rows and columns properly. Set the "verify_array_shape" property on this channel reader object to True to validate that the NumPy array object is shaped properly. Setting this property may marginally adversely impact the performance of the method.

- **number_of_samples_per_channel** (*Optional[int]*) – Specifies the number of samples to read.

  If you set this input to nidaqmx.constants. READ_ALL_AVAILABLE, NI-DAQmx determines how many samples to read based on if the task acquires samples continuously or acquires a finite number of samples.

  If the task acquires samples continuously and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, this method reads all the samples currently available in the buffer.

If the task acquires a finite number of samples and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, the method waits for the task to acquire all requested samples, then reads those samples. If you set the "read_all_avail_samp" property to True, the method reads the samples currently available in the buffer and does not wait for the task to acquire all requested samples.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

Returns Indicates the number of samples acquired by each channel. NI-DAQmx returns a single value because this value is the same for all channels.

Return type int

read_many_sample_port_uint32 (*data*, *number_of_samples_per_channel=-1*, *timeout=10.0*)

Reads one or more 32-bit unsigned integer samples from one or more digital input channels in a task.

Use this method for devices with up to 32 lines per port.

This read method accepts a preallocated NumPy array to hold the samples requested, which can be advantageous for performance and interoperability with NumPy and SciPy.

Passing in a preallocated array is valuable in continuous acquisition scenarios, where the same array can be used repeatedly in each call to the method.

Parameters

- **data** (*numpy.ndarray*) – Specifies a preallocated 2D NumPy array of 32-bit unsigned integer values to hold the samples requested. The size of the array must be large enough to hold all requested samples from all channels in the task; otherwise, an error is thrown.

  Each row corresponds to a channel in the task. Each column corresponds to a sample from each channel. The order of the channels in the array corresponds to the order in which you add the channels to the task or to the order of the channels you specify with the "channels_to_read" property.

  If the size of the array is too large or the array is shaped incorrectly, the previous statement may not hold true as the samples read may not be separated into rows and columns properly. Set the "verify_array_shape" property on this channel reader object to True to validate that the NumPy array object is shaped properly. Setting this property may marginally adversely impact the performance of the method.

- **number_of_samples_per_channel** (*Optional[int]*) – Specifies the number of samples to read.

  If you set this input to nidaqmx.constants. READ_ALL_AVAILABLE, NI-DAQmx determines how many samples to read based on if the task acquires samples continuously or acquires a finite number of samples.

  If the task acquires samples continuously and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, this method reads all the samples currently available in the buffer.

  If the task acquires a finite number of samples and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, the method waits for the task to acquire all requested samples, then reads those samples. If you set the "read_all_avail_samp" property to True, the method reads the samples currently available in the buffer and does not wait for the task to acquire all requested samples.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

Returns Indicates the number of samples acquired by each channel. NI-DAQmx returns a single value because this value is the same for all channels.

Return type int

read_one_sample_multi_line(*data*, *timeout=10*)

Reads a single boolean sample from one or more digital input channels in a task. The channels can contain multiple digital lines.

This read method accepts a preallocated NumPy array to hold the samples requested, which can be advantageous for performance and interoperability with NumPy and SciPy.

Passing in a preallocated array is valuable in continuous acquisition scenarios, where the same array can be used repeatedly in each call to the method.

Parameters

- **data** (*numpy.ndarray*) – Specifies a preallocated 2D NumPy array of boolean values to hold the samples requested. The size of the array must be large enough to hold all requested samples from all channels in the task; otherwise, an error is thrown.

  Each row corresponds to a channel in the task. Each column corresponds to a line from each channel. The order of the channels in the array corresponds to the order in which you add the channels to the task or to the order of the channels you specify with the "channels_to_read" property.

  If the size of the array is too large or the array is shaped incorrectly, the previous statement may not hold true as the samples read may not be separated into rows and columns properly. Set the "verify_array_shape" property on this channel reader object to True to validate that the NumPy array object is shaped properly. Setting this property may marginally adversely impact the performance of the method.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

read_one_sample_one_line(*data*, *timeout=10*)

Reads a single boolean sample from one or more digital input channels in a task. The channel can contain only one digital line.

This read method accepts a preallocated NumPy array to hold the samples requested, which can be advantageous for performance and interoperability with NumPy and SciPy.

Passing in a preallocated array is valuable in continuous acquisition scenarios, where the same array can be used repeatedly in each call to the method.

Parameters

- **data** (*numpy.ndarray*) – Specifies a preallocated 1D NumPy array of boolean values to hold the samples requested.

  Each element in the array corresponds to a sample from each channel. The size of the array must be large enough to hold all requested samples from the channel in the task; otherwise, an error is thrown.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

read_one_sample_port_byte(*data*, *timeout=10*)

Reads a single 8-bit unsigned integer sample from one or more digital input channels in a task.

Use this method for devices with up to 8 lines per port.

This read method accepts a preallocated NumPy array to hold the samples requested, which can be advantageous for performance and interoperability with NumPy and SciPy.

Passing in a preallocated array is valuable in continuous acquisition scenarios, where the same array can be used repeatedly in each call to the method.

Parameters

- **data** (*numpy.ndarray*) – Specifies a preallocated 1D NumPy array of 8-bit unsigned integer values to hold the samples requested.

  Each element in the array corresponds to a sample from each channel. The size of the array must be large enough to hold all requested samples from the channel in the task; otherwise, an error is thrown.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

read_one_sample_port_uint16(*data*, *timeout=10*)

Reads a single 16-bit unsigned integer sample from one or more digital input channels in a task.

Use this method for devices with up to 16 lines per port.

This read method accepts a preallocated NumPy array to hold the samples requested, which can be advantageous for performance and interoperability with NumPy and SciPy.

Passing in a preallocated array is valuable in continuous acquisition scenarios, where the same array can be used repeatedly in each call to the method.

Parameters

- **data** (*numpy.ndarray*) – Specifies a preallocated 1D NumPy array of 16-bit unsigned integer values to hold the samples requested.

  Each element in the array corresponds to a sample from each channel. The size of the array must be large enough to hold all requested samples from the channel in the task; otherwise, an error is thrown.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

read_one_sample_port_uint32(*data*, *timeout=10*)

Reads a single 32-bit unsigned integer sample from one or more digital input channels in a task.

Use this method for devices with up to 32 lines per port.

This read method accepts a preallocated NumPy array to hold the samples requested, which can be advantageous for performance and interoperability with NumPy and SciPy.

Passing in a preallocated array is valuable in continuous acquisition scenarios, where the same array can be used repeatedly in each call to the method.

Parameters

- **data** (*numpy.ndarray*) – Specifies a preallocated 1D NumPy array of 32-bit unsigned integer values to hold the samples requested.

  Each element in the array corresponds to a sample from each channel. The size of the array must be large enough to hold all requested samples from the channel in the task; otherwise, an error is thrown.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

verify_array_shape

bool – Indicates whether the size and shape of the user-defined NumPy arrays passed to read methods are verified. Defaults to True when this object is instantiated.

Setting this property to True may marginally adversely impact the performance of read methods.

nidaqmx.stream_writers

class nidaqmx.stream_writers.**AnalogSingleChannelWriter**(*task_out_stream*, *auto_start=<nidaqmx.stream_writers.UnsetAutoStartSe object>*)

Bases: nidaqmx.stream_writers.ChannelWriterBase

Writes samples to an analog output channel in an NI-DAQmx task.

auto_start

bool – Specifies if the write method automatically starts the task if you did not explicitly start it with the DAQmx Start Task method.

If you do not specify a value for this parameter, NI-DAQmx determines its value based on the type of write method used. If you use a one sample write method, its value is True; conversely, if you use a many sample write method, its value is False.

verify_array_shape

bool – Indicates whether the size and shape of the user-defined NumPy arrays passed to read methods are verified. Defaults to True when this object is instantiated.

Setting this property to True may marginally adversely impact the performance of read methods.

write_many_sample(*data*, *timeout=10.0*)

Writes one or more floating-point samples to a single analog output channel in a task.

If the task uses on-demand timing, this method returns only after the device generates all samples. On-demand is the default timing type if you do not use the timing property on the task to configure a sample timing type. If the task uses any timing type other than on-demand, this method returns immediately and does not wait for the device to generate all samples. Your application must determine if the task is done to ensure that the device generated all samples.

Parameters

- **data** (*numpy.ndarray*) – Contains a 1D NumPy array of floating-point samples to write to the task. Each element of the array corresponds to a sample to write.
- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for the method to write all samples. NI-DAQmx performs a timeout check only if the method must wait before it writes data. This method returns an error if the time elapses. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to write the submitted samples. If the method could not write all the submitted samples, it returns an error and the number of samples successfully written.

Returns Specifies the actual number of samples this method successfully wrote.

Return type int

write_one_sample(*data*, *timeout=10*)

Writes a single floating-point sample to a single analog output channel in a task.

Parameters

- **data** (*float*) – Specifies the floating-point sample to write to the task.
- **auto_start** (*Optional[bool]*) – Specifies if this method automatically starts the task if you did not explicitly start it with the DAQmx Start Task method.
- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for the method to write all samples. NI-DAQmx performs a timeout check only if the method must wait before it writes data. This method returns an error if the time elapses. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to write the submitted samples. If the method could not write all the submitted samples, it returns an error and the number of samples successfully written.

class nidaqmx.stream_writers.**AnalogMultiChannelWriter**(*task_out_stream*, *auto_start=<nidaqmx.stream_writers.UnsetAutoStartSent object>*)

Bases: nidaqmx.stream_writers.ChannelWriterBase

Writes samples to one or more analog output channels in an NI-DAQmx task.

auto_start

bool – Specifies if the write method automatically starts the task if you did not explicitly start it with the DAQmx Start Task method.

If you do not specify a value for this parameter, NI-DAQmx determines its value based on the type of write method used. If you use a one sample write method, its value is True; conversely, if you use a many sample write method, its value is False.

verify_array_shape

bool – Indicates whether the size and shape of the user-defined NumPy arrays passed to read methods are verified. Defaults to True when this object is instantiated.

Setting this property to True may marginally adversely impact the performance of read methods.

write_many_sample(*data*, *timeout=10.0*)

Writes one or more floating-point samples to one or more analog output channels in a task.

If the task uses on-demand timing, this method returns only after the device generates all samples. On-demand is the default timing type if you do not use the timing property on the task to configure a sample timing type. If the task uses any timing type other than on-demand, this method returns immediately and does not wait for the device to generate all samples. Your application must determine if the task is done to ensure that the device generated all samples.

Parameters

- **data** (*numpy.ndarray*) – Contains a 2D NumPy array of floating-point samples to write to the task.

  Each row corresponds to a channel in the task. Each column corresponds to a sample to write to each channel. The order of the channels in the array corresponds to the order in which you add the channels to the task.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for the method to write all samples. NI-DAQmx performs a timeout check only if the method must wait before it writes data. This method returns an error if the time elapses. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to write the submitted samples. If the method could not write all the submitted samples, it returns an error and the number of samples successfully written.

Returns Specifies the actual number of samples this method successfully wrote to each channel in the task.

Return type int

write_one_sample(*data*, *timeout=10*)

Writes a single floating-point sample to one or more analog output channels in a task.

Parameters

- **data** (*numpy.ndarray*) – Contains a 1D NumPy array of floating-point samples to write to the task.

  Each element of the array corresponds to a channel in the task. The order of the channels in the array corresponds to the order in which you add the channels to the task.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for the method to write all samples. NI-DAQmx performs a timeout check only if the method must wait before it writes data. This method returns an error if the time elapses. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to write the submitted samples. If the method could not write all the submitted samples, it returns an error and the number of samples successfully written.

class nidaqmx.stream_writers.**AnalogUnscaledWriter**(*task_out_stream*, *auto_start=<nidaqmx.stream_writers.UnsetAutoStartSentinel object>*)

Bases: nidaqmx.stream_writers.ChannelWriterBase

Writes unscaled samples to one or more analog output channels in an NI-DAQmx task.

auto_start

bool – Specifies if the write method automatically starts the task if you did not explicitly start it with the DAQmx Start Task method.

If you do not specify a value for this parameter, NI-DAQmx determines its value based on the type of write method used. If you use a one sample write method, its value is True; conversely, if you use a many sample write method, its value is False.

verify_array_shape

bool – Indicates whether the size and shape of the user-defined NumPy arrays passed to read methods are verified. Defaults to True when this object is instantiated.

Setting this property to True may marginally adversely impact the performance of read methods.

write_int16(*data*, *timeout=10.0*)

Writes one or more unscaled 16-bit integer samples to one or more analog output channels in a task.

If the task uses on-demand timing, this method returns only after the device generates all samples. On-demand is the default timing type if you do not use the timing property on the task to configure a sample timing type. If the task uses any timing type other than on-demand, this method returns immediately and does not wait for the device to generate all samples. Your application must determine if the task is done to ensure that the device generated all samples.

Parameters

- **data** (*numpy.ndarray*) – Contains a 2D NumPy array of unscaled 16-bit integer samples to write to the task.

  Each row corresponds to a channel in the task. Each column corresponds to a sample to write to each channel.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for the method to write all samples. NI-DAQmx performs a timeout check only if the method must wait before it writes data. This method returns an error if the time elapses. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to write the submitted samples. If the method could not write all the submitted samples, it returns an error and the number of samples successfully written.

Returns Specifies the actual number of samples this method successfully wrote to each channel in the task.

Return type int

write_int32(*data*, *timeout=10.0*)

Writes one or more unscaled 32-bit integer samples to one or more analog output channels in a task.

If the task uses on-demand timing, this method returns only after the device generates all samples. On-demand is the default timing type if you do not use the timing property on the task to configure a sample timing type. If the task uses any timing type other than on-demand, this method returns immediately and does not wait for the device to generate all samples. Your application must determine if the task is done to ensure that the device generated all samples.

Parameters

- **data** (*numpy.ndarray*) – Contains a 2D NumPy array of unscaled 32-bit integer samples to write to the task.

  Each row corresponds to a channel in the task. Each column corresponds to a sample to write to each channel.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for the method to write all samples. NI-DAQmx performs a timeout check only if the method must wait before it writes data. This method returns an error if the time elapses. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to write the

submitted samples. If the method could not write all the submitted samples, it returns an error and the number of samples successfully written.

Returns Specifies the actual number of samples this method successfully wrote to each channel in the task.

Return type int

write_uint16(*data*, *timeout=10.0*)

Writes one or more unscaled 16-bit unsigned integer samples to one or more analog output channels in a task.

If the task uses on-demand timing, this method returns only after the device generates all samples. On-demand is the default timing type if you do not use the timing property on the task to configure a sample timing type. If the task uses any timing type other than on-demand, this method returns immediately and does not wait for the device to generate all samples. Your application must determine if the task is done to ensure that the device generated all samples.

Parameters

- **data** (*numpy.ndarray*) – Contains a 2D NumPy array of unscaled 16-bit unsigned integer samples to write to the task.

  Each row corresponds to a channel in the task. Each column corresponds to a sample to write to each channel.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for the method to write all samples. NI-DAQmx performs a timeout check only if the method must wait before it writes data. This method returns an error if the time elapses. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to write the submitted samples. If the method could not write all the submitted samples, it returns an error and the number of samples successfully written.

Returns Specifies the actual number of samples this method successfully wrote to each channel in the task.

Return type int

write_uint32(*data*, *timeout=10.0*)

Writes one or more unscaled 32-bit unsigned integer samples to one or more analog output channels in a task.

If the task uses on-demand timing, this method returns only after the device generates all samples. On-demand is the default timing type if you do not use the timing property on the task to configure a sample timing type. If the task uses any timing type other than on-demand, this method returns immediately and does not wait for the device to generate all samples. Your application must determine if the task is done to ensure that the device generated all samples.

Parameters

- **data** (*numpy.ndarray*) – Contains a 2D NumPy array of unscaled 32-bit unsigned integer samples to write to the task.

  Each row corresponds to a channel in the task. Each column corresponds to a sample to write to each channel.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for the method to write all samples. NI-DAQmx performs a timeout check only if the method must wait before it writes data. This method returns an error if the time elapses. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to write the

submitted samples. If the method could not write all the submitted samples, it returns an error and the number of samples successfully written.

Returns Specifies the actual number of samples this method successfully wrote to each channel in the task.

Return type int

class nidaqmx.stream_writers.**CounterWriter**(*task_out_stream*, *auto_start=<nidaqmx.stream_writers.UnsetAutoStartSentinel object>*)

Bases: nidaqmx.stream_writers.ChannelWriterBase

Writes samples to a counter output channel in an NI-DAQmx task.

auto_start

bool – Specifies if the write method automatically starts the task if you did not explicitly start it with the DAQmx Start Task method.

If you do not specify a value for this parameter, NI-DAQmx determines its value based on the type of write method used. If you use a one sample write method, its value is True; conversely, if you use a many sample write method, its value is False.

verify_array_shape

bool – Indicates whether the size and shape of the user-defined NumPy arrays passed to read methods are verified. Defaults to True when this object is instantiated.

Setting this property to True may marginally adversely impact the performance of read methods.

write_many_sample_pulse_frequency(*frequencies*, *duty_cycles*, *timeout=10.0*)

Writes one or more pulse samples in terms of frequency to a single counter output channel in a task.

If the task uses on-demand timing, this method returns only after the device generates all samples. On-demand is the default timing type if you do not use the timing property on the task to configure a sample timing type. If the task uses any timing type other than on-demand, this method returns immediately and does not wait for the device to generate all samples. Your application must determine if the task is done to ensure that the device generated all samples.

Parameters

- **frequencies** (*numpy.ndarray*) – Contains a 1D NumPy array of floating-point values that holds the frequency portion of the pulse samples to write to the task. Each element of the array corresponds to a sample to write.
- **duty_cycles** (*numpy.ndarray*) – Contains a 1D NumPy array of floating-point values that holds the duty cycle portion of the pulse samples to write to the task. Each element of the array corresponds to a sample to write.
- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for the method to write all samples. NI-DAQmx performs a timeout check only if the method must wait before it writes data. This method returns an error if the time elapses. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to write the submitted samples. If the method could not write all the submitted samples, it returns an error and the number of samples successfully written.

Returns Specifies the actual number of samples this method successfully wrote.

Return type int

write_many_sample_pulse_ticks(*high_ticks*, *low_ticks*, *timeout=10.0*)

Writes one or more pulse samples in terms of ticks to a single counter output channel in a task.

If the task uses on-demand timing, this method returns only after the device generates all samples. On-demand is the default timing type if you do not use the timing property on the task to configure a sample timing type. If the task uses any timing type other than on-demand, this method returns immediately and does not wait for the device to generate all samples. Your application must determine if the task is done to ensure that the device generated all samples.

Parameters

- **high_ticks** (*numpy.ndarray*) – Contains a 1D NumPy array of 32-bit unsigned integer values that holds the high ticks portion of the pulse samples to write to the task. Each element of the array corresponds to a sample to write.
- **low_ticks** (*numpy.ndarray*) – Contains a 1D NumPy array of 32-bit unsigned integer values that holds the low ticks portion of the pulse samples to write to the task. Each element of the array corresponds to a sample to write.
- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for the method to write all samples. NI-DAQmx performs a timeout check only if the method must wait before it writes data. This method returns an error if the time elapses. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to write the submitted samples. If the method could not write all the submitted samples, it returns an error and the number of samples successfully written.

Returns Specifies the actual number of samples this method successfully wrote.

Return type int

write_many_sample_pulse_time(*high_times*, *low_times*, *timeout=10.0*)

Writes one or more pulse samples in terms of time to a single counter output channel in a task.

If the task uses on-demand timing, this method returns only after the device generates all samples. On-demand is the default timing type if you do not use the timing property on the task to configure a sample timing type. If the task uses any timing type other than on-demand, this method returns immediately and does not wait for the device to generate all samples. Your application must determine if the task is done to ensure that the device generated all samples.

Parameters

- **high_times** (*numpy.ndarray*) – Contains a 1D NumPy array of floating-point values that holds the high time portion of the pulse samples to write to the task. Each element of the array corresponds to a sample to write.
- **low_times** (*numpy.ndarray*) – Contains a 1D NumPy array of floating-point values that holds the low time portion of the pulse samples to write to the task. Each element of the array corresponds to a sample to write.
- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for the method to write all samples. NI-DAQmx performs a timeout check only if the method must wait before it writes data. This method returns an error if the time elapses. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to write the submitted samples. If the method could not write all the submitted samples, it returns an error and the number of samples successfully written.

Returns Specifies the actual number of samples this method successfully wrote.

Return type int

write_one_sample_pulse_frequency(*frequency*, *duty_cycle*, *timeout=10*)

Writes a new pulse frequency and duty cycle to a single counter output channel in a task.

Parameters

- **frequency** (*float*) – Specifies at what frequency to generate pulses.
- **duty_cycle** (*float*) – Specifies the width of the pulse divided by the pulse period. NI-DAQmx uses this ratio combined with frequency to determine pulse width and the interval between pulses.
- **auto_start** (*Optional[bool]*) – Specifies if this method automatically starts the task if you did not explicitly start it with the DAQmx Start Task method.
- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for the method to write all samples. NI-DAQmx performs a timeout check only if the method must wait before it writes data. This method returns an error if the time elapses. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to write the submitted samples. If the method could not write all the submitted samples, it returns an error and the number of samples successfully written.

write_one_sample_pulse_ticks(*high_ticks*, *low_ticks*, *timeout=10*)

Writes a new pulse high tick count and low tick count to a single counter output channel in a task.

Parameters

- **high_ticks** (*float*) – Specifies the number of ticks the pulse is high.
- **low_ticks** (*float*) – Specifies the number of ticks the pulse is low.
- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for the method to write all samples. NI-DAQmx performs a timeout check only if the method must wait before it writes data. This method returns an error if the time elapses. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to write the submitted samples. If the method could not write all the submitted samples, it returns an error and the number of samples successfully written.

write_one_sample_pulse_time(*high_time*, *low_time*, *timeout=10*)

Writes a new pulse high time and low time to a single counter output channel in a task.

Parameters

- **high_time** (*float*) – Specifies the amount of time the pulse is high.
- **low_time** (*float*) – Specifies the amount of time the pulse is low.
- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for the method to write all samples. NI-DAQmx performs a timeout check only if the method must wait before it writes data. This method returns an error if the time elapses. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to write the submitted samples. If the method could not write all the submitted samples, it returns an error and the number of samples successfully written.

class nidaqmx.stream_writers.**DigitalSingleChannelWriter**(*task_out_stream*, *auto_start=<nidaqmx.stream_writers.UnsetAutoStartS object>*)

Bases: nidaqmx.stream_writers.ChannelWriterBase

Writes samples to a single digital output channel in an NI-DAQmx task.

auto_start

bool – Specifies if the write method automatically starts the task if you did not explicitly start it with the DAQmx Start Task method.

If you do not specify a value for this parameter, NI-DAQmx determines its value based on the type of write method used. If you use a one sample write method, its value is True; conversely, if you use a many sample write method, its value is False.

verify_array_shape

bool – Indicates whether the size and shape of the user-defined NumPy arrays passed to read methods are verified. Defaults to True when this object is instantiated.

Setting this property to True may marginally adversely impact the performance of read methods.

write_many_sample_port_byte(*data*, *timeout=10.0*)

Writes one or more 8-bit unsigned integer samples to a single digital output channel in a task.

Use this method for devices with up to 8 lines per port.

If the task uses on-demand timing, this method returns only after the device generates all samples. On-demand is the default timing type if you do not use the timing property on the task to configure a sample timing type. If the task uses any timing type other than on-demand, this method returns immediately and does not wait for the device to generate all samples. Your application must determine if the task is done to ensure that the device generated all samples.

Parameters

- **data** (*numpy.ndarray*) – Contains a 1D NumPy array of 8-bit unsigned integer samples to write to the task. Each element of the array corresponds to a sample to write.
- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for the method to write all samples. NI-DAQmx performs a timeout check only if the method must wait before it writes data. This method returns an error if the time elapses. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to write the submitted samples. If the method could not write all the submitted samples, it returns an error and the number of samples successfully written.

Returns Specifies the actual number of samples this method successfully wrote.

Return type int

write_many_sample_port_uint16(*data*, *timeout=10.0*)

Writes one or more 16-bit unsigned integer samples to a single digital output channel in a task.

Use this method for devices with up to 16 lines per port.

If the task uses on-demand timing, this method returns only after the device generates all samples. On-demand is the default timing type if you do not use the timing property on the task to configure a sample timing type. If the task uses any timing type other than on-demand, this method returns immediately and does not wait for the device to generate all samples. Your application must determine if the task is done to ensure that the device generated all samples.

Parameters

- **data** (*numpy.ndarray*) – Contains a 1D NumPy array of 16-bit unsigned integer samples to write to the task. Each element of the array corresponds to a sample to write.
- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for the method to write all samples. NI-DAQmx performs a timeout check only if the method must wait before it writes data. This method returns an error if the time elapses. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to write the submitted samples. If the method could not write all the submitted samples, it returns an error and the number of samples successfully written.

Returns Specifies the actual number of samples this method successfully wrote.

Return type int

write_many_sample_port_uint32(*data*, *timeout=10.0*)
Writes one or more 32-bit unsigned integer samples to a single digital output channel in a task.

Use this method for devices with up to 32 lines per port.

If the task uses on-demand timing, this method returns only after the device generates all samples. On-demand is the default timing type if you do not use the timing property on the task to configure a sample timing type. If the task uses any timing type other than on-demand, this method returns immediately and does not wait for the device to generate all samples. Your application must determine if the task is done to ensure that the device generated all samples.

Parameters

- **data** (*numpy.ndarray*) – Contains a 1D NumPy array of 32-bit unsigned integer samples to write to the task. Each element of the array corresponds to a sample to write.
- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for the method to write all samples. NI-DAQmx performs a timeout check only if the method must wait before it writes data. This method returns an error if the time elapses. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to write the submitted samples. If the method could not write all the submitted samples, it returns an error and the number of samples successfully written.

Returns Specifies the actual number of samples this method successfully wrote.

Return type int

write_one_sample_multi_line(*data*, *timeout=10*)
Writes a single boolean sample to a single digital output channel in a task. The channel can contain multiple digital lines.

Parameters

- **data** (*numpy.ndarray*) – Contains a 1D NumPy array of boolean samples to write to the task. Each element of the array corresponds to a line in the channel.
- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for the method to write all samples. NI-DAQmx performs a timeout check only if the method must wait before it writes data. This method returns an error if the time elapses. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to write the submitted samples. If the method could not write all the submitted samples, it returns an error and the number of samples successfully written.

write_one_sample_one_line(*data*, *timeout=10*)
Writes a single boolean sample to a single digital output channel in a task. The channel can contain only one digital line.

Parameters

- **data** (*int*) – Specifies the boolean sample to write to the task.
- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for the method to write all samples. NI-DAQmx performs a timeout check only if the method must wait before it writes data. This method returns an error if the time elapses. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to write the

submitted samples. If the method could not write all the submitted samples, it returns an error and the number of samples successfully written.

write_one_sample_port_byte(*data*, *timeout=10*)

Writes a single 8-bit unsigned integer sample to a single digital output channel in a task.

Use this method for devices with up to 8 lines per port.

Parameters

- **data** (*int*) – Specifies the 8-bit unsigned integer sample to write to the task.
- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for the method to write all samples. NI-DAQmx performs a timeout check only if the method must wait before it writes data. This method returns an error if the time elapses. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to write the submitted samples. If the method could not write all the submitted samples, it returns an error and the number of samples successfully written.

write_one_sample_port_uint16(*data*, *timeout=10*)

Writes a single 16-bit unsigned integer sample to a single digital output channel in a task.

Use this method for devices with up to 16 lines per port.

Parameters

- **data** (*int*) – Specifies the 16-bit unsigned integer sample to write to the task.
- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for the method to write all samples. NI-DAQmx performs a timeout check only if the method must wait before it writes data. This method returns an error if the time elapses. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to write the submitted samples. If the method could not write all the submitted samples, it returns an error and the number of samples successfully written.

write_one_sample_port_uint32(*data*, *timeout=10*)

Writes a single 32-bit unsigned integer sample to a single digital output channel in a task.

Use this method for devices with up to 32 lines per port.

Parameters

- **data** (*int*) – Specifies the 32-bit unsigned integer sample to write to the task.
- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for the method to write all samples. NI-DAQmx performs a timeout check only if the method must wait before it writes data. This method returns an error if the time elapses. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to write the submitted samples. If the method could not write all the submitted samples, it returns an error and the number of samples successfully written.

class nidaqmx.stream_writers.**DigitalMultiChannelWriter**(*task_out_stream*, *auto_start=<nidaqmx.stream_writers.UnsetAutoStartSe object>*)

Bases: nidaqmx.stream_writers.ChannelWriterBase

Writes samples to one or more digital output channels in an NI-DAQmx task.

`auto_start`

bool – Specifies if the write method automatically starts the task if you did not explicitly start it with the DAQmx Start Task method.

If you do not specify a value for this parameter, NI-DAQmx determines its value based on the type of write method used. If you use a one sample write method, its value is True; conversely, if you use a many sample write method, its value is False.

`verify_array_shape`

bool – Indicates whether the size and shape of the user-defined NumPy arrays passed to read methods are verified. Defaults to True when this object is instantiated.

Setting this property to True may marginally adversely impact the performance of read methods.

`write_many_sample_port_byte`(*data*, *timeout=10.0*)

Writes one or more 8-bit unsigned integer samples to one or more digital output channels in a task.

Use this method for devices with up to 8 lines per port.

If the task uses on-demand timing, this method returns only after the device generates all samples. On-demand is the default timing type if you do not use the timing property on the task to configure a sample timing type. If the task uses any timing type other than on-demand, this method returns immediately and does not wait for the device to generate all samples. Your application must determine if the task is done to ensure that the device generated all samples.

Parameters

- **`data`** (*`numpy.ndarray`*) – Contains a 2D NumPy array of 8-bit unsigned integer samples to write to the task.

  Each row corresponds to a channel in the task. Each column corresponds to a sample to write to each channel. The order of the channels in the array corresponds to the order in which you add the channels to the task.

- **`timeout`** (*`Optional[float]`*) – Specifies the amount of time in seconds to wait for the method to write all samples. NI-DAQmx performs a timeout check only if the method must wait before it writes data. This method returns an error if the time elapses. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to write the submitted samples. If the method could not write all the submitted samples, it returns an error and the number of samples successfully written.

Returns Specifies the actual number of samples this method successfully wrote to each channel in the task.

Return type int

`write_many_sample_port_uint16`(*data*, *timeout=10.0*)

Writes one or more 16-bit unsigned integer samples to one or more digital output channels in a task.

Use this method for devices with up to 16 lines per port.

If the task uses on-demand timing, this method returns only after the device generates all samples. On-demand is the default timing type if you do not use the timing property on the task to configure a sample timing type. If the task uses any timing type other than on-demand, this method returns immediately and does not wait for the device to generate all samples. Your application must determine if the task is done to ensure that the device generated all samples.

Parameters

- **`data`** (*`numpy.ndarray`*) – Contains a 2D NumPy array of 16-bit unsigned integer samples to write to the task.

Each row corresponds to a channel in the task. Each column corresponds to a sample to write to each channel. The order of the channels in the array corresponds to the order in which you add the channels to the task.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for the method to write all samples. NI-DAQmx performs a timeout check only if the method must wait before it writes data. This method returns an error if the time elapses. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to write the submitted samples. If the method could not write all the submitted samples, it returns an error and the number of samples successfully written.

Returns Specifies the actual number of samples this method successfully wrote to each channel in the task.

Return type int

write_many_sample_port_uint32(*data*, *timeout=10.0*)

Writes one or more 32-bit unsigned integer samples to one or more digital output channels in a task.

Use this method for devices with up to 32 lines per port.

If the task uses on-demand timing, this method returns only after the device generates all samples. On-demand is the default timing type if you do not use the timing property on the task to configure a sample timing type. If the task uses any timing type other than on-demand, this method returns immediately and does not wait for the device to generate all samples. Your application must determine if the task is done to ensure that the device generated all samples.

Parameters

- **data** (*numpy.ndarray*) – Contains a 2D NumPy array of 32-bit unsigned integer samples to write to the task.

  Each row corresponds to a channel in the task. Each column corresponds to a sample to write to each channel. The order of the channels in the array corresponds to the order in which you add the channels to the task.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for the method to write all samples. NI-DAQmx performs a timeout check only if the method must wait before it writes data. This method returns an error if the time elapses. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to write the submitted samples. If the method could not write all the submitted samples, it returns an error and the number of samples successfully written.

Returns Specifies the actual number of samples this method successfully wrote to each channel in the task.

Return type int

write_one_sample_multi_line(*data*, *timeout=10*)

Writes a single boolean sample to one or more digital output channels in a task. The channel can contain multiple digital lines.

Parameters

- **data** (*numpy.ndarray*) – Contains a 2D NumPy array of boolean samples to write to the task.

  Each row corresponds to a channel in the task. Each column corresponds to a line from each channel. The order of the channels in the array corresponds to the order in which you add the channels to the task.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for the method to write all samples. NI-DAQmx performs a timeout check only if the method must wait before it writes data. This method returns an error if the time elapses. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to write the submitted samples. If the method could not write all the submitted samples, it returns an error and the number of samples successfully written.

write_one_sample_one_line(*data*, *timeout=10*)

Writes a single boolean sample to one or more digital output channels in a task. The channel can contain only one digital line.

Parameters

- **data** (*numpy.ndarray*) – Contains a 1D NumPy array of boolean samples to write to the task.

  Each element in the array corresponds to a channel in the task. The order of the channels in the array corresponds to the order in which you add the channels to the task.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for the method to write all samples. NI-DAQmx performs a timeout check only if the method must wait before it writes data. This method returns an error if the time elapses. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to write the submitted samples. If the method could not write all the submitted samples, it returns an error and the number of samples successfully written.

write_one_sample_port_byte(*data*, *timeout=10*)

Writes a single 8-bit unsigned integer sample to one or more digital output channels in a task.

Use this method for devices with up to 8 lines per port.

Parameters

- **data** (*numpy.ndarray*) – Contains a 1D NumPy array of 8-bit unsigned integer samples to write to the task.

  Each element in the array corresponds to a channel in the task. The order of the channels in the array corresponds to the order in which you add the channels to the task.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for the method to write all samples. NI-DAQmx performs a timeout check only if the method must wait before it writes data. This method returns an error if the time elapses. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to write the submitted samples. If the method could not write all the submitted samples, it returns an error and the number of samples successfully written.

write_one_sample_port_uint16(*data*, *timeout=10*)

Writes a single 16-bit unsigned integer sample to one or more digital output channels in a task.

Use this method for devices with up to 16 lines per port.

Parameters

- **data** (*numpy.ndarray*) – Contains a 1D NumPy array of 16-bit unsigned integer samples to write to the task.

  Each element in the array corresponds to a channel in the task. The order of the channels in the array corresponds to the order in which you add the channels to the task.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for the method to write all samples. NI-DAQmx performs a timeout check only if the method must wait before it writes data. This method returns an error if the time elapses. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to write the submitted samples. If the method could not write all the submitted samples, it returns an error and the number of samples successfully written.

write_one_sample_port_uint32(*data*, *timeout=10*)

Writes a single 32-bit unsigned integer sample to one or more digital output channels in a task.

Use this method for devices with up to 32 lines per port.

Parameters

- **data** (*numpy.ndarray*) – Contains a 1D NumPy array of 32-bit unsigned integer samples to write to the task.

  Each element in the array corresponds to a channel in the task. The order of the channels in the array corresponds to the order in which you add the channels to the task.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for the method to write all samples. NI-DAQmx performs a timeout check only if the method must wait before it writes data. This method returns an error if the time elapses. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to write the submitted samples. If the method could not write all the submitted samples, it returns an error and the number of samples successfully written.

nidaqmx.system

class nidaqmx.system.system.**System**

Bases: object

Represents a DAQmx system.

Contains static properties that access tasks, scales, and global channels stored in Measurement Automation Explorer (MAX), performs immediate operations on DAQ hardware, and creates classes from which you can get information about the hardware.

add_cdaq_sync_connection(*ports_to_connect*)

Adds a cDAQ Sync connection between devices. The connection is not verified.

Parameters **ports_to_connect** (nidaqmx.types.CDAQSyncConnection) – Specifies the cDAQ Sync ports to connect.

are_configured_cdaq_sync_ports_disconnected(*chassis_devices_ports=u''*, *timeout=-1.0*)

Verifies configured cDAQ Sync connections between devices. Failures generally indicate a wiring issue or that a device has been powered off or removed. Stop all NI-DAQmx tasks running on the devices prior to running this function because any running tasks cause the verification process to fail.

Parameters

- **chassis_devices_ports** (*Optional[str]*) – Specifies the names of the CompactDAQ chassis, C Series modules, or cDAQ Sync ports in comma separated form to search. If no names are specified, all cDAQ Sync ports on connected, non-simulated devices are scanned.

- **timeout** (*Optional[float]*) – Specifies the time in seconds to wait for the device to respond before timing out.

Returns Returns the port-to-port connections that failed verification.

Return type List[*nidaqmx.types.CDAQSyncConnection*]

auto_configure_cdaq_sync_connections (*chassis_devices_ports=u''*, *timeout=-1.0*)

Detects and configures cDAQ Sync connections between devices. Stop all NI-DAQmx tasks running on the devices prior to running this function because any running tasks cause auto-configuration to fail.

Parameters

- **chassis_devices_ports** (*Optional[str]*) – Specifies the names of the CompactDAQ chassis, C Series modules, or cDAQ Sync ports in comma separated form to search. If no names are specified, all cDAQ Sync ports on connected, non-simulated devices are scanned.
- **timeout** (*Optional[float]*) – Specifies the time in seconds to wait for the device to respond before timing out. If a timeout occurs, no configuration is changed.

Returns Returns the configured port-to-port connections.

Return type List[*nidaqmx.types.CDAQSyncConnection*]

connect_terms (*source_terminal*, *destination_terminal*, *signal_modifiers=<SignalModifiers.DO_NOT_INVERT_POLARITY: 0>*)

Creates a route between a source and destination terminal. The route can carry a variety of digital signals, such as triggers, clocks, and hardware events.

Parameters

- **source_terminal** (*str*) – Specifies the originating terminal of the route. A DAQmx terminal constant lists all terminals available on devices installed in the system. You also can specify a source terminal by specifying a string that contains a terminal name.
- **destination_terminal** (*str*) – Specifies the receiving terminal of the route. A DAQmx terminal constant provides a list of all terminals available on devices installed in the system. You also can specify a destination terminal by specifying a string that contains a terminal name.
- **signal_modifiers** (*Optional[nidaqmx.constants.SignalModifiers]*) – Specifies whether to invert the signal this function routes from the source terminal to the destination terminal.

devices

nidaqmx.system._collections.DeviceCollection – Indicates the collection of devices for this DAQmx system.

disconnect_terms (*source_terminal*, *destination_terminal*)

Removes signal routes you created by using the DAQmx Connect Terminals function. The DAQmx Disconnect Terminals function cannot remove task-based routes, such as those you create through timing and triggering configuration.

Parameters

- **source_terminal** (*str*) – Specifies the originating terminal of the route. A DAQmx terminal constant lists all terminals available on devices installed in the system. You also can specify a source terminal by specifying a string that contains a terminal name.
- **destination_terminal** (*str*) – Specifies the receiving terminal of the route. A DAQmx terminal constant provides a list of all terminals available on devices installed in

the system. You also can specify a destination terminal by specifying a string that contains a terminal name.

driver_version

collections.namedtuple – Indicates the major, minor and update portions of the installed version of NI-DAQmx.

- major_version (int): Indicates the major portion of the installed version of NI-DAQmx, such as 7 for version 7.0.
- minor_version (int): Indicates the minor portion of the installed version of NI-DAQmx, such as 0 for version 7.0.
- update_version (int): Indicates the update portion of the installed version of NI-DAQmx, such as 1 for version 9.0.1.

get_analog_power_up_states(*device_name*)

Gets the power up states for analog physical channels.

Parameters **device_name** (`str`) – Specifies the name as configured in MAX of the device to which this operation applies.

Returns

Contains the physical channels and power up states set. Each element of the list contains a physical channel and the power up state set for that physical channel.

- physical_channel (str): Specifies the physical channel that was modified.
- power_up_state (float): Specifies the power up state set for the physical channel specified with the **physical_channel** input.
- channel_type (`nidaqmx.constants.AOPowerUpOutputBehavior`): Specifies the output type for the physical channel specified with the **physical_channel** input.

Return type power_up_states (List[*nidaqmx.types.AOPowerUpState*])

get_analog_power_up_states_with_output_type(*physical_channels*)

Gets the power up states for analog physical channels.

Parameters **physical_channels** (`List[str]`) – Indicates the physical channels that were modified.

Returns

Contains the physical channels and power up states set. Each element of the list contains a physical channel and the power up state set for that physical channel.

- physical_channel (str): Specifies the physical channel that was modified.
- power_up_state (float): Specifies the power up state set for the physical channel specified with the **physical_channel** input.
- channel_type (`nidaqmx.constants.AOPowerUpOutputBehavior`): Specifies the output type for the physical channel specified with the **physical_channel** input.

Return type power_up_states (List[*nidaqmx.types.AOPowerUpState*])

get_digital_logic_family_power_up_state(*device_name*)

Gets the digital logic family for a device.

Parameters **device_name** (`str`) – Specifies the name as configured in MAX of the device to which this operation applies.

Returns Specifies the logic family to set the device to when it powers up. A logic family corresponds to voltage thresholds that are compatible with a group of voltage standards. Refer to device documentation for information on the logic high and logic low voltages for these logic families.

Return type *nidaqmx.constants.LogicFamily*

get_digital_power_up_states(*device_name*)
Gets the power up states for digital physical lines.

Parameters **device_name** (*str*) – Specifies the name as configured in MAX of the device to which this operation applies.

Returns

Contains the physical channels and power up states set. Each element of the list contains a physical channel and the power up state set for that physical channel.

- physical_channel (str): Indicates the physical channel that was modified.
- power_up_state (*nidaqmx.constants.PowerUpStates*): Indicates the power up state set for the physical channel specified with the **physical_channel** output.

Return type List[*nidaqmx.types.DOPowerUpState*]

get_digital_pull_up_pull_down_states(*device_name*)
Gets the resistor level for lines when they are in tristate logic.

Parameters **device_name** (*str*) – Specifies the name as configured in MAX of the device to which this operation applies.

Returns

Contains the physical channels and power up states set. Each element of the list contains a physical channel and the power up state set for that physical channel.

- physical_channel (str): Indicates the physical channel that was modified.
- power_up_state (*nidaqmx.constants.ResistorState*): Indicates the power up state set for the physical channel specified with the **physical_channel** output.

Return type List[*nidaqmx.types.DOResistorPowerUpState*]

global_channels
nidaqmx.system._collections.PersistedChannelCollection – Indicates the collection of global channels for this DAQmx system.

static local()
nidaqmx.system.system.System: Represents the local DAQmx system.

remove_cdaq_sync_connection(*ports_to_disconnect*)
Removes a cDAQ Sync connection between devices. The connection is not verified.

Parameters **ports_to_disconnect** (nidaqmx.types.CDAQSyncConnection) – Specifies the cDAQ Sync ports to disconnect.

scales
nidaqmx.system._collections.PersistedScaleCollection – Indicates the collection of custom scales for this DAQmx system.

set_analog_power_up_states(*device_name*, *power_up_states*)
Updates power up states for analog physical channels.

Parameters

- **device_name** (*str*) – Specifies the name as configured in MAX of the device to which this operation applies.
- **power_up_states** (*List[nidaqmx.types.AOPowerUpState]*) – Contains the physical channels and power up states to set. Each element of the list contains a physical channel and the power up state to set for that physical channel.
  - physical_channel (str): Specifies the physical channel to modify.
  - power_up_state (float): Specifies the power up state to set for the physical channel specified with the **physical_channel** input.
  - channel_type (*nidaqmx.constants.AOPowerUpOutputBehavior*): Specifies the output type for the physical channel specified with the **physical_channel** input.

set_analog_power_up_states_with_output_type(*power_up_states*)
Updates power up states for analog physical channels.

Parameters **power_up_states** (*List[nidaqmx.types.AOPowerUpState]*) – Contains the physical channels and power up states to set. Each element of the list contains a physical channel and the power up state to set for that physical channel.

- physical_channel (str): Specifies the physical channel to modify.
- power_up_state (float): Specifies the power up state to set for the physical channel specified with the **physical_channel** input.
- channel_type (*nidaqmx.constants.AOPowerUpOutputBehavior*): Specifies the output type for the physical channel specified with the **physical_channel** input.

set_digital_logic_family_power_up_state(*device_name*, *logic_family*)
Sets the digital logic family to use when the device powers up.

Parameters

- **device_name** (*str*) – Specifies the name as configured in MAX of the device to which this operation applies.
- **logic_family** (*nidaqmx.constants.LogicFamily*) – Specifies the logic family set to the device to when it powers up. A logic family corresponds to voltage thresholds that are compatible with a group of voltage standards. Refer to device documentation for information on the logic high and logic low voltages for these logic families.

set_digital_power_up_states(*device_name*, *power_up_states*)
Updates power up states for digital physical channels.

Parameters

- **device_name** (*str*) – Specifies the name as configured in MAX of the device to which this operation applies.
- **power_up_states** (*List[nidaqmx.types.DOPowerUpState]*) – Contains the physical channels and power up states to set. Each element of the list contains a physical channel and the power up state to set for that physical channel.
  - physical_channel (str): Specifies the digital line or port to modify. You cannot modify dedicated digital input lines.
  - power_up_state (*nidaqmx.constants.PowerUpStates*): Specifies the power up state to set for the physical channel specified with the **physical_channel** input.

set_digital_pull_up_pull_down_states(*device_name*, *power_up_states*)
Sets the resistor level to pull up or pull down for lines when they are in tristate logic.

Parameters

- **device_name** (*str*) – Specifies the name as configured in MAX of the device to which this operation applies.
- **power_up_states** (*List[nidaqmx.types.DOResistorPowerUpState]*) – Contains the physical channels and power up states to set. Each element of the list contains a physical channel and the power up state to set for that physical channel.
  - physical_channel (str): Specifies the digital line or port to modify. You cannot modify dedicated digital input lines.
  - power_up_state (*nidaqmx.constants.ResistorState*): Specifies the power up state to set for the physical channel specified with the **physical_channel** input.

tasks

nidaqmx.system._collections.PersistedTaskCollection – Indicates the collection of saved tasks for this DAQmx system.

tristate_output_term(*output_terminal*)

Sets a terminal to high-impedance state. If you connect an external signal to a terminal on the I/O connector, the terminal must be in high-impedance state. Otherwise, the device could double-drive the terminal and damage the hardware. If you use this function on a terminal in an active route, the function fails and returns an error.

Parameters **output_terminal** (*str*) – Specifies the terminal on the I/O connector to set to high-impedance state. A DAQmx terminal constant lists all available terminals on installed devices. You also can specify an output terminal by using a string that contains a terminal name.

nidaqmx.system.collections

nidaqmx.system.device_collection

class nidaqmx.system._collections.device_collection.**DeviceCollection**

Bases: `_abcoll.Sequence`

Contains the collection of devices for a DAQmx system.

This class defines methods that implements a container object.

device_names

List[str] – Indicates the names of all devices on this device collection.

nidaqmx.system.persisted_channel_collection

class nidaqmx.system._collections.persisted_channel_collection.**PersistedChannelCollection**

Bases: `_abcoll.Sequence`

Contains the collection of global channels for a DAQmx system.

This class defines methods that implements a container object.

global_channel_names

List[str] – The names of all the global channels on this collection.

nidaqmx.system.persisted_scale_collection

class `nidaqmx.system._collections.persisted_scale_collection.`**`PersistedScaleCollection`**

Bases: `_abcoll.Sequence`

Contains the collection of custom scales on a DAQmx system.

This class defines methods that implements a container object.

`scale_names`

List[str] – Indicates the names of all the custom scales on this collection.

nidaqmx.system.persisted_task_collection

class `nidaqmx.system._collections.persisted_task_collection.`**`PersistedTaskCollection`**

Bases: `_abcoll.Sequence`

Contains the collection of task saved on a DAQmx system.

This class defines methods that implements a container object.

`task_names`

List[str] – Indicates the names of all the tasks on this collection.

nidaqmx.system.physical_channel_collection

class `nidaqmx.system._collections.physical_channel_collection.`**`AIPhysicalChannelCollection`**(*devi*

Bases: *`nidaqmx.system._collections.physical_channel_collection.PhysicalChannelCollection`*

Contains the collection of analog input physical channels for a DAQmx device.

This class defines methods that implements a container object.

class `nidaqmx.system._collections.physical_channel_collection.`**`AOPhysicalChannelCollection`**(*devi*

Bases: *`nidaqmx.system._collections.physical_channel_collection.PhysicalChannelCollection`*

Contains the collection of analog output physical channels for a DAQmx device.

This class defines methods that implements a container object.

class `nidaqmx.system._collections.physical_channel_collection.`**`CIPhysicalChannelCollection`**(*devi*

Bases: *`nidaqmx.system._collections.physical_channel_collection.PhysicalChannelCollection`*

Contains the collection of counter input physical channels for a DAQmx device.

This class defines methods that implements a container object.

class `nidaqmx.system._collections.physical_channel_collection.`**`COPhysicalChannelCollection`**(*devi*

Bases: *`nidaqmx.system._collections.physical_channel_collection.PhysicalChannelCollection`*

Contains the collection of counter output physical channels for a DAQmx device.

This class defines methods that implements a container object.

class `nidaqmx.system._collections.physical_channel_collection.`**`DILinesCollection`**(*device_name*)

Bases: *`nidaqmx.system._collections.physical_channel_collection.PhysicalChannelCollection`*

Contains the collection of digital input lines for a DAQmx device.

This class defines methods that implements a container object.

class nidaqmx.system._collections.physical_channel_collection.**DIPortsCollection**(*device_name*)

Bases: *nidaqmx.system._collections.physical_channel_collection.PhysicalChannelCollection*

Contains the collection of digital input ports for a DAQmx device.

This class defines methods that implements a container object.

class nidaqmx.system._collections.physical_channel_collection.**DOLinesCollection**(*device_name*)

Bases: *nidaqmx.system._collections.physical_channel_collection.PhysicalChannelCollection*

Contains the collection of digital output lines for a DAQmx device.

This class defines methods that implements a container object.

class nidaqmx.system._collections.physical_channel_collection.**DOPortsCollection**(*device_name*)

Bases: *nidaqmx.system._collections.physical_channel_collection.PhysicalChannelCollection*

Contains the collection of digital output ports for a DAQmx device.

This class defines methods that implements a container object.

class nidaqmx.system._collections.physical_channel_collection.**PhysicalChannelCollection**(*device_*

Bases: _abcoll.Sequence

Contains the collection of physical channels for a DAQmx device.

This class defines methods that implements a container object.

all

nidaqmx.system.physical_channel.PhysicalChannel – Specifies a physical channel object that represents the entire list of physical channels on this channel collection.

channel_names

List[str] – Specifies the entire list of physical channels on this collection.

nidaqmx.system.device

class nidaqmx.system.device.**Device**(*name*)

Bases: object

Represents a DAQmx device.

__init__(*name*)

Parameters **name** (*str*) – Specifies the name of the device.

__weakref__

list of weak references to the object (if defined)

accessory_product_nums

List[int] – Indicates the unique hardware identification number for accessories connected to the device. Each list element corresponds to a connector. For example, index 0 corresponds to connector 0. The list contains 0 for each connector with no accessory connected.

accessory_product_types

List[str] – Indicates the model names of accessories connected to the device. Each list element corresponds to a connector. For example, index 0 corresponds to connector 0. The list contains an empty string for each connector with no accessory connected.

accessory_serial_nums
List[int] – Indicates the serial number for accessories connected to the device. Each list element corresponds to a connector. For example, index 0 corresponds to connector 0. The list contains 0 for each connector with no accessory connected.

static add_network_device(*ip_address*, *device_name=u''*, *attempt_reservation=False*, *timeout=10.0*)
Adds a Network cDAQ device to the system and, if specified, attempts to reserve it.

Parameters

- **ip_address** (*str*) – Specifies the string containing the IP address (in dotted decimal notation) or hostname of the device to add to the system.
- **device_name** (*Optional[str]*) – Indicates the name to assign to the device. If unspecified, NI-DAQmx chooses the device name.
- **attempt_reservation** (*Optional[bool]*) – Indicates if a reservation should be attempted after the device is successfully added. By default, this parameter is set to False.
- **timeout** (*Optional[float]*) – Specifies the time in seconds to wait for the device to respond before timing out.

Returns Specifies the object that represents the device this operation applied to.

Return type *nidaqmx.system.device.Device*

ai_bridge_rngs
List[float] – Indicates pairs of input voltage ratio ranges, in volts per volt, supported by devices that acquire using ratiometric measurements. Each pair consists of the low value followed by the high value.

ai_charge_rngs
List[float] – Indicates in coulombs pairs of input charge ranges for the device. Each pair consists of the low value followed by the high value.

ai_couplings
List[*nidaqmx.constants.Coupling*] – Indicates the coupling types supported by this device.

ai_current_int_excit_discrete_vals
List[float] – Indicates the set of discrete internal current excitation values supported by this device.

ai_current_rngs
List[float] – Indicates the pairs of current input ranges supported by this device. Each pair consists of the low value, followed by the high value.

ai_dig_fltr_lowpass_cutoff_freq_discrete_vals
List[float] – Indicates the set of discrete lowpass cutoff frequencies supported by this device. If the device supports ranges of lowpass cutoff frequencies, use AI.DigFltr.Lowpass.CutoffFreq.RangeVals to determine supported frequencies.

ai_dig_fltr_lowpass_cutoff_freq_range_vals
List[float] – Indicates pairs of lowpass cutoff frequency ranges supported by this device. Each pair consists of the low value, followed by the high value. If the device supports a set of discrete lowpass cutoff frequencies, use AI.DigFltr.Lowpass.CutoffFreq.DiscreteVals to determine the supported frequencies.

ai_dig_fltr_types
List[*nidaqmx.constants.FilterType*] – Indicates the AI digital filter types supported by the device.

ai_freq_rngs
List[float] – Indicates the pairs of frequency input ranges supported by this device. Each pair consists of the low value, followed by the high value.

ai_gains
List[float] – Indicates the input gain settings supported by this device.

ai_lowpass_cutoff_freq_discrete_vals
List[float] – Indicates the set of discrete lowpass cutoff frequencies supported by this device. If the device supports ranges of lowpass cutoff frequencies, use **ai_lowpass_cutoff_freq_range_vals** to determine supported frequencies.

ai_lowpass_cutoff_freq_range_vals
List[float] – Indicates pairs of lowpass cutoff frequency ranges supported by this device. Each pair consists of the low value, followed by the high value. If the device supports a set of discrete lowpass cutoff frequencies, use **ai_lowpass_cutoff_freq_discrete_vals** to determine the supported frequencies.

ai_max_multi_chan_rate
float – Indicates the maximum sampling rate for an analog input task from this device. To find the maximum rate for the task, take the minimum of **ai_max_single_chan_rate** or the indicated sampling rate of this device divided by the number of channels to acquire data from (including cold-junction compensation and autozero channels).

ai_max_single_chan_rate
float – Indicates the maximum rate for an analog input task if the task contains only a single channel from this device.

ai_meas_types
List[`nidaqmx.constants.UsageTypeAI`] – Indicates the measurement types supported by the physical channels of the device. Refer to **ai_meas_types** for information on specific channels.

ai_min_rate
float – Indicates the minimum rate for an analog input task on this device. NI-DAQmx returns a warning or error if you attempt to sample at a slower rate.

ai_physical_chans
List[nidaqmx.system._collections.PhysicalChannelCollection] – Indicates a collection that contains all the analog input physical channels available on the device.

ai_resistance_rngs
List[float] – Indicates pairs of input resistance ranges, in ohms, supported by devices that have the necessary signal conditioning to measure resistances. Each pair consists of the low value followed by the high value.

ai_samp_modes
List[`nidaqmx.constants.AcquisitionType`] – Indicates sample modes supported by devices that support sample clocked analog input.

ai_simultaneous_sampling_supported
bool – Indicates if the device supports simultaneous sampling.

ai_trig_usage
List[`nidaqmx.constants.TriggerUsage`] – Indicates the triggers supported by this device for an analog input task.

ai_voltage_int_excit_discrete_vals
List[float] – Indicates the set of discrete internal voltage excitation values supported by this device. If the device supports ranges of internal excitation values, use **ai_voltage_int_excit_range_vals** to determine supported excitation values.

ai_voltage_int_excit_range_vals
List[float] – Indicates pairs of internal voltage excitation ranges supported by this device. Each pair consists of the low value, followed by the high value. If the device supports a set of discrete internal excitation values, use **ai_voltage_int_excit_discrete_vals** to determine the supported excitation values.

ai_voltage_rngs
List[float] – Indicates pairs of input voltage ranges supported by this device. Each pair consists of the low value, followed by the high value.

anlg_trig_supported
bool – Indicates if the device supports analog triggering.

ao_current_rngs
List[float] – Indicates pairs of output current ranges supported by this device. Each pair consists of the low value, followed by the high value.

ao_gains
List[float] – Indicates the output gain settings supported by this device.

ao_max_rate
float – Indicates the maximum analog output rate of the device.

ao_min_rate
float – Indicates the minimum analog output rate of the device.

ao_output_types
List[`nidaqmx.constants.UsageTypeAO`] – Indicates the generation types supported by the physical channels of the device. Refer to **ao_output_types** for information on specific channels.

ao_physical_chans
List[nidaqmx.system._collections.PhysicalChannelCollection] – Indicates a collection that contains all the analog output physical channels available on the device.

ao_samp_clk_supported
bool – Indicates if the device supports the sample clock timing type for analog output tasks.

ao_samp_modes
List[`nidaqmx.constants.AcquisitionType`] – Indicates sample modes supported by devices that support sample clocked analog output.

ao_trig_usage
List[`nidaqmx.constants.TriggerUsage`] – Indicates the triggers supported by this device for analog output tasks.

ao_voltage_rngs
List[float] – Indicates pairs of output voltage ranges supported by this device. Each pair consists of the low value, followed by the high value.

bus_type
`nidaqmx.constants.BusType` – Indicates the bus type of the device.

carrier_serial_num
int – Indicates the serial number of the device carrier. This value is zero if the carrier does not have a serial number.

chassis_module_devices
List[`nidaqmx.system.device.Device`] – Indicates a list containing the names of the modules in the chassis.

ci_max_size
int – Indicates in bits the size of the counters on the device.

ci_max_timebase
float – Indicates in hertz the maximum counter timebase frequency.

ci_meas_types
List[*nidaqmx.constants.UsageTypeCI*] – Indicates the measurement types supported by the physical channels of the device. Refer to **ci_meas_types** for information on specific channels.

ci_physical_chans
List[nidaqmx.system._collections.PhysicalChannelCollection] – Indicates a collection that contains all the counter input physical channels available on the device.

ci_samp_clk_supported
bool – Indicates if the device supports the sample clock timing type for counter input tasks.

ci_samp_modes
List[*nidaqmx.constants.AcquisitionType*] – Indicates sample modes supported by devices that support sample clocked counter input.

ci_trig_usage
List[*nidaqmx.constants.TriggerUsage*] – Indicates the triggers supported by this device for counter input tasks.

co_max_size
int – Indicates in bits the size of the counters on the device.

co_max_timebase
float – Indicates in hertz the maximum counter timebase frequency.

co_output_types
List[*nidaqmx.constants.UsageTypeCO*] – Indicates the generation types supported by the physical channels of the device. Refer to **co_output_types** for information on specific channels.

co_physical_chans
List[nidaqmx.system._collections.PhysicalChannelCollection] – Indicates a collection that contains all the counter output physical channels available on the device.

co_samp_clk_supported
bool – Indicates if the device supports Sample Clock timing for counter output tasks.

co_samp_modes
List[*nidaqmx.constants.AcquisitionType*] – Indicates sample modes supported by devices that support sample clocked counter output.

co_trig_usage
List[*nidaqmx.constants.TriggerUsage*] – Indicates the triggers supported by this device for counter output tasks.

compact_daq_chassis_device
nidaqmx.system.device.Device – Indicates the name of the CompactDAQ chassis that contains this module.

compact_daq_slot_num
int – Indicates the slot number in which this module is located in the CompactDAQ chassis.

delete_network_device()
Deletes a Network DAQ device previously added to the host. If the device is reserved, it is unreserved before it is removed.

dev_is_simulated
bool – Indicates if the device is a simulated device.

dev_serial_num
int – Indicates the serial number of the device. This value is zero if the device does not have a serial number.

di_lines
List[nidaqmx.system._collections.PhysicalChannelCollection] – Indicates a collection that contains all the digital input lines available on the device.

di_max_rate
float – Indicates the maximum digital input rate of the device.

di_ports
List[nidaqmx.system._collections.PhysicalChannelCollection] – Indicates a collection that contains all the digital input ports available on the device.

di_trig_usage
List[*nidaqmx.constants.TriggerUsage*] – Indicates the triggers supported by this device for digital input tasks.

dig_trig_supported
bool – Indicates if the device supports digital triggering.

do_lines
List[nidaqmx.system._collections.PhysicalChannelCollection] – Indicates a collection that contains all the digital output lines available on the device.

do_max_rate
float – Indicates the maximum digital output rate of the device.

do_ports
List[nidaqmx.system._collections.PhysicalChannelCollection] – Indicates a collection that contains all the digital output ports available on the device.

do_trig_usage
List[*nidaqmx.constants.TriggerUsage*] – Indicates the triggers supported by this device for digital output tasks.

name
str – Specifies the name of this device.

num_dma_chans
int – Indicates the number of DMA channels on the device.

pci_bus_num
int – Indicates the PCI bus number of the device.

pci_dev_num
int – Indicates the PCI slot number of the device.

product_category
nidaqmx.constants.ProductCategory – Indicates the product category of the device. This category corresponds to the category displayed in MAX when creating NI-DAQmx simulated devices.

product_num
int – Indicates the unique hardware identification number for the device.

product_type
str – Indicates the product name of the device.

pxi_chassis_num
int – Indicates the PXI chassis number of the device, as identified in MAX.

pxi_slot_num
int – Indicates the PXI slot number of the device.

reserve_network_device(*override_reservation=None*)
Reserves the Network DAQ device for the current host. Reservation is required to run NI-DAQmx tasks, and the device must be added in MAX before it can be reserved.

Parameters **override_reservation** (*Optional[bool]*) – Indicates if an existing reservation on the device should be overridden by this reservation. By default, this parameter is set to false.

reset_device()
Immediately aborts all active tasks associated with a device, disconnects any routes, and returns the device to an initialized state. Aborting a task immediately terminates the currently active operation, such as a read or a write. Aborting a task puts the task into an unstable but recoverable state. To recover the task, use DAQmx Start to restart the task or use DAQmx Stop to reset the task without starting it.

self_test_device()
Performs a brief test of device resources. If a failure occurs, refer to your device documentation for more information.

tcpip_ethernet_ip
str – Indicates the IPv4 address of the Ethernet interface in dotted decimal format. This property returns 0.0.0.0 if the Ethernet interface cannot acquire an address.

tcpip_hostname
str – Indicates the IPv4 hostname of the device.

tcpip_wireless_ip
str – Indicates the IPv4 address of the 802.11 wireless interface in dotted decimal format. This property returns 0.0.0.0 if the wireless interface cannot acquire an address.

tedshwteds_supported
bool – Indicates whether the device supports hardware TEDS.

terminals
List[str] – Indicates a list of all terminals on the device.

unreserve_network_device()
Unreserves or releases a Network DAQ device previously reserved by the host.

nidaqmx.system.physical_channel

class nidaqmx.system.physical_channel.**PhysicalChannel**(*name*)
Bases: object

Represents a DAQmx physical channel.

__init__(*name*)

Parameters **name** (*str*) – Specifies the name of the physical channel.

__weakref__
list of weak references to the object (if defined)

ai_input_srcs
List[str] – Indicates the list of input sources supported by the channel. Channels may support using the signal from the I/O connector or one of several calibration signals.

ai_meas_types
List[*nidaqmx.constants.UsageTypeAI*] – Indicates the measurement types supported by the channel.

ai_term_cfgs
List[*nidaqmx.constants.TerminalConfiguration*] – Indicates the list of terminal configurations supported by the channel.

ao_manual_control_amplitude
float – Indicates the current value of the front panel amplitude control for the physical channel in volts.

ao_manual_control_enable
bool – Specifies if you can control the physical channel externally via a manual control located on the device. You cannot simultaneously control a channel manually and with NI-DAQmx.

ao_manual_control_freq
float – Indicates the current value of the front panel frequency control for the physical channel in hertz.

ao_manual_control_short_detected
bool – Indicates whether the physical channel is currently disabled due to a short detected on the channel.

ao_output_types
List[*nidaqmx.constants.UsageTypeAO*] – Indicates the output types supported by the channel.

ao_power_amp_channel_enable
bool – Specifies whether to enable or disable a channel for amplification. This property can also be used to check if a channel is enabled.

ao_power_amp_gain
float – Indicates the calibrated gain of the channel.

ao_power_amp_offset
float – Indicates the calibrated offset of the channel in volts.

ao_power_amp_overcurrent
bool – Indicates if the channel detected an overcurrent condition.

ao_power_amp_scaling_coeff
List[float] – Indicates the coefficients of a polynomial equation used to scale from pre-amplified values.

ao_power_up_output_types
List[*nidaqmx.constants.AOPowerUpOutputBehavior*] – Indicates the power up output types supported by the channel.

ao_term_cfgs
List[*nidaqmx.constants.TerminalConfiguration*] – Indicates the list of terminal configurations supported by the channel.

ci_meas_types
List[*nidaqmx.constants.UsageTypeCI*] – Indicates the measurement types supported by the channel.

clear_teds()
Removes TEDS information from the physical channel you specify. This function temporarily overrides any TEDS configuration for the physical channel that you performed in MAX.

co_output_types
List[*nidaqmx.constants.UsageTypeCO*] – Indicates the output types supported by the channel.

configure_teds(*file_path=u''*)
Associates TEDS information with the physical channel you specify. If you do not specify the filename of a data sheet in the **file_path** input, this function attempts to find a TEDS sensor connected to the physical

channel. This function temporarily overrides any TEDS configuration for the physical channel that you performed in MAX.

Parameters **file_path** (*Optional[str]*) – Is the path to a Virtual TEDS data sheet that you want to associate with the physical channel. If you do not specify anything for this input, this function attempts to find a TEDS sensor connected to the physical channel.

di_change_detect_supported
bool – Indicates if the change detection timing type is supported for the digital input physical channel.

di_port_width
int – Indicates in bits the width of digital input port.

di_samp_clk_supported
bool – Indicates if the sample clock timing type is supported for the digital input physical channel.

di_samp_modes
List[*nidaqmx.constants.AcquisitionType*] – Indicates the sample modes supported by devices that support sample clocked digital input.

do_port_width
int – Indicates in bits the width of digital output port.

do_samp_clk_supported
bool – Indicates if the sample clock timing type is supported for the digital output physical channel.

do_samp_modes
List[*nidaqmx.constants.AcquisitionType*] – Indicates the sample modes supported by devices that support sample clocked digital output.

name
str – Specifies the name of this physical channel.

teds_bit_stream
List[int] – Indicates the TEDS binary bitstream without checksums.

teds_mfg_id
int – Indicates the manufacturer ID of the sensor.

teds_model_num
int – Indicates the model number of the sensor.

teds_serial_num
int – Indicates the serial number of the sensor.

teds_template_ids
List[int] – Indicates the IDs of the templates in the bitstream in **teds_bit_stream**.

teds_version_letter
str – Indicates the version letter of the sensor.

teds_version_num
int – Indicates the version number of the sensor.

write_to_teds_from_array (*bit_stream=None*, *basic_teds_options=<WriteBasicTEDSOptions.DO_NOT_WRITE: 12540>*)
Writes data from a 1D list of 8-bit unsigned integers to the TEDS sensor.

Parameters

- **bit_stream** (*Optional[List[int]]*) – Is the TEDS bitstream to write to the sensor. This bitstream must be constructed according to the IEEE 1451.4 specification.

- **basic_teds_options** (*Optional[nidaqmx.constants.WriteBasicTEDSOptions]*) – Specifies how to handle basic TEDS data in the bitstream.

write_to_teds_from_file(*file_path=u'', basic_teds_options=<WriteBasicTEDSOptions.DO_NOT_WRITE: 12540>*)

Writes data from a virtual TEDS file to the TEDS sensor.

Parameters

- **file_path** (*Optional[str]*) – Specifies the filename of a virtual TEDS file that contains the bitstream to write.
- **basic_teds_options** (*Optional[nidaqmx.constants.WriteBasicTEDSOptions]*) – Specifies how to handle basic TEDS data in the bitstream.

nidaqmx.system.storage

nidaqmx.system.persisted_channel

class nidaqmx.system.storage.persisted_channel.**PersistedChannel**(*name*)

Bases: object

Represents a saved DAQmx global channel.

Use the DAQmx Persisted Channel properties to query information about programmatically saved global channels.

__init__(*name*)

Parameters **name** – Specifies the name of the global channel.

__weakref__

list of weak references to the object (if defined)

allow_interactive_deletion

bool – Indicates whether the global channel can be deleted through MAX.

allow_interactive_editing

bool – Indicates whether the global channel can be edited in the DAQ Assistant.

author

str – Indicates the author of the global channel.

delete()

Deletes this global channel from MAX.

This function does not remove the global channel from tasks that use it.

nidaqmx.system.persisted_scale

class nidaqmx.system.storage.persisted_scale.**PersistedScale**(*name*)

Bases: object

Represents a saved DAQmx custom scale.

Use the DAQmx Persisted Scale properties to query information about programmatically saved custom scales.

__init__(*name*)

Parameters **name** – Specifies the name of the saved scale.

__weakref__
list of weak references to the object (if defined)

allow_interactive_deletion
bool – Indicates whether the custom scale can be deleted through MAX.

allow_interactive_editing
bool – Indicates whether the custom scale can be edited in the DAQ Assistant.

author
str – Indicates the author of the custom scale.

delete()
Deletes this custom scale from MAX.

This function does not remove the custom scale from virtual channels that use it.

load()
Loads this custom scale.

Returns Indicates the loaded Scale object.

Return type *nidaqmx.scale.Scale*

nidaqmx.system.persisted_task

class nidaqmx.system.storage.persisted_task.**PersistedTask**(*name*)
Bases: object

Represents a saved DAQmx task.

Use the DAQmx Persisted Task properties to query information about programmatically saved tasks.

__init__(*name*)

Parameters **name** – Specifies the name of the saved task.

__weakref__
list of weak references to the object (if defined)

allow_interactive_deletion
bool – Indicates whether the task can be deleted through MAX.

allow_interactive_editing
bool – Indicates whether the task can be edited in the DAQ Assistant.

author
str – Indicates the author of the task.

delete()
Deletes this task from MAX.

This function does not clear the copy of the task stored in memory. Use the DAQmx Clear Task function to clear that copy of the task.

load()
Loads this saved task.

If you use this function to load a task, you must use DAQmx Clear Task to destroy it.

Returns Indicates the loaded Task object.

Return type *nidaqmx.task.Task*

nidaqmx.system.watchdog

class `nidaqmx.system.watchdog.`**`WatchdogTask`**(*device_name*, *task_name=u''*, *timeout=10*)
Bases: `object`

Represents the watchdog configurations for a DAQmx task.

`__init__`(*device_name*, *task_name=u''*, *timeout=10*)
Creates and configures a task that controls the watchdog timer of a device. The timer activates when you start the task.

Use the DAQmx Configure Watchdog Expiration States functions to configure channel expiration states. This class does not program the watchdog timer on a real-time controller.

Parameters

- **`device_name`** (*`str`*) – Specifies is the name as configured in MAX of the device to which this operation applies.
- **`task_name`** (*`str`*) – Specifies the name to assign to the task. If you use this constructor in a loop and specify a name for the task, you must use the DAQmx Clear Task method within the loop after you are finished with the task. Otherwise, NI-DAQmx attempts to create multiple tasks with the same name, which results in an error.
- **`timeout`** (*`float`*) – Specifies the amount of time in seconds until the watchdog timer expires. A value of -1 means the internal timer never expires. Set this input to -1 if you use an Expiration Trigger to expire the watchdog task. If this time elapses, the device sets the physical channels to the states you specify with the digital physical channel expiration states input.

`__weakref__`
list of weak references to the object (if defined)

`cfg_watchdog_ao_expir_states`(*expiration_states*)
Configures the expiration states for an analog watchdog timer task.

Parameters **`expiration_states`** – (List[nidaqmx.system.watchdog.AOExpirationState]): Contains the states to which to set analog physical channels when the watchdog timer expires. Each element of the list contains an analog physical channel name, the corresponding expiration state, and the output type for that analog physical channel. The units of "expiration state" must be specified in volts for an analog output voltage expiration state, or amps for an analog output current expiration state.

physical_channel (str): Specifies the analog output channel to modify. You cannot modify dedicated analog input lines.

expiration_state (float): Specifies the value to set the channel to upon expiration.

output_type (nidaqmx.constants.WatchdogAOExpirState): Specifies the output type of the physical channel.

Returns Indicates the list of objects representing the configured expiration states.

Return type List[*nidaqmx.system._watchdog_modules.expiration_state.ExpirationState*]

`cfg_watchdog_co_expir_states`(*expiration_states*)
Configures the expiration states for a counter watchdog timer task.

Parameters **`expiration_states`** – (List[nidaqmx.system.watchdog.COExpirationState]): Contains the states to which to set counter physical channels when the watchdog timer expires. Each element of the list contains a counter physical channel name and the corresponding state for that counter physical channel.

physical_channel (str): Specifies the counter output channel to modify. You cannot modify dedicated counter input lines.

expiration_state (nidaqmx.constants.WatchdogCOExpirState): Specifies the value to set the channel to upon expiration.

Returns Indicates the list of objects representing the configured expiration states.

Return type List[*nidaqmx.system._watchdog_modules.expiration_state.ExpirationState*]

`cfg_watchdog_do_expir_states`(*expiration_states*)
Configures the expiration states for a digital watchdog timer task.

Parameters **`expiration_states`** – (List[nidaqmx.system.watchdog.DOExpirationState]): Contains the states to which to set digital physical channels when the watchdog timer expires. Each element of the list contains a digital physical channel name and the corresponding state for that digital physical channel.

physical_channel (str): Specifies the digital output channel to modify. You cannot modify dedicated digital input lines.

expiration_state (nidaqmx.constants.Level): Specifies the value to set the channel to upon expiration.

Returns Indicates the list of objects representing the configured expiration states.

Return type List[*nidaqmx.system._watchdog_modules.expiration_state.ExpirationState*]

`clear_expiration`()
Unlock a device whose watchdog timer expired.

This function does not program the watchdog timer on a real-time controller. Use the Real-Time Watchdog VIs to program the watchdog timer on a real-time controller.

`close`()
Clears the task.

Before clearing, this method aborts the task, if necessary, and releases any resources the task reserved. You cannot use a task after you clear it unless you recreate the task.

If you create a DAQmx Task object within a loop, use this method within the loop after you are finished with the task to avoid allocating unnecessary memory.

`control`(*action*)
Alters the state of a task according to the action you specify.

Parameters **`action`** (`nidaqmx.constants.TaskMode`) – Specifies how to alter the task state.

`expir_trig_dig_edge_edge`
`nidaqmx.constants.Edge` – Specifies on which edge of a digital signal to expire the watchdog task.

`expir_trig_dig_edge_src`
str – Specifies the name of a terminal where a digital signal exists to use as the source of the Expiration Trigger.

`expir_trig_trig_on_network_conn_loss`
bool – Specifies the watchdog timer behavior when the network connection is lost between the host and the chassis. If set to true, the watchdog timer expires when the chassis detects the loss of network connection.

`expir_trig_trig_type`
`nidaqmx.constants.TriggerType` – Specifies the type of trigger to use to expire a watchdog task.

`expiration_states`

nidaqmx.system._watchdog_modules.expiration_states_collection. ExpirationStatesCollection:

Gets the collection of expiration states for this watchdog task.

expired
bool – Indicates if the watchdog timer expired. You can read this property only while the task is running.

name
str – Indicates the name of the task.

reset_timer()
Reset the internal timer. You must continually reset the internal timer to prevent it from timing out and locking the device.

This function does not program the watchdog timer on a real-time controller. Use the Real-Time Watchdog VIs to program the watchdog timer on a real-time controller.

start()
Transitions the task to the running state to begin the measurement or generation. Using this method is required for some applications and is optional for others.

stop()
Stops the task and returns it to the state the task was in before the DAQmx Start Task method ran.

timeout
float – Specifies in seconds the amount of time until the watchdog timer expires. A value of -1 means the internal timer never expires. Set this input to -1 if you use an Expiration Trigger to expire the watchdog task.

nidaqmx.system.expiration_state

class nidaqmx.system._watchdog_modules.expiration_state.**ExpirationState**(*task_handle*, *physical_channel*)

Bases: object

Represents a DAQmx Watchdog expiration state.

expir_states_ao_state
float – Specifies the state to set the analog output physical channels when the watchdog task expires.

expir_states_ao_type
nidaqmx.constants.WatchdogAOExpirState – Specifies the output type of the analog output physical channels when the watchdog task expires.

expir_states_co_state
nidaqmx.constants.WatchdogCOExpirState – Specifies the state to set the counter output channel terminal when the watchdog task expires.

expir_states_do_state
nidaqmx.constants.Level – Specifies the state to which to set the digital physical channels when the watchdog task expires. You cannot modify the expiration state of dedicated digital input physical channels.

nidaqmx.system.expiration_states_collection

class nidaqmx.system._watchdog_modules.expiration_states_collection.**ExpirationStatesCollectio**

Bases: object

Contains the collection of expiration states for a DAQmx Watchdog Task.

This class defines methods that implements a container object.

nidaqmx.task

class `nidaqmx.task.Task`(*new_task_name=u''*)

Bases: `object`

Represents a DAQmx Task.

`__init__`(*new_task_name=u''*)

Creates a DAQmx task.

Parameters **`new_task_name`** (*`Optional[str]`*) – Specifies the name to assign to the task.

If you use this method in a loop and specify a name for the task, you must use the DAQmx Clear Task method within the loop after you are finished with the task. Otherwise, NI-DAQmx attempts to create multiple tasks with the same name, which results in an error.

`__weakref__`

list of weak references to the object (if defined)

`add_global_channels`(*global_channels*)

Adds global virtual channels from MAX to the given task.

Parameters **`global_channels`** (*`List[nidaqmx.system.storage.persisted_channel.PersistedChannel]`*) – Specifies the channels to add to the task.

These channels must be valid channels available from MAX. If you pass an invalid channel, NI-DAQmx returns an error. This value is ignored if it is empty.

`ai_channels`

`nidaqmx._task_modules.ai_channel_collection.AIChannelCollection` – Gets the collection of analog input channels for this task.

`ao_channels`

`nidaqmx._task_modules.ao_channel_collection.AOChannelCollection` – Gets the collection of analog output channels for this task.

`channel_names`

List[str] – Indicates the names of all virtual channels in the task.

`channels`

`nidaqmx._task_modules.channels.channel.Channel` – Specifies a channel object that represents the entire list of virtual channels in this task.

`ci_channels`

`nidaqmx._task_modules.ci_channel_collection.CIChannelCollection` – Gets the collection of counter input channels for this task.

`close`()

Clears the task.

Before clearing, this method aborts the task, if necessary, and releases any resources the task reserved. You cannot use a task after you clear it unless you recreate the task.

If you create a DAQmx Task object within a loop, use this method within the loop after you are finished with the task to avoid allocating unnecessary memory.

co_channels
nidaqmx._task_modules.co_channel_collection.COChannelCollection – Gets the collection of counter output channels for this task.

control (*action*)
Alters the state of a task according to the action you specify.

Parameters **action** (nidaqmx.constants.TaskMode) – Specifies how to alter the task state.

devices
List[*nidaqmx.system.device.Device*] – Indicates a list of Device objects representing all the devices in the task.

di_channels
nidaqmx._task_modules.di_channel_collection.DIChannelCollection – Gets the collection of digital input channels for this task.

do_channels
nidaqmx._task_modules.do_channel_collection.DOChannelCollection – Gets the collection of digital output channels for this task.

export_signals
nidaqmx._task_modules.export_signals.ExportSignals – Gets the exported signal configurations for the task.

in_stream
nidaqmx._task_modules.in_stream.InStream – Gets the read configurations for the task.

is_task_done ()
Queries the status of the task and indicates if it completed execution. Use this function to ensure that the specified operation is complete before you stop the task.

Returns Indicates if the measurement or generation completed.

Return type bool

name
str – Indicates the name of the task.

number_of_channels
int – Indicates the number of virtual channels in the task.

number_of_devices
int – Indicates the number of devices in the task.

out_stream
nidaqmx._task_modules.out_stream.OutStream – Gets the write configurations for the task.

read (*number_of_samples_per_channel=<nidaqmx.task.UnsetNumSamplesSentinel object>, timeout=10.0*)
Reads samples from the task or virtual channels you specify.

This read method is dynamic, and is capable of inferring an appropriate return type based on these factors: - The channel type of the task. - The number of channels to read. - The number of samples per channel.

The data type of the samples returned is independently determined by the channel type of the task.

For digital input measurements, the data type of the samples returned is determined by the line grouping format of the digital lines. If the line grouping format is set to “one channel for all lines”, the data type of the samples returned is int. If the line grouping format is set to “one channel per line”, the data type of the samples returned is boolean.

If you do not set the number of samples per channel, this method assumes one sample was requested. This method then returns either a scalar (1 channel to read) or a list (N channels to read).

If you set the number of samples per channel to ANY value (even 1), this method assumes multiple samples were requested. This method then returns either a list (1 channel to read) or a list of lists (N channels to read).

Parameters

- **number_of_samples_per_channel** (*Optional[int]*) – Specifies the number of samples to read. If this input is not set, assumes samples to read is 1. Conversely, if this input is set, assumes there are multiple samples to read.

  If you set this input to nidaqmx.constants. READ_ALL_AVAILABLE, NI-DAQmx determines how many samples to read based on if the task acquires samples continuously or acquires a finite number of samples.

  If the task acquires samples continuously and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, this method reads all the samples currently available in the buffer.

  If the task acquires a finite number of samples and you set this input to nidaqmx.constants.READ_ALL_AVAILABLE, the method waits for the task to acquire all requested samples, then reads those samples. If you set the "read_all_avail_samp" property to True, the method reads the samples currently available in the buffer and does not wait for the task to acquire all requested samples.

- **timeout** (*Optional[float]*) – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

Returns

The samples requested in the form of a scalar, a list, or a list of lists. See method docstring for more info.

NI-DAQmx scales the data to the units of the measurement, including any custom scaling you apply to the channels. Use a DAQmx Create Channel method to specify these units.

Return type dynamic

Example

```
>>> task = Task()
>>> task.ai_channels.add_voltage_channel('Dev1/ai0:3')
>>> data = task.read()
>>> type(data)
<type 'list'>
>>> type(data[0])
<type 'float'>
```

register_done_event (*callback_method*)

Registers a callback function to receive an event when a task stops due to an error or when a finite acquisition task or finite generation task completes execution. A Done event does not occur when a task is stopped explicitly, such as by calling DAQmx Stop Task.

Parameters **callback_method** (*function*) – Specifies the function that you want DAQmx to call when the event occurs. The function you pass in this parameter must have the following prototype:

```
>>> def callback(task_handle, status, callback_data):
>>>     return 0
```

Upon entry to the callback, the task_handle parameter contains the handle to the task on which the event occurred. The status parameter contains the status of the task when the event occurred. If the status value is negative, it indicates an error. If the status value is zero, it indicates no error. If the status value is positive, it indicates a warning. The callbackData parameter contains the value you passed in the callbackData parameter of this function.

Passing None for this parameter unregisters the event callback function.

register_every_n_samples_acquired_into_buffer_event(*sample_interval*, *callback_method*)

Registers a callback function to receive an event when the specified number of samples is written from the device to the buffer. This function only works with devices that support buffered tasks.

When you stop a task explicitly any pending events are discarded. For example, if you call DAQmx Stop Task then you do not receive any pending events.

Parameters

- **sample_interval** (*int*) – Specifies the number of samples after which each event should occur.
- **callback_method** (*function*) – Specifies the function that you want DAQmx to call when the event occurs. The function you pass in this parameter must have the following prototype:

```
>>> def callback(task_handle, every_n_samples_event_type,
>>>         number_of_samples, callback_data):
>>>     return 0
```

  Upon entry to the callback, the task_handle parameter contains the handle to the task on which the event occurred. The every_n_samples_event_type parameter contains the EveryNSamplesEventType.ACQUIRED_INTO_BUFFER value. The number_of_samples parameter contains the value you passed in the sample_interval parameter of this function. The callback_data parameter contains the value you passed in the callback_data parameter of this function.

  Passing None for this parameter unregisters the event callback function.

register_every_n_samples_transferred_from_buffer_event(*sample_interval*, *callback_method*)

Registers a callback function to receive an event when the specified number of samples is written from the buffer to the device. This function only works with devices that support buffered tasks.

When you stop a task explicitly any pending events are discarded. For example, if you call DAQmx Stop Task then you do not receive any pending events.

Parameters

- **sample_interval** (*int*) – Specifies the number of samples after which each event should occur.
- **callback_method** (*function*) – Specifies the function that you want DAQmx to call when the event occurs. The function you pass in this parameter must have the following prototype:

```
>>> def callback(task_handle, every_n_samples_event_type,
>>>         number_of_samples, callback_data):
>>>     return 0
```

Upon entry to the callback, the task_handle parameter contains the handle to the task on which the event occurred. The every_n_samples_event_type parameter contains the EveryNSamplesEventType.TRANSFERRED_FROM_BUFFER value. The number_of_samples parameter contains the value you passed in the sample_interval parameter of this function. The callback_data parameter contains the value you passed in the callback_data parameter of this function.

Passing None for this parameter unregisters the event callback function.

register_signal_event (*signal_type*, *callback_method*)

Registers a callback function to receive an event when the specified hardware event occurs.

When you stop a task explicitly any pending events are discarded. For example, if you call DAQmx Stop Task then you do not receive any pending events.

Parameters

- **signal_type** (`nidaqmx.constants.Signal`) – Specifies the type of signal for which you want to receive results.
- **callback_method** (*function*) – Specifies the function that you want DAQmx to call when the event occurs. The function you pass in this parameter must have the following prototype:

```
>>> def callback(task_handle, signal_type, callback_data):
>>>     return 0
```

Upon entry to the callback, the task_handle parameter contains the handle to the task on which the event occurred. The signal_type parameter contains the integer value you passed in the signal_type parameter of this function. The callback_data parameter contains the value you passed in the callback_data parameter of this function.

Passing None for this parameter unregisters the event callback function.

save (*save_as=u''*, *author=u''*, *overwrite_existing_task=False*, *allow_interactive_editing=True*, *allow_interactive_deletion=True*)

Saves this task and any local channels it contains to MAX.

This function does not save global channels. Use the DAQmx Save Global Channel function to save global channels.

Parameters

- **save_as** (*Optional[str]*) – Is the name to save the task, global channel, or custom scale as. If you do not specify a value for this input, NI-DAQmx uses the name currently assigned to the task, global channel, or custom scale.
- **author** (*Optional[str]*) – Is a name to store with the task, global channel, or custom scale.
- **overwrite_existing_task** (*Optional[bool]*) – Specifies whether to overwrite a task of the same name if one is already saved in MAX. If this input is False and a task of the same name is already saved in MAX, this function returns an error.
- **allow_interactive_editing** (*Optional[bool]*) – Specifies whether to allow the task, global channel, or custom scale to be edited in the DAQ Assistant. If al-

low_interactive_editing is True, the DAQ Assistant must support all task or global channel settings.

- **allow_interactive_deletion** (*Optional[bool]*) – Specifies whether to allow the task, global channel, or custom scale to be deleted through MAX.

start()

Transitions the task to the running state to begin the measurement or generation. Using this method is required for some applications and is optional for others.

If you do not use this method, a measurement task starts automatically when the DAQmx Read method runs. The autostart input of the DAQmx Write method determines if a generation task starts automatically when the DAQmx Write method runs.

If you do not use the DAQmx Start Task method and the DAQmx Stop Task method when you use the DAQmx Read method or the DAQmx Write method multiple times, such as in a loop, the task starts and stops repeatedly. Starting and stopping a task repeatedly reduces the performance of the application.

stop()

Stops the task and returns it to the state the task was in before the DAQmx Start Task method ran or the DAQmx Write method ran with the autostart input set to TRUE.

If you do not use the DAQmx Start Task method and the DAQmx Stop Task method when you use the DAQmx Read method or the DAQmx Write method multiple times, such as in a loop, the task starts and stops repeatedly. Starting and stopping a task repeatedly reduces the performance of the application.

timing

nidaqmx._task_modules.timing.Timing – Gets the timing configurations for the task.

triggers

nidaqmx._task_modules.triggers.Triggers – Gets the trigger configurations for the task.

wait_until_done(*timeout=10.0*)

Waits for the measurement or generation to complete.

Use this method to ensure that the specified operation is complete before you stop the task.

Parameters **timeout** (*Optional[float]*) – Specifies the maximum amount of time in seconds to wait for the measurement or generation to complete. This method returns an error if the time elapses. The default is 10. If you set timeout (sec) to nidaqmx.WAIT_INFINITELY, the method waits indefinitely. If you set timeout (sec) to 0, the method checks once and returns an error if the measurement or generation is not done.

write(*data*, *auto_start=<nidaqmx.task.UnsetAutoStartSentinel object>*, *timeout=10.0*)

Writes samples to the task or virtual channels you specify.

This write method is dynamic, and is capable of accepting the samples to write in the various forms for most operations:

- Scalar: Single sample for 1 channel.
- List/1D numpy.ndarray: Multiple samples for 1 channel or 1 sample for multiple channels.
- List of lists/2D numpy.ndarray: Multiple samples for multiple channels.

The data type of the samples passed in must be appropriate for the channel type of the task.

For counter output pulse operations, this write method only accepts samples in these forms:

- Scalar CtrFreq, CtrTime, CtrTick (from nidaqmx.types): Single sample for 1 channel.
- List of CtrFreq, CtrTime, CtrTick (from nidaqmx.types): Multiple samples for 1 channel or 1 sample for multiple channels.

If the task uses on-demand timing, this method returns only after the device generates all samples. On-demand is the default timing type if you do not use the timing property on the task to configure a sample timing type. If the task uses any timing type other than on-demand, this method returns immediately and does not wait for the device to generate all samples. Your application must determine if the task is done to ensure that the device generated all samples.

Parameters

- **`data`** (*`dynamic`*) – Contains the samples to write to the task.

  The data you write must be in the units of the generation, including any custom scales. Use the DAQmx Create Channel methods to specify these units.

- **`auto_start`** (*`Optional[bool]`*) – Specifies if this method automatically starts the task if you did not explicitly start it with the DAQmx Start Task method.

  The default value of this parameter depends on whether you specify one sample or many samples to write to each channel. If one sample per channel was specified, the default value is True. If multiple samples per channel were specified, the default value is False.

- **`timeout`** (*`Optional[float]`*) – Specifies the amount of time in seconds to wait for the method to write all samples. NI-DAQmx performs a timeout check only if the method must wait before it writes data. This method returns an error if the time elapses. The default timeout is 10 seconds. If you set timeout to nidaqmx.constants.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to write the submitted samples. If the method could not write all the submitted samples, it returns an error and the number of samples successfully written.

Returns Specifies the actual number of samples this method successfully wrote.

Return type int

nidaqmx.task.channel

class `nidaqmx._task_modules.channels.channel.`**`Channel`**(*task_handle*, *virtual_or_physical_name*)

Bases: `object`

Represents virtual channel or a list of virtual channels.

`chan_type`
`nidaqmx.constants.ChannelType` – Indicates the type of the virtual channel.

`channel_names`
List[str] – Specifies the unflattened list of the virtual channels.

`description`
str – Specifies a user-defined description for the channel.

`is_global`
bool – Indicates whether the channel is a global channel.

`name`
str – Specifies the name of the virtual channel this object represents.

`physical_channel`
`nidaqmx.system.physical_channel.PhysicalChannel` – Specifies the name of the physical channel upon which this virtual channel is based.

`save`(*save_as=u''*, *author=u''*, *overwrite_existing_channel=False*, *allow_interactive_editing=True*, *allow_interactive_deletion=True*)
Saves this local or global channel to MAX as a global channel.

Parameters

- **save_as** (*Optional[str]*) – Is the name to save the task, global channel, or custom scale as. If you do not specify a value for this input, NI-DAQmx uses the name currently assigned to the task, global channel, or custom scale.
- **author** (*Optional[str]*) – Is a name to store with the task, global channel, or custom scale.
- **overwrite_existing_channel** (*Optional[bool]*) – Specifies whether to overwrite a global channel of the same name if one is already saved in MAX. If this input is False and a global channel of the same name is already saved in MAX, this function returns an error.
- **allow_interactive_editing** (*Optional[bool]*) – Specifies whether to allow the task, global channel, or custom scale to be edited in the DAQ Assistant. If allow_interactive_editing is True, the DAQ Assistant must support all task or global channel settings.
- **allow_interactive_deletion** (*Optional[bool]*) – Specifies whether to allow the task, global channel, or custom scale to be deleted through MAX.

nidaqmx.task.ai_channel

class nidaqmx._task_modules.channels.ai_channel.**AIChannel**(*task_handle*, *virtual_or_physical_name*)

Bases: *nidaqmx._task_modules.channels.channel.Channel*

Represents one or more analog input virtual channels and their properties.

ai_ac_excit_freq

float – Specifies the AC excitation frequency in Hertz.

ai_ac_excit_sync_enable

bool – Specifies whether to synchronize the AC excitation source of the channel to that of another channel. Synchronize the excitation sources of multiple channels to use multichannel sensors. Set this property to False for the master channel and to True for the slave channels.

ai_ac_excit_wire_mode

nidaqmx.constants.ACExcitWireMode – Specifies the number of leads on the LVDT or RVDT. Some sensors require you to tie leads together to create a four- or five- wire sensor. Refer to the sensor documentation for more information.

ai_accel_4_wire_dc_voltage_sensitivity

float – Specifies the sensitivity of the 4 wire DC voltage acceleration sensor connected to the channel. This value is the units you specify with AI.Accel.4WireDCVoltage.SensitivityUnits. Refer to the sensor documentation to determine this value.

ai_accel_4_wire_dc_voltage_sensitivity_units

nidaqmx.constants.AccelSensitivityUnits – Specifies the units of AI.Accel.4WireDCVoltage.Sensitivity.

ai_accel_charge_sensitivity

float – Specifies the sensitivity of the charge acceleration sensor connected to the channel. This value is the units you specify with AI.Accel.Charge.SensitivityUnits. Refer to the sensor documentation to determine this value.

ai_accel_charge_sensitivity_units

nidaqmx.constants.AccelChargeSensitivityUnits – Specifies the units of AI.Accel.Charge.Sensitivity.

ai_accel_sensitivity
float – Specifies the sensitivity of the accelerometer. This value is in the units you specify with **ai_accel_sensitivity_units**. Refer to the sensor documentation to determine this value.

ai_accel_sensitivity_units
nidaqmx.constants.AccelSensitivityUnits – Specifies the units of **ai_accel_sensitivity**.

ai_accel_units
nidaqmx.constants.AccelUnits – Specifies the units to use to return acceleration measurements from the channel.

ai_acceld_b_ref
float – Specifies the decibel reference level in the units of the channel. When you read samples as a waveform, the decibel reference level is included in the waveform attributes.

ai_adc_custom_timing_mode
int – Specifies the timing mode of the ADC when **ai_adc_timing_mode** is **ADCTimingMode.CUSTOM**.

ai_adc_timing_mode
nidaqmx.constants.ADCTimingMode – Specifies the ADC timing mode, controlling the tradeoff between speed and effective resolution. Some ADC timing modes provide increased powerline noise rejection. On devices that have an AI Convert clock, this setting affects both the maximum and default values for **ai_conv_rate**. You must use the same ADC timing mode for all channels on a device, but you can use different ADC timing modes for different devices in the same task.

ai_atten
float – Specifies the amount of attenuation to use.

ai_auto_zero_mode
nidaqmx.constants.AutoZeroType – Specifies how often to measure ground. NI-DAQmx subtracts the measured ground voltage from every sample.

ai_averaging_win_size
int – Specifies the number of samples to average while acquiring data. Increasing the number of samples to average reduces noise in your measurement.

ai_bridge_balance_coarse_pot
int – Specifies by how much to compensate for offset in the signal. This value can be between 0 and 127.

ai_bridge_balance_fine_pot
int – Specifies by how much to compensate for offset in the signal. This value can be between 0 and 4095.

ai_bridge_cfg
nidaqmx.constants.BridgeConfiguration – Specifies the type of Wheatstone bridge connected to the channel.

ai_bridge_electrical_units
nidaqmx.constants.BridgeElectricalUnits – Specifies from which electrical unit to scale data. Select the same unit that the sensor data sheet or calibration certificate uses for electrical values.

ai_bridge_initial_ratio
float – Specifies in volts per volt the ratio of output voltage from the bridge to excitation voltage supplied to the bridge while not under load. NI-DAQmx subtracts this value from any measurements before applying scaling equations. If you set **ai_bridge_initial_voltage**, NI-DAQmx coerces this property to **ai_bridge_initial_voltage** divided by **ai_excit_actual_val**. If you set this property, NI-DAQmx coerces **ai_bridge_initial_voltage** to the value of this property times **ai_excit_actual_val**. If you set both this property and **ai_bridge_initial_voltage**, and their values conflict, NI-DAQmx returns an error. To avoid this error, reset one property to its default value before setting the other.

ai_bridge_initial_voltage
float – Specifies in volts the output voltage of the bridge while not under load. NI-DAQmx subtracts

this value from any measurements before applying scaling equations. If you set **ai_bridge_initial_ratio**, NI-DAQmx coerces this property to **ai_bridge_initial_ratio** times **ai_excit_actual_val**. This property is set by DAQmx Perform Bridge Offset Nulling Calibration. If you set this property, NI-DAQmx coerces **ai_bridge_initial_ratio** to the value of this property divided by **ai_excit_actual_val**. If you set both this property and **ai_bridge_initial_ratio**, and their values conflict, NI- DAQmx returns an error. To avoid this error, reset one property to its default value before setting the other.

ai_bridge_nom_resistance
float – Specifies in ohms the resistance of the bridge while not under load.

ai_bridge_physical_units
`nidaqmx.constants.BridgePhysicalUnits` – Specifies to which physical unit to scale electrical data. Select the same unit that the sensor data sheet or calibration certificate uses for physical values.

ai_bridge_poly_forward_coeff
List[float] – Specifies an list of coefficients for the polynomial that converts electrical values to physical values. Each element of the list corresponds to a term of the equation. For example, if index three of the list is 9, the fourth term of the equation is 9x^3.

ai_bridge_poly_reverse_coeff
List[float] – Specifies an list of coefficients for the polynomial that converts physical values to electrical values. Each element of the list corresponds to a term of the equation. For example, if index three of the list is 9, the fourth term of the equation is 9x^3.

ai_bridge_scale_type
`nidaqmx.constants.ScaleType` – Specifies the scaling type to use when scaling electrical values from the sensor to physical units.

ai_bridge_shunt_cal_enable
bool – Specifies whether to enable a shunt calibration switch. Use **ai_bridge_shunt_cal_select** to select the switch(es) to enable.

ai_bridge_shunt_cal_gain_adjust
float – Specifies the result of a shunt calibration. This property is set by DAQmx Perform Shunt Calibration. NI-DAQmx multiplies data read from the channel by the value of this property. This value should be close to 1.0.

ai_bridge_shunt_cal_select
`nidaqmx.constants.ShuntCalSelect` – Specifies which shunt calibration switch(es) to enable. Use **ai_bridge_shunt_cal_enable** to enable the switch(es) you specify with this property.

ai_bridge_shunt_cal_shunt_cal_a_actual_resistance
float – Specifies in ohms the actual value of the internal shunt calibration A resistor.

ai_bridge_shunt_cal_shunt_cal_a_resistance
float – Specifies in ohms the desired value of the internal shunt calibration A resistor.

ai_bridge_shunt_cal_shunt_cal_a_src
`nidaqmx.constants.BridgeShuntCalSource` – Specifies whether to use internal or external shunt when Shunt Cal A is selected.

ai_bridge_shunt_cal_shunt_cal_b_actual_resistance
float – Specifies in ohms the actual value of the internal shunt calibration B resistor.

ai_bridge_shunt_cal_shunt_cal_b_resistance
float – Specifies in ohms the desired value of the internal shunt calibration B resistor.

ai_bridge_table_electrical_vals
List[float] – Specifies the list of electrical values that map to the values in **ai_bridge_table_physical_vals**. Specify this value in the unit indicated by **ai_bridge_electrical_units**.

ai_bridge_table_physical_vals
List[float] – Specifies the list of physical values that map to the values in **ai_bridge_table_electrical_vals**. Specify this value in the unit indicated by **ai_bridge_physical_units**.

ai_bridge_two_point_lin_first_electrical_val
float – Specifies the first electrical value, corresponding to **ai_bridge_two_point_lin_first_physical_val**. Specify this value in the unit indicated by **ai_bridge_electrical_units**.

ai_bridge_two_point_lin_first_physical_val
float – Specifies the first physical value, corresponding to **ai_bridge_two_point_lin_first_electrical_val**. Specify this value in the unit indicated by **ai_bridge_physical_units**.

ai_bridge_two_point_lin_second_electrical_val
float – Specifies the second electrical value, corresponding to **ai_bridge_two_point_lin_second_physical_val**. Specify this value in the unit indicated by **ai_bridge_electrical_units**.

ai_bridge_two_point_lin_second_physical_val
float – Specifies the second physical value, corresponding to **ai_bridge_two_point_lin_second_electrical_val**. Specify this value in the unit indicated by **ai_bridge_physical_units**.

ai_bridge_units
`nidaqmx.constants.BridgeUnits` – Specifies in which unit to return voltage ratios from the channel.

ai_charge_units
`nidaqmx.constants.ChargeUnits` – Specifies the units to use to return charge measurements from the channel.

ai_coupling
`nidaqmx.constants.Coupling` – Specifies the coupling for the channel.

ai_current_acrms_units
`nidaqmx.constants.CurrentUnits` – Specifies the units to use to return current RMS measurements from the channel.

ai_current_shunt_loc
`nidaqmx.constants.CurrentShuntResistorLocation` – Specifies the shunt resistor location for current measurements.

ai_current_shunt_resistance
float – Specifies in ohms the external shunt resistance for current measurements.

ai_current_units
`nidaqmx.constants.CurrentUnits` – Specifies the units to use to return current measurements from the channel.

ai_custom_scale
`nidaqmx.system.scale.Scale` – Specifies the name of a custom scale for the channel.

ai_data_xfer_custom_threshold
int – Specifies the number of samples that must be in the FIFO to transfer data from the device if **ai_data_xfer_req_cond** is **InputDataTransferCondition.ONBOARD_MEMORY_CUSTOM_THRESHOLD**.

ai_data_xfer_mech
`nidaqmx.constants.DataTransferActiveTransferMode` – Specifies the data transfer mode for the device.

ai_data_xfer_req_cond
nidaqmx.constants.InputDataTransferCondition – Specifies under what condition to transfer data from the onboard memory of the device to the buffer.

ai_dc_offset
float – Specifies the DC value to add to the input range of the device. Use **ai_rng_high** and **ai_rng_low** to specify the input range. This offset is in the native units of the device .

ai_dev_scaling_coeff
List[float] – Indicates the coefficients of a polynomial equation that NI-DAQmx uses to scale values from the native format of the device to volts. Each element of the list corresponds to a term of the equation. For example, if index two of the list is 4, the third term of the equation is 4x^2. Scaling coefficients do not account for any custom scales or sensors contained by the channel.

ai_dig_fltr_bandpass_center_freq
float – Specifies the center frequency of the passband for the digital filter.

ai_dig_fltr_bandpass_width
float – Specifies the width of the passband centered around the center frequency for the digital filter.

ai_dig_fltr_coeff
List[float] – Specifies the digital filter coefficients.

ai_dig_fltr_enable
bool – Specifies whether the digital filter is enabled or disabled.

ai_dig_fltr_highpass_cutoff_freq
float – Specifies the highpass cutoff frequency of the digital filter.

ai_dig_fltr_lowpass_cutoff_freq
float – Specifies the lowpass cutoff frequency of the digital filter.

ai_dig_fltr_notch_center_freq
float – Specifies the center frequency of the stopband for the digital filter.

ai_dig_fltr_notch_width
float – Specifies the width of the stopband centered around the center frequency for the digital filter.

ai_dig_fltr_order
int – Specifies the order of the digital filter.

ai_dig_fltr_response
nidaqmx.constants.FilterResponse – Specifies the digital filter response.

ai_dig_fltr_type
nidaqmx.constants.FilterType – Specifies the digital filter type.

ai_dither_enable
bool – Specifies whether to enable dithering. Dithering adds Gaussian noise to the input signal. You can use dithering to achieve higher resolution measurements by over sampling the input signal and averaging the results.

ai_eddy_current_prox_sensitivity
float – Specifies the sensitivity of the eddy current proximity probe . This value is in the units you specify with **ai_eddy_current_prox_sensitivity_units**. Refer to the sensor documentation to determine this value.

ai_eddy_current_prox_sensitivity_units
nidaqmx.constants.EddyCurrentProxProbeSensitivityUnits – Specifies the units of **ai_eddy_current_prox_sensitivity**.

ai_eddy_current_prox_units
nidaqmx.constants.LengthUnits – Specifies the units to use to return proximity measurements from the channel.

ai_enhanced_alias_rejection_enable
bool – Specifies whether to enable enhanced alias rejection. Leave this property set to the default value for most applications.

ai_excit_actual_val
float – Specifies the actual amount of excitation supplied by an internal excitation source. If you read an internal excitation source more precisely with an external device, set this property to the value you read. NI-DAQmx ignores this value for external excitation. When performing shunt calibration, some devices set this property automatically.

ai_excit_d_cor_ac
nidaqmx.constants.ExcitationDCorAC – Specifies if the excitation supply is DC or AC.

ai_excit_idle_output_behavior
nidaqmx.constants.ExcitationIdleOutputBehavior – Specifies whether this channel will disable excitation after the task is uncommitted. Setting this to Zero Volts or Amps disables excitation after task uncommit. Setting this attribute to Maintain Existing Value leaves the excitation on after task uncommit.

ai_excit_sense
nidaqmx.constants.Sense – Specifies whether to use local or remote sense to sense excitation.

ai_excit_src
nidaqmx.constants.ExcitationSource – Specifies the source of excitation.

ai_excit_use_for_scaling
bool – Specifies if NI-DAQmx divides the measurement by the excitation. You should typically set this property to True for ratiometric transducers. If you set this property to True, set **ai_max** and **ai_min** to reflect the scaling.

ai_excit_use_multiplexed
bool – Specifies if the SCXI-1122 multiplexes the excitation to the upper half of the channels as it advances through the scan list.

ai_excit_val
float – Specifies the amount of excitation that the sensor requires. If **ai_excit_voltage_or_current** is **ExcitationVoltageOrCurrent.USE_VOLTAGE**, this value is in volts. If **ai_excit_voltage_or_current** is **ExcitationVoltageOrCurrent.USE_CURRENT**, this value is in amperes.

ai_excit_voltage_or_current
nidaqmx.constants.ExcitationVoltageOrCurrent – Specifies if the channel uses current or voltage excitation.

ai_filter_delay
float – Indicates the amount of time between when the ADC samples data and when the sample is read by the host device. This value is in the units you specify with **ai_filter_delay_units**. You can adjust this amount of time using **ai_filter_delay_adjustment**.

ai_filter_delay_adjustment
float – Specifies the amount of filter delay that gets removed if **ai_remove_filter_delay** is enabled. This delay adjustment is in addition to the value indicated by **ai_filter_delay**. This delay adjustment is in the units you specify with **ai_filter_delay_units**.

ai_filter_delay_units
nidaqmx.constants.DigitalWidthUnits – Specifies the units of **ai_filter_delay** and **ai_filter_delay_adjustment**.

ai_force_iepe_sensor_sensitivity
float – Specifies the sensitivity of the IEPE force sensor connected to the channel. Specify this value in the unit indicated by **ai_force_iepe_sensor_sensitivity_units**.

ai_force_iepe_sensor_sensitivity_units
`nidaqmx.constants.ForceIEPESensorSensitivityUnits` – Specifies the units for **ai_force_iepe_sensor_sensitivity**.

ai_force_read_from_chan
bool – Specifies whether to read from the channel if it is a cold-junction compensation channel. By default, DAQmx Read does not return data from cold-junction compensation channels. Setting this property to True forces read operations to return the cold-junction compensation channel data with the other channels in the task.

ai_force_units
`nidaqmx.constants.ForceUnits` – Specifies in which unit to return force or load measurements from the channel.

ai_freq_hyst
float – Specifies in volts a window below **ai_freq_thresh_voltage**. The input voltage must pass below **ai_freq_thresh_voltage** minus this value before NI- DAQmx recognizes a waveform repetition at **ai_freq_thresh_voltage**. Hysteresis can improve the measurement accuracy when the signal contains noise or jitter.

ai_freq_thresh_voltage
float – Specifies the voltage level at which to recognize waveform repetitions. You should select a voltage level that occurs only once within the entire period of a waveform. You also can select a voltage that occurs only once while the voltage rises or falls.

ai_freq_units
`nidaqmx.constants.FrequencyUnits` – Specifies the units to use to return frequency measurements from the channel.

ai_gain
float – Specifies a gain factor to apply to the channel.

ai_impedance
`nidaqmx.constants.Impedance1` – Specifies the input impedance of the channel.

ai_input_src
str – Specifies the source of the channel. You can use the signal from the I/O connector or one of several calibration signals. Certain devices have a single calibration signal bus. For these devices, you must specify the same calibration signal for all channels you connect to a calibration signal.

ai_lead_wire_resistance
float – Specifies in ohms the resistance of the wires that lead to the sensor.

ai_lossy_lsb_removal_compressed_samp_size
int – Specifies the number of bits to return in a raw sample when **ai_raw_data_compression_type** is set to **RawDataCompressionType.LOSSY_LSB_REMOVAL**.

ai_lowpass_cutoff_freq
float – Specifies the frequency in Hertz that corresponds to the -3dB cutoff of the filter.

ai_lowpass_enable
bool – Specifies whether to enable the lowpass filter of the channel.

ai_lowpass_switch_cap_clk_src
`nidaqmx.constants.SourceSelection` – Specifies the source of the filter clock. If you need a higher resolution for the filter, you can supply an external clock to increase the resolution. Refer to the SCXI-1141/1142/1143 User Manual for more information.

ai_lowpass_switch_cap_ext_clk_div
int – Specifies the divisor for the external clock when you set **ai_lowpass_switch_cap_clk_src** to **SourceSelection.EXTERNAL**. On the SCXI-1141, SCXI-1142, and SCXI-1143, NI-DAQmx determines the filter cutoff by using the equation f/(100*n), where f is the external frequency, and n is the external clock divisor. Refer to the SCXI-1141/1142/1143 User Manual for more information.

ai_lowpass_switch_cap_ext_clk_freq
float – Specifies the frequency of the external clock when you set **ai_lowpass_switch_cap_clk_src** to **SourceSelection.EXTERNAL**. NI-DAQmx uses this frequency to set the pre- and post- filters on the SCXI-1141, SCXI-1142, and SCXI-1143. On those devices, NI-DAQmx determines the filter cutoff by using the equation f/(100*n), where f is the external frequency, and n is the external clock divisor. Refer to the SCXI-1141/1142/1143 User Manual for more information.

ai_lowpass_switch_cap_out_clk_div
int – Specifies the divisor for the output clock. NI-DAQmx uses the cutoff frequency to determine the output clock frequency. Refer to the SCXI-1141/1142/1143 User Manual for more information.

ai_lvdt_sensitivity
float – Specifies the sensitivity of the LVDT. This value is in the units you specify with **ai_lvdt_sensitivity_units**. Refer to the sensor documentation to determine this value.

ai_lvdt_sensitivity_units
nidaqmx.constants.LVDTSensitivityUnits – Specifies the units of **ai_lvdt_sensitivity**.

ai_lvdt_units
nidaqmx.constants.LengthUnits – Specifies the units to use to return linear position measurements from the channel.

ai_max
float – Specifies the maximum value you expect to measure. This value is in the units you specify with a units property. When you query this property, it returns the coerced maximum value that the device can measure with the current settings.

ai_meas_type
nidaqmx.constants.UsageTypeAI – Indicates the measurement to take with the analog input channel and in some cases, such as for temperature measurements, the sensor to use.

ai_mem_map_enable
bool – Specifies for NI-DAQmx to map hardware registers to the memory space of the application, if possible. Normally, NI- DAQmx maps hardware registers to memory accessible only to the kernel. Mapping the registers to the memory space of the application increases performance. However, if the application accesses the memory space mapped to the registers, it can adversely affect the operation of the device and possibly result in a system crash.

ai_microphone_sensitivity
float – Specifies the sensitivity of the microphone. This value is in mV/Pa. Refer to the sensor documentation to determine this value.

ai_min
float – Specifies the minimum value you expect to measure. This value is in the units you specify with a units property. When you query this property, it returns the coerced minimum value that the device can measure with the current settings.

ai_open_chan_detect_enable
bool – Specifies whether to enable open channel detection.

ai_open_thrmcpl_detect_enable
bool – Specifies whether to apply the open thermocouple detection bias voltage to the channel. Changing the value of this property on a channel may require settling time before the data returned is valid. To compensate for this settling time, discard unsettled data or add a delay between committing and starting

the task. Refer to your device specifications for the required settling time. When open thermocouple detection is enabled, use **open_thrmcpl_chans_exist** to determine if any channels were open.

ai_pressure_units
`nidaqmx.constants.PressureUnits` – Specifies in which unit to return pressure measurements from the channel.

ai_probe_atten
float – Specifies the amount of attenuation provided by the probe connected to the channel. Specify this attenuation as a ratio.

ai_raw_data_compression_type
`nidaqmx.constants.RawDataCompressionType` – Specifies the type of compression to apply to raw samples returned from the device.

ai_raw_samp_justification
`nidaqmx.constants.DataJustification` – Indicates the justification of a raw sample from the device.

ai_raw_samp_size
int – Indicates in bits the size of a raw sample from the device.

ai_remove_filter_delay
bool – Specifies if filter delay removal is enabled on the device.

ai_resistance_cfg
`nidaqmx.constants.ResistanceConfiguration` – Specifies the resistance configuration for the channel. NI-DAQmx uses this value for any resistance-based measurements, including temperature measurement using a thermistor or RTD.

ai_resistance_units
`nidaqmx.constants.ResistanceUnits` – Specifies the units to use to return resistance measurements.

ai_resolution
float – Indicates the resolution of the analog-to-digital converter of the channel. This value is in the units you specify with **ai_resolution_units**.

ai_resolution_units
`nidaqmx.constants.ResolutionType` – Indicates the units of **ai_resolution**.

ai_rng_high
float – Specifies the upper limit of the input range of the device. This value is in the native units of the device. On E Series devices, for example, the native units is volts.

ai_rng_low
float – Specifies the lower limit of the input range of the device. This value is in the native units of the device. On E Series devices, for example, the native units is volts.

ai_rosette_strain_gage_gage_orientation
float – Specifies gage orientation in degrees with respect to the X axis.

ai_rosette_strain_gage_rosette_meas_type
`nidaqmx.constants.StrainGageRosetteMeasurementType` – Specifies the type of rosette measurement.

ai_rosette_strain_gage_rosette_type
`nidaqmx.constants.StrainGageRosetteType` – Indicates the type of rosette gage.

ai_rosette_strain_gage_strain_chans
List[str] – Indicates the raw strain channels that comprise the strain rosette.

ai_rtd_a
float – Specifies the 'A' constant of the Callendar-Van Dusen equation. NI-DAQmx requires this value when you use a custom RTD.

ai_rtd_b
float – Specifies the 'B' constant of the Callendar-Van Dusen equation. NI-DAQmx requires this value when you use a custom RTD.

ai_rtd_c
float – Specifies the 'C' constant of the Callendar-Van Dusen equation. NI-DAQmx requires this value when you use a custom RTD.

ai_rtd_r_0
float – Specifies in ohms the sensor resistance at 0 deg C. The Callendar-Van Dusen equation requires this value. Refer to the sensor documentation to determine this value.

ai_rtd_type
nidaqmx.constants.RTDType – Specifies the type of RTD connected to the channel.

ai_rvdt_sensitivity
float – Specifies the sensitivity of the RVDT. This value is in the units you specify with **ai_rvdt_sensitivity_units**. Refer to the sensor documentation to determine this value.

ai_rvdt_sensitivity_units
nidaqmx.constants.RVDTSensitivityUnits – Specifies the units of **ai_rvdt_sensitivity**.

ai_rvdt_units
nidaqmx.constants.AngleUnits – Specifies the units to use to return angular position measurements from the channel.

ai_samp_and_hold_enable
bool – Specifies whether to enable the sample and hold circuitry of the device. When you disable sample and hold circuitry, a small voltage offset might be introduced into the signal. You can eliminate this offset by using **ai_auto_zero_mode** to perform an auto zero on the channel.

ai_sound_pressure_max_sound_pressure_lvl
float – Specifies the maximum instantaneous sound pressure level you expect to measure. This value is in decibels, referenced to 20 micropascals. NI-DAQmx uses the maximum sound pressure level to calculate values in pascals for **ai_max** and **ai_min** for the channel.

ai_sound_pressure_units
nidaqmx.constants.SoundPressureUnits – Specifies the units to use to return sound pressure measurements from the channel.

ai_sound_pressured_b_ref
float – Specifies the decibel reference level in the units of the channel. When you read samples as a waveform, the decibel reference level is included in the waveform attributes. NI- DAQmx also uses the decibel reference level when converting **ai_sound_pressure_max_sound_pressure_lvl** to a voltage level.

ai_strain_force_read_from_chan
bool – Specifies whether the data is returned by DAQmx Read when set on a raw strain channel that is part of a rosette configuration.

ai_strain_gage_cfg
nidaqmx.constants.StrainGageBridgeType – Specifies the bridge configuration of the strain gages.

ai_strain_gage_gage_factor
float – Specifies the sensitivity of the strain gage. Gage factor relates the change in electrical resistance to the change in strain. Refer to the sensor documentation for this value.

ai_strain_gage_poisson_ratio
float – Specifies the ratio of lateral strain to axial strain in the material you are measuring.

ai_strain_units
`nidaqmx.constants.StrainUnits` – Specifies the units to use to return strain measurements from the channel.

ai_teds_is_teds
bool – Indicates if the virtual channel was initialized using a TEDS bitstream from the corresponding physical channel.

ai_teds_units
str – Indicates the units defined by TEDS information associated with the channel.

ai_temp_units
`nidaqmx.constants.TemperatureUnits` – Specifies the units to use to return temperature measurements from the channel.

ai_term_cfg
`nidaqmx.constants.TerminalConfiguration` – Specifies the terminal configuration for the channel.

ai_thrmcpl_cjc_chan
`nidaqmx._task_modules.channels.channel.Channel` – Indicates the channel that acquires the temperature of the cold junction if **ai_thrmcpl_cjc_src** is **CJCSource1.SCANNABLE_CHANNEL**. If the channel is a temperature channel, NI-DAQmx acquires the temperature in the correct units. Other channel types, such as a resistance channel with a custom sensor, must use a custom scale to scale values to degrees Celsius.

ai_thrmcpl_cjc_src
`nidaqmx.constants.CJCSource` – Indicates the source of cold-junction compensation.

ai_thrmcpl_cjc_val
float – Specifies the temperature of the cold junction if **ai_thrmcpl_cjc_src** is **CJCSource1.CONSTANT_USER_VALUE**. Specify this value in the units of the measurement.

ai_thrmcpl_lead_offset_voltage
float – Specifies the lead offset nulling voltage to subtract from measurements on a device. This property is ignored if open thermocouple detection is disabled.

ai_thrmcpl_scale_type
`nidaqmx.constants.ScaleType` – Specifies the method or equation form that the thermocouple scale uses.

ai_thrmcpl_type
`nidaqmx.constants.ThermocoupleType` – Specifies the type of thermocouple connected to the channel. Thermocouple types differ in composition and measurement range.

ai_thrmstr_a
float – Specifies the 'A' constant of the Steinhart-Hart thermistor equation.

ai_thrmstr_b
float – Specifies the 'B' constant of the Steinhart-Hart thermistor equation.

ai_thrmstr_c
float – Specifies the 'C' constant of the Steinhart-Hart thermistor equation.

ai_thrmstr_r_1
float – Specifies in ohms the value of the reference resistor for the thermistor if you use voltage excitation. NI-DAQmx ignores this value for current excitation.

ai_torque_units
[nidaqmx.constants.TorqueUnits] – Specifies in which unit to return torque measurements from the channel.

ai_usb_xfer_req_count
int – Specifies the maximum number of simultaneous USB transfers used to stream data. Modify this value to affect performance under different combinations of operating system and device.

ai_usb_xfer_req_size
int – Specifies the maximum size of a USB transfer request in bytes. Modify this value to affect performance under different combinations of operating system and device.

ai_velocity_iepe_sensor_sensitivity
float – Specifies the sensitivity of the IEPE velocity sensor connected to the channel. Specify this value in the unit indicated by **ai_velocity_iepe_sensor_sensitivity_units**.

ai_velocity_iepe_sensor_sensitivity_units
nidaqmx.constants.VelocityIEPESensorSensitivityUnits – Specifies the units for **ai_velocity_iepe_sensor_sensitivity**.

ai_velocity_iepe_sensord_b_ref
float – Specifies the decibel reference level in the units of the channel. When you read samples as a waveform, the decibel reference level is included in the waveform attributes.

ai_velocity_units
nidaqmx.constants.VelocityUnits – Specifies in which unit to return velocity measurements from the channel.

ai_voltage_acrms_units
nidaqmx.constants.VoltageUnits – Specifies the units to use to return voltage RMS measurements from the channel.

ai_voltage_units
nidaqmx.constants.VoltageUnits – Specifies the units to use to return voltage measurements from the channel.

ai_voltaged_b_ref
float – Specifies the decibel reference level in the units of the channel. When you read samples as a waveform, the decibel reference level is included in the waveform attributes.

chan_type
nidaqmx.constants.ChannelType – Indicates the type of the virtual channel.

channel_names
List[str] – Specifies the unflattened list of the virtual channels.

description
str – Specifies a user-defined description for the channel.

is_global
bool – Indicates whether the channel is a global channel.

name
str – Specifies the name of the virtual channel this object represents.

physical_channel
nidaqmx.system.physical_channel.PhysicalChannel – Specifies the name of the physical channel upon which this virtual channel is based.

save (*save_as=u''*, *author=u''*, *overwrite_existing_channel=False*, *allow_interactive_editing=True*, *allow_interactive_deletion=True*)
Saves this local or global channel to MAX as a global channel.

Parameters

- **save_as** (*Optional[str]*) – Is the name to save the task, global channel, or custom scale as. If you do not specify a value for this input, NI-DAQmx uses the name currently assigned to the task, global channel, or custom scale.
- **author** (*Optional[str]*) – Is a name to store with the task, global channel, or custom scale.
- **overwrite_existing_channel** (*Optional[bool]*) – Specifies whether to overwrite a global channel of the same name if one is already saved in MAX. If this input is False and a global channel of the same name is already saved in MAX, this function returns an error.
- **allow_interactive_editing** (*Optional[bool]*) – Specifies whether to allow the task, global channel, or custom scale to be edited in the DAQ Assistant. If allow_interactive_editing is True, the DAQ Assistant must support all task or global channel settings.
- **allow_interactive_deletion** (*Optional[bool]*) – Specifies whether to allow the task, global channel, or custom scale to be deleted through MAX.

nidaqmx.task.ao_channel

class nidaqmx._task_modules.channels.ao_channel.**AOChannel**(*task_handle*, *virtual_or_physical_name*)

Bases: *nidaqmx._task_modules.channels.channel.Channel*

Represents one or more analog output virtual channels and their properties.

ao_current_units
nidaqmx.constants.CurrentUnits – Specifies in what units to generate current on the channel. Write data to the channel in the units you select.

ao_custom_scale
nidaqmx.system.scale.Scale – Specifies the name of a custom scale for the channel.

ao_dac_offset_ext_src
str – Specifies the source of the DAC offset voltage if **ao_dac_offset_src** is **SourceSelection.EXTERNAL**. The valid sources for this signal vary by device.

ao_dac_offset_src
nidaqmx.constants.SourceSelection – Specifies the source of the DAC offset voltage. The value of this voltage source determines the full-scale value of the DAC.

ao_dac_offset_val
float – Specifies in volts the value of the DAC offset voltage. To achieve best accuracy, the DAC offset value should be hand calibrated.

ao_dac_ref_allow_conn_to_gnd
bool – Specifies whether to allow grounding the internal DAC reference at run time. You must set this property to True and set **ao_dac_ref_src** to **SourceSelection.INTERNAL** before you can set **ao_dac_ref_conn_to_gnd** to True.

ao_dac_ref_conn_to_gnd
bool – Specifies whether to ground the internal DAC reference. Grounding the internal DAC reference has the effect of grounding all analog output channels and stopping waveform generation across all analog output channels regardless of whether the channels belong to the current task. You can ground the internal DAC reference only when **ao_dac_ref_src** is **SourceSelection.INTERNAL** and **ao_dac_ref_allow_conn_to_gnd** is True.

ao_dac_ref_ext_src
str – Specifies the source of the DAC reference voltage if **ao_dac_ref_src** is **SourceSelection.EXTERNAL**. The valid sources for this signal vary by device.

ao_dac_ref_src
`nidaqmx.constants.SourceSelection` – Specifies the source of the DAC reference voltage. The value of this voltage source determines the full-scale value of the DAC.

ao_dac_ref_val
float – Specifies in volts the value of the DAC reference voltage. This voltage determines the full-scale range of the DAC. Smaller reference voltages result in smaller ranges, but increased resolution.

ao_dac_rng_high
float – Specifies the upper limit of the output range of the device. This value is in the native units of the device. On E Series devices, for example, the native units is volts.

ao_dac_rng_low
float – Specifies the lower limit of the output range of the device. This value is in the native units of the device. On E Series devices, for example, the native units is volts.

ao_data_xfer_mech
`nidaqmx.constants.DataTransferActiveTransferMode` – Specifies the data transfer mode for the device.

ao_data_xfer_req_cond
`nidaqmx.constants.OutputDataTransferCondition` – Specifies under what condition to transfer data from the buffer to the onboard memory of the device.

ao_dev_scaling_coeff
List[float] – Indicates the coefficients of a linear equation that NI-DAQmx uses to scale values from a voltage to the native format of the device. Each element of the list corresponds to a term of the equation. The first element of the list corresponds to the y-intercept, and the second element corresponds to the slope. Scaling coefficients do not account for any custom scales that may be applied to the channel.

ao_enhanced_image_rejection_enable
bool – Specifies whether to enable the DAC interpolation filter. Disable the interpolation filter to improve DAC signal-to- noise ratio at the expense of degraded image rejection.

ao_filter_delay
float – Specifies the amount of time between when the sample is written by the host device and when the sample is output by the DAC. This value is in the units you specify with **ao_filter_delay_units**.

ao_filter_delay_adjustment
float – Specifies an additional amount of time to wait between when the sample is written by the host device and when the sample is output by the DAC. This delay adjustment is in addition to the value indicated by **ao_filter_delay**. This delay adjustment is in the units you specify with **ao_filter_delay_units**.

ao_filter_delay_units
`nidaqmx.constants.DigitalWidthUnits` – Specifies the units of **ao_filter_delay** and **ao_filter_delay_adjustment**.

ao_func_gen_amplitude
float – Specifies the zero-to-peak amplitude of the waveform to generate in volts. Zero and negative values are valid.

ao_func_gen_fm_deviation
float – Specifies the FM deviation in hertz per volt when **ao_func_gen_modulation_type** is **ModulationType.FM**.

ao_func_gen_freq
float – Specifies the frequency of the waveform to generate in hertz.

ao_func_gen_modulation_type
`nidaqmx.constants.ModulationType` – Specifies if the device generates a modulated version of the waveform using the original waveform as a carrier and input from an external terminal as the signal.

ao_func_gen_offset
float – Specifies the voltage offset of the waveform to generate.

ao_func_gen_square_duty_cycle
float – Specifies the square wave duty cycle of the waveform to generate.

ao_func_gen_type
`nidaqmx.constants.FuncGenType` – Specifies the kind of the waveform to generate.

ao_gain
float – Specifies in decibels the gain factor to apply to the channel.

ao_idle_output_behavior
`nidaqmx.constants.AOIdleOutputBehavior` – Specifies the state of the channel when no generation is in progress.

ao_load_impedance
float – Specifies in ohms the load impedance connected to the analog output channel.

ao_max
float – Specifies the maximum value you expect to generate. The value is in the units you specify with a units property. If you try to write a value larger than the maximum value, NI- DAQmx generates an error. NI-DAQmx might coerce this value to a smaller value if other task settings restrict the device from generating the desired maximum.

ao_mem_map_enable
bool – Specifies for NI-DAQmx to map hardware registers to the memory space of the application, if possible. Normally, NI- DAQmx maps hardware registers to memory accessible only to the kernel. Mapping the registers to the memory space of the application increases performance. However, if the application accesses the memory space mapped to the registers, it can adversely affect the operation of the device and possibly result in a system crash.

ao_min
float – Specifies the minimum value you expect to generate. The value is in the units you specify with a units property. If you try to write a value smaller than the minimum value, NI- DAQmx generates an error. NI-DAQmx might coerce this value to a larger value if other task settings restrict the device from generating the desired minimum.

ao_output_impedance
float – Specifies in ohms the impedance of the analog output stage of the device.

ao_output_type
`nidaqmx.constants.UsageTypeAO` – Indicates whether the channel generates voltage, current, or a waveform.

ao_reglitch_enable
bool – Specifies whether to enable reglitching. The output of a DAC normally glitches whenever the DAC is updated with a new value. The amount of glitching differs from code to code and is generally largest at major code transitions. Reglitching generates uniform glitch energy at each code transition and provides for more uniform glitches. Uniform glitch energy makes it easier to filter out the noise introduced from glitching during spectrum analysis.

ao_resolution
float – Indicates the resolution of the digital-to-analog converter of the channel. This value is in the units you specify with **ao_resolution_units**.

ao_resolution_units
nidaqmx.constants.ResolutionType – Specifies the units of **ao_resolution**.

ao_term_cfg
nidaqmx.constants.TerminalConfiguration – Specifies the terminal configuration of the channel.

ao_usb_xfer_req_count
int – Specifies the maximum number of simultaneous USB transfers used to stream data. Modify this value to affect performance under different combinations of operating system and device.

ao_usb_xfer_req_size
int – Specifies the maximum size of a USB transfer request in bytes. Modify this value to affect performance under different combinations of operating system and device.

ao_use_only_on_brd_mem
bool – Specifies whether to write samples directly to the onboard memory of the device, bypassing the memory buffer. Generally, you cannot update onboard memory directly after you start the task. Onboard memory includes data FIFOs.

ao_voltage_current_limit
float – Specifies the current limit, in amperes, for the voltage channel.

ao_voltage_units
nidaqmx.constants.VoltageUnits – Specifies in what units to generate voltage on the channel. Write data to the channel in the units you select.

chan_type
nidaqmx.constants.ChannelType – Indicates the type of the virtual channel.

channel_names
List[str] – Specifies the unflattened list of the virtual channels.

description
str – Specifies a user-defined description for the channel.

is_global
bool – Indicates whether the channel is a global channel.

name
str – Specifies the name of the virtual channel this object represents.

physical_channel
nidaqmx.system.physical_channel.PhysicalChannel – Specifies the name of the physical channel upon which this virtual channel is based.

save (*save_as=u''*, *author=u''*, *overwrite_existing_channel=False*, *allow_interactive_editing=True*, *allow_interactive_deletion=True*)
Saves this local or global channel to MAX as a global channel.

Parameters

- **save_as** (*Optional[str]*) – Is the name to save the task, global channel, or custom scale as. If you do not specify a value for this input, NI-DAQmx uses the name currently assigned to the task, global channel, or custom scale.
- **author** (*Optional[str]*) – Is a name to store with the task, global channel, or custom scale.
- **overwrite_existing_channel** (*Optional[bool]*) – Specifies whether to overwrite a global channel of the same name if one is already saved in MAX. If this input is False and a global channel of the same name is already saved in MAX, this function returns an error.

- **allow_interactive_editing** (*Optional[bool]*) – Specifies whether to allow the task, global channel, or custom scale to be edited in the DAQ Assistant. If allow_interactive_editing is True, the DAQ Assistant must support all task or global channel settings.
- **allow_interactive_deletion** (*Optional[bool]*) – Specifies whether to allow the task, global channel, or custom scale to be deleted through MAX.

nidaqmx.task.ci_channel

class nidaqmx._task_modules.channels.ci_channel.**CIChannel**(*task_handle*, *virtual_or_physical_name*)

Bases: *nidaqmx._task_modules.channels.channel.Channel*

Represents one or more counter input virtual channels and their properties.

chan_type
nidaqmx.constants.ChannelType – Indicates the type of the virtual channel.

channel_names
List[str] – Specifies the unflattened list of the virtual channels.

ci_ang_encoder_initial_angle
float – Specifies the starting angle of the encoder. This value is in the units you specify with **ci_ang_encoder_units**.

ci_ang_encoder_pulses_per_rev
int – Specifies the number of pulses the encoder generates per revolution. This value is the number of pulses on either signal A or signal B, not the total number of pulses on both signal A and signal B.

ci_ang_encoder_units
nidaqmx.constants.AngleUnits – Specifies the units to use to return angular position measurements from the channel.

ci_count
int – Indicates the current value of the count register.

ci_count_edges_active_edge
nidaqmx.constants.Edge – Specifies on which edges to increment or decrement the counter.

ci_count_edges_count_dir_dig_fltr_enable
bool – Specifies whether to apply the pulse width filter to the signal.

ci_count_edges_count_dir_dig_fltr_min_pulse_width
float – Specifies in seconds the minimum pulse width the filter recognizes.

ci_count_edges_count_dir_dig_fltr_timebase_rate
float – Specifies in hertz the rate of the pulse width filter timebase. NI-DAQmx uses this value to compute settings for the filter.

ci_count_edges_count_dir_dig_fltr_timebase_src
str – Specifies the input terminal of the signal to use as the timebase of the pulse width filter.

ci_count_edges_count_dir_dig_sync_enable
bool – Specifies whether to synchronize recognition of transitions in the signal to the internal timebase of the device.

ci_count_edges_count_dir_logic_lvl_behavior
nidaqmx.constants.LogicLvlBehavior – Specifies the logic level behavior on the count reset line.

ci_count_edges_count_dir_term_cfg
nidaqmx.constants.TerminalConfiguration – Specifies the input terminal configuration.

ci_count_edges_count_reset_active_edge
nidaqmx.constants.Edge – Specifies on which edge of the signal to reset the count.

ci_count_edges_count_reset_dig_fltr_enable
bool – Specifies whether to apply the pulse width filter to the signal.

ci_count_edges_count_reset_dig_fltr_min_pulse_width
float – Specifies the minimum pulse width the filter recognizes.

ci_count_edges_count_reset_dig_fltr_timebase_rate
float – Specifies in hertz the rate of the pulse width filter timebase. NI-DAQmx uses this value to compute settings for the filter.

ci_count_edges_count_reset_dig_fltr_timebase_src
str – Specifies the input of the signal to use as the timebase of the pulse width filter.

ci_count_edges_count_reset_dig_sync_enable
bool – Specifies whether to synchronize recognition of transitions in the signal to the internal timebase of the device.

ci_count_edges_count_reset_enable
bool – Specifies whether to reset the count on the active edge specified with **ci_count_edges_count_reset_term**.

ci_count_edges_count_reset_logic_lvl_behavior
nidaqmx.constants.LogicLvlBehavior – Specifies the logic level behavior on the count reset line.

ci_count_edges_count_reset_reset_cnt
int – Specifies the value to reset the count to.

ci_count_edges_count_reset_term
str – Specifies the input terminal of the signal to reset the count.

ci_count_edges_count_reset_term_cfg
nidaqmx.constants.TerminalConfiguration – Specifies the input terminal configuration.

ci_count_edges_dig_fltr_enable
bool – Specifies whether to apply the pulse width filter to the signal.

ci_count_edges_dig_fltr_min_pulse_width
float – Specifies in seconds the minimum pulse width the filter recognizes.

ci_count_edges_dig_fltr_timebase_rate
float – Specifies in hertz the rate of the pulse width filter timebase. NI-DAQmx uses this value to compute settings for the filter.

ci_count_edges_dig_fltr_timebase_src
str – Specifies the input terminal of the signal to use as the timebase of the pulse width filter.

ci_count_edges_dig_sync_enable
bool – Specifies whether to synchronize recognition of transitions in the signal to the internal timebase of the device.

ci_count_edges_dir
nidaqmx.constants.CountDirection – Specifies whether to increment or decrement the counter on each edge.

ci_count_edges_dir_term
str – Specifies the source terminal of the digital signal that controls the count direction if **ci_count_edges_dir** is **CountDirection1.EXTERNAL_SOURCE**.

ci_count_edges_gate_dig_fltr_enable
bool – Specifies whether to apply the pulse width filter to the gate input signal.

ci_count_edges_gate_dig_fltr_min_pulse_width
float – Specifies in seconds the minimum pulse width the digital filter recognizes.

ci_count_edges_gate_dig_fltr_timebase_rate
float – Specifies in hertz the rate of the pulse width filter timebase. NI-DAQmx uses this value to compute settings for the filter.

ci_count_edges_gate_dig_fltr_timebase_src
str – Specifies the input terminal of the signal to use as the timebase of the pulse width filter.

ci_count_edges_gate_enable
bool – Specifies whether to enable the functionality to gate the counter input signal for a count edges measurement.

ci_count_edges_gate_logic_lvl_behavior
nidaqmx.constants.LogicLvlBehavior – Specifies the logic level behavior on the gate input line.

ci_count_edges_gate_term
str – Specifies the gate terminal.

ci_count_edges_gate_term_cfg
nidaqmx.constants.TerminalConfiguration – Specifies the gate terminal configuration.

ci_count_edges_gate_when
nidaqmx.constants.Level – Specifies whether the counter gates input pulses while the signal is high or low.

ci_count_edges_initial_cnt
int – Specifies the starting value from which to count.

ci_count_edges_logic_lvl_behavior
nidaqmx.constants.LogicLvlBehavior – Specifies the logic level behavior on the input line.

ci_count_edges_term
str – Specifies the input terminal of the signal to measure.

ci_count_edges_term_cfg
nidaqmx.constants.TerminalConfiguration – Specifies the input terminal configuration.

ci_ctr_timebase_active_edge
nidaqmx.constants.Edge – Specifies whether a timebase cycle is from rising edge to rising edge or from falling edge to falling edge.

ci_ctr_timebase_dig_fltr_enable
bool – Specifies whether to apply the pulse width filter to the signal.

ci_ctr_timebase_dig_fltr_min_pulse_width
float – Specifies in seconds the minimum pulse width the filter recognizes.

ci_ctr_timebase_dig_fltr_timebase_rate
float – Specifies in hertz the rate of the pulse width filter timebase. NI-DAQmx uses this value to compute settings for the filter.

ci_ctr_timebase_dig_fltr_timebase_src
str – Specifies the input terminal of the signal to use as the timebase of the pulse width filter.

ci_ctr_timebase_dig_sync_enable
bool – Specifies whether to synchronize recognition of transitions in the signal to the internal timebase of the device.

ci_ctr_timebase_master_timebase_div
int – Specifies the divisor for an external counter timebase. You can divide the counter timebase in order to measure slower signals without causing the count register to roll over.

ci_ctr_timebase_rate
float – Specifies in Hertz the frequency of the counter timebase. Specifying the rate of a counter timebase allows you to take measurements in terms of time or frequency rather than in ticks of the timebase. If you use an external timebase and do not specify the rate, you can take measurements only in terms of ticks of the timebase.

ci_ctr_timebase_src
str – Specifies the terminal of the timebase to use for the counter.

ci_custom_scale
`nidaqmx.system.scale.Scale` – Specifies the name of a custom scale for the channel.

ci_data_xfer_mech
nidaqmx.constants.DataTransferActiveTransferMode – Specifies the data transfer mode for the channel.

ci_data_xfer_req_cond
nidaqmx.constants.InputDataTransferCondition – Specifies under what condition to transfer data from the onboard memory of the device to the buffer.

ci_dup_count_prevention
bool – Specifies whether to enable duplicate count prevention for the channel. Duplicate count prevention is enabled by default. Setting **ci_prescaler** disables duplicate count prevention unless you explicitly enable it.

ci_duty_cycle_dig_fltr_enable
bool – Specifies whether to apply the pulse width filter to the signal.

ci_duty_cycle_dig_fltr_min_pulse_width
float – Specifies in seconds the minimum pulse width the digital filter recognizes.

ci_duty_cycle_dig_fltr_timebase_rate
float – Specifies in hertz the rate of the pulse width filter timebase. NI-DAQmx uses this value to compute settings for the filter.

ci_duty_cycle_dig_fltr_timebase_src
str – Specifies the input terminal of the signal to use as the timebase of the pulse width filter.

ci_duty_cycle_logic_lvl_behavior
nidaqmx.constants.LogicLvlBehavior – Specifies the logic level behavior on the input line.

ci_duty_cycle_starting_edge
nidaqmx.constants.Edge – Specifies which edge of the input signal to begin the duty cycle measurement.

ci_duty_cycle_term
str – Specifies the input terminal of the signal to measure.

ci_duty_cycle_term_cfg
nidaqmx.constants.TerminalConfiguration – Specifies the input terminal configuration.

ci_encoder_a_input_dig_fltr_enable
bool – Specifies whether to apply the pulse width filter to the signal.

ci_encoder_a_input_dig_fltr_min_pulse_width
float – Specifies in seconds the minimum pulse width the filter recognizes.

ci_encoder_a_input_dig_fltr_timebase_rate
float – Specifies in hertz the rate of the pulse width filter timebase. NI-DAQmx uses this value to compute settings for the filter.

ci_encoder_a_input_dig_fltr_timebase_src
str – Specifies the input terminal of the signal to use as the timebase of the pulse width filter.

ci_encoder_a_input_dig_sync_enable
bool – Specifies whether to synchronize recognition of transitions in the signal to the internal timebase of the device.

ci_encoder_a_input_logic_lvl_behavior
nidaqmx.constants.LogicLvlBehavior – Specifies the logic level behavior on the input line.

ci_encoder_a_input_term
str – Specifies the terminal to which signal A is connected.

ci_encoder_a_input_term_cfg
nidaqmx.constants.TerminalConfiguration – Specifies the input terminal configuration.

ci_encoder_b_input_dig_fltr_enable
bool – Specifies whether to apply the pulse width filter to the signal.

ci_encoder_b_input_dig_fltr_min_pulse_width
float – Specifies in seconds the minimum pulse width the filter recognizes.

ci_encoder_b_input_dig_fltr_timebase_rate
float – Specifies in hertz the rate of the pulse width filter timebase. NI-DAQmx uses this value to compute settings for the filter.

ci_encoder_b_input_dig_fltr_timebase_src
str – Specifies the input terminal of the signal to use as the timebase of the pulse width filter.

ci_encoder_b_input_dig_sync_enable
bool – Specifies whether to synchronize recognition of transitions in the signal to the internal timebase of the device.

ci_encoder_b_input_logic_lvl_behavior
nidaqmx.constants.LogicLvlBehavior – Specifies the logic level behavior on the input line.

ci_encoder_b_input_term
str – Specifies the terminal to which signal B is connected.

ci_encoder_b_input_term_cfg
nidaqmx.constants.TerminalConfiguration – Specifies the input terminal configuration.

ci_encoder_decoding_type
nidaqmx.constants.EncoderType – Specifies how to count and interpret the pulses the encoder generates on signal A and signal B. **EncoderType2.X_1**, **EncoderType2.X_2**, and **EncoderType2.X_4** are valid for quadrature encoders only. **EncoderType2.TWO_PULSE_COUNTING** is valid for two-pulse encoders only.

ci_encoder_z_index_enable
bool – Specifies whether to use Z indexing for the channel.

ci_encoder_z_index_phase
nidaqmx.constants.EncoderZIndexPhase – Specifies the states at which signal A and signal B must be while signal Z is high for NI-DAQmx to reset the measurement. If signal Z is never high

while signal A and signal B are high, for example, you must choose a phase other than **EncoderZIndexPhase1.AHIGH_BHIGH**.

ci_encoder_z_index_val
float – Specifies the value to which to reset the measurement when signal Z is high and signal A and signal B are at the states you specify with **ci_encoder_z_index_phase**. Specify this value in the units of the measurement.

ci_encoder_z_input_dig_fltr_enable
bool – Specifies whether to apply the pulse width filter to the signal.

ci_encoder_z_input_dig_fltr_min_pulse_width
float – Specifies in seconds the minimum pulse width the filter recognizes.

ci_encoder_z_input_dig_fltr_timebase_rate
float – Specifies in hertz the rate of the pulse width filter timebase. NI-DAQmx uses this value to compute settings for the filter.

ci_encoder_z_input_dig_fltr_timebase_src
str – Specifies the input terminal of the signal to use as the timebase of the pulse width filter.

ci_encoder_z_input_dig_sync_enable
bool – Specifies whether to synchronize recognition of transitions in the signal to the internal timebase of the device.

ci_encoder_z_input_logic_lvl_behavior
nidaqmx.constants.LogicLvlBehavior – Specifies the logic level behavior on the input line.

ci_encoder_z_input_term
str – Specifies the terminal to which signal Z is connected.

ci_encoder_z_input_term_cfg
nidaqmx.constants.TerminalConfiguration – Specifies the input terminal configuration.

ci_freq_dig_fltr_enable
bool – Specifies whether to apply the pulse width filter to the signal.

ci_freq_dig_fltr_min_pulse_width
float – Specifies in seconds the minimum pulse width the filter recognizes.

ci_freq_dig_fltr_timebase_rate
float – Specifies in hertz the rate of the pulse width filter timebase. NI-DAQmx uses this value to compute settings for the filter.

ci_freq_dig_fltr_timebase_src
str – Specifies the input terminal of the signal to use as the timebase of the pulse width filter.

ci_freq_dig_sync_enable
bool – Specifies whether to synchronize recognition of transitions in the signal to the internal timebase of the device.

ci_freq_div
int – Specifies the value by which to divide the input signal if **ci_freq_meas_meth** is **CounterFrequencyMethod.LARGE_RANGE_2_COUNTERS**. The larger the divisor, the more accurate the measurement. However, too large a value could cause the count register to roll over, which results in an incorrect measurement.

ci_freq_enable_averaging
bool – Specifies whether to enable averaging mode for Sample Clock-timed frequency measurements.

ci_freq_logic_lvl_behavior
nidaqmx.constants.LogicLvlBehavior – Specifies the logic level behavior on the input line.

ci_freq_meas_meth
nidaqmx.constants.CounterFrequencyMethod – Specifies the method to use to measure the frequency of the signal.

ci_freq_meas_time
float – Specifies in seconds the length of time to measure the frequency of the signal if **ci_freq_meas_meth** is **CounterFrequencyMethod.HIGH_FREQUENCY_2_COUNTERS**. Measurement accuracy increases with increased measurement time and with increased signal frequency. If you measure a high-frequency signal for too long, however, the count register could roll over, which results in an incorrect measurement.

ci_freq_starting_edge
nidaqmx.constants.Edge – Specifies between which edges to measure the frequency of the signal.

ci_freq_term
str – Specifies the input terminal of the signal to measure.

ci_freq_term_cfg
nidaqmx.constants.TerminalConfiguration – Specifies the input terminal configuration.

ci_freq_units
nidaqmx.constants.FrequencyUnits – Specifies the units to use to return frequency measurements.

ci_gps_sync_method
nidaqmx.constants.GpsSignalType – Specifies the method to use to synchronize the counter to a GPS receiver.

ci_gps_sync_src
str – Specifies the terminal to which the GPS synchronization signal is connected.

ci_lin_encoder_dist_per_pulse
float – Specifies the distance to measure for each pulse the encoder generates on signal A or signal B. This value is in the units you specify with **ci_lin_encoder_units**.

ci_lin_encoder_initial_pos
float – Specifies the position of the encoder when the measurement begins. This value is in the units you specify with **ci_lin_encoder_units**.

ci_lin_encoder_units
nidaqmx.constants.LengthUnits – Specifies the units to use to return linear encoder measurements from the channel.

ci_max
float – Specifies the maximum value you expect to measure. This value is in the units you specify with a units property. When you query this property, it returns the coerced maximum value that the hardware can measure with the current settings.

ci_max_meas_period
float – Specifies the maximum period (in seconds) in which the device will recognize signals. For frequency measurements, a signal with a higher period than the one set in this property will return 0 Hz. For duty cycle, the device will return 0 or 1 depending on the state of the line during the max defined period of time. Period measurements will return NaN. Pulse width measurement will return zero.

ci_meas_type
nidaqmx.constants.UsageTypeCI – Indicates the measurement to take with the channel.

ci_mem_map_enable
bool – Specifies for NI-DAQmx to map hardware registers to the memory space of the application, if possible. Normally, NI- DAQmx maps hardware registers to memory accessible only to the kernel. Mapping the registers to the memory space of the application increases performance. However, if the application

accesses the memory space mapped to the registers, it can adversely affect the operation of the device and possibly result in a system crash.

ci_min
float – Specifies the minimum value you expect to measure. This value is in the units you specify with a units property. When you query this property, it returns the coerced minimum value that the hardware can measure with the current settings.

ci_num_possibly_invalid_samps
int – Indicates the number of samples that the device might have overwritten before it could transfer them to the buffer.

ci_output_state
`nidaqmx.constants.Level` – Indicates the current state of the out terminal of the counter.

ci_period_dig_fltr_enable
bool – Specifies whether to apply the pulse width filter to the signal.

ci_period_dig_fltr_min_pulse_width
float – Specifies in seconds the minimum pulse width the filter recognizes.

ci_period_dig_fltr_timebase_rate
float – Specifies in hertz the rate of the pulse width filter timebase. NI-DAQmx uses this value to compute settings for the filter.

ci_period_dig_fltr_timebase_src
str – Specifies the input terminal of the signal to use as the timebase of the pulse width filter.

ci_period_dig_sync_enable
bool – Specifies whether to synchronize recognition of transitions in the signal to the internal timebase of the device.

ci_period_div
int – Specifies the value by which to divide the input signal if **ci_period_meas_meth** is **CounterFrequencyMethod.LARGE_RANGE_2_COUNTERS**. The larger the divisor, the more accurate the measurement. However, too large a value could cause the count register to roll over, which results in an incorrect measurement.

ci_period_enable_averaging
bool – Specifies whether to enable averaging mode for Sample Clock-timed period measurements.

ci_period_logic_lvl_behavior
`nidaqmx.constants.LogicLvlBehavior` – Specifies the logic level behavior on the input line.

ci_period_meas_meth
`nidaqmx.constants.CounterFrequencyMethod` – Specifies the method to use to measure the period of the signal.

ci_period_meas_time
float – Specifies in seconds the length of time to measure the period of the signal if **ci_period_meas_meth** is **CounterFrequencyMethod.HIGH_FREQUENCY_2_COUNTERS**. Measurement accuracy increases with increased measurement time and with increased signal frequency. If you measure a high-frequency signal for too long, however, the count register could roll over, which results in an incorrect measurement.

ci_period_starting_edge
`nidaqmx.constants.Edge` – Specifies between which edges to measure the period of the signal.

ci_period_term
str – Specifies the input terminal of the signal to measure.

ci_period_term_cfg
nidaqmx.constants.TerminalConfiguration – Specifies the input terminal configuration.

ci_period_units
nidaqmx.constants.TimeUnits – Specifies the unit to use to return period measurements.

ci_prescaler
int – Specifies the divisor to apply to the signal you connect to the counter source terminal. Scaled data that you read takes this setting into account. You should use a prescaler only when you connect an external signal to the counter source terminal and when that signal has a higher frequency than the fastest onboard timebase. Setting this value disables duplicate count prevention unless you explicitly set **ci_dup_count_prevention** to True.

ci_pulse_freq_dig_fltr_enable
bool – Specifies whether to apply a digital filter to the signal to measure.

ci_pulse_freq_dig_fltr_min_pulse_width
float – Specifies in seconds the minimum pulse width the filter recognizes.

ci_pulse_freq_dig_fltr_timebase_rate
float – Specifies in hertz the rate of the digital filter timebase. NI-DAQmx uses this value to compute settings for the filter.

ci_pulse_freq_dig_fltr_timebase_src
str – Specifies the terminal of the signal to use as the timebase of the digital filter.

ci_pulse_freq_dig_sync_enable
bool – Specifies whether to synchronize recognition of transitions in the signal to the internal timebase of the device.

ci_pulse_freq_logic_lvl_behavior
nidaqmx.constants.LogicLvlBehavior – Specifies the logic level behavior on the count reset line.

ci_pulse_freq_starting_edge
nidaqmx.constants.Edge – Specifies on which edge of the input signal to begin pulse measurement.

ci_pulse_freq_term
str – Specifies the input terminal of the signal to measure.

ci_pulse_freq_term_cfg
nidaqmx.constants.TerminalConfiguration – Specifies the input terminal configuration.

ci_pulse_freq_units
nidaqmx.constants.FrequencyUnits – Specifies the units to use to return pulse specifications in terms of frequency.

ci_pulse_ticks_dig_fltr_enable
bool – Specifies whether to apply a digital filter to the signal to measure.

ci_pulse_ticks_dig_fltr_min_pulse_width
float – Specifies in seconds the minimum pulse width the filter recognizes.

ci_pulse_ticks_dig_fltr_timebase_rate
float – Specifies in hertz the rate of the digital filter timebase. NI-DAQmx uses this value to compute settings for the filter.

ci_pulse_ticks_dig_fltr_timebase_src
str – Specifies the terminal of the signal to use as the timebase of the digital filter.

ci_pulse_ticks_dig_sync_enable
bool – Specifies whether to synchronize recognition of transitions in the signal to the internal timebase of the device.

ci_pulse_ticks_logic_lvl_behavior
nidaqmx.constants.LogicLvlBehavior – Specifies the logic level behavior on the count reset line.

ci_pulse_ticks_starting_edge
nidaqmx.constants.Edge – Specifies on which edge of the input signal to begin pulse measurement.

ci_pulse_ticks_term
str – Specifies the input terminal of the signal to measure.

ci_pulse_ticks_term_cfg
nidaqmx.constants.TerminalConfiguration – Specifies the input terminal configuration.

ci_pulse_time_dig_fltr_enable
bool – Specifies whether to apply a digital filter to the signal to measure.

ci_pulse_time_dig_fltr_min_pulse_width
float – Specifies in seconds the minimum pulse width the filter recognizes.

ci_pulse_time_dig_fltr_timebase_rate
float – Specifies in hertz the rate of the digital filter timebase. NI-DAQmx uses this value to compute settings for the filter.

ci_pulse_time_dig_fltr_timebase_src
str – Specifies the terminal of the signal to use as the timebase of the digital filter.

ci_pulse_time_dig_sync_enable
bool – Specifies whether to synchronize recognition of transitions in the signal to the internal timebase of the device.

ci_pulse_time_logic_lvl_behavior
nidaqmx.constants.LogicLvlBehavior – Specifies the logic level behavior on the count reset line.

ci_pulse_time_starting_edge
nidaqmx.constants.Edge – Specifies on which edge of the input signal to begin pulse measurement.

ci_pulse_time_term
str – Specifies the input terminal of the signal to measure.

ci_pulse_time_term_cfg
nidaqmx.constants.TerminalConfiguration – Specifies the input terminal configuration.

ci_pulse_time_units
nidaqmx.constants.TimeUnits – Specifies the units to use to return pulse specifications in terms of high time and low time.

ci_pulse_width_dig_fltr_enable
bool – Specifies whether to apply the pulse width filter to the signal.

ci_pulse_width_dig_fltr_min_pulse_width
float – Specifies in seconds the minimum pulse width the filter recognizes.

ci_pulse_width_dig_fltr_timebase_rate
float – Specifies in hertz the rate of the pulse width filter timebase. NI-DAQmx uses this value to compute settings for the filter.

ci_pulse_width_dig_fltr_timebase_src
str – Specifies the input terminal of the signal to use as the timebase of the pulse width filter.

ci_pulse_width_dig_sync_enable
bool – Specifies whether to synchronize recognition of transitions in the signal to the internal timebase of the device.

ci_pulse_width_logic_lvl_behavior
nidaqmx.constants.LogicLvlBehavior – Specifies the logic level behavior on the input line.

ci_pulse_width_starting_edge
nidaqmx.constants.Edge – Specifies on which edge of the input signal to begin each pulse width measurement.

ci_pulse_width_term
str – Specifies the input terminal of the signal to measure.

ci_pulse_width_term_cfg
nidaqmx.constants.TerminalConfiguration – Specifies the input terminal configuration.

ci_pulse_width_units
nidaqmx.constants.TimeUnits – Specifies the units to use to return pulse width measurements.

ci_samp_clk_overrun_behavior
nidaqmx.constants.SampClkOverrunBehavior – Specifies the counter behavior when data is read but a new value was not detected during a sample clock.

ci_samp_clk_overrun_sentinel_val
int – Specifies the sentinel value returned when the No New Sample Behavior is set to Sentinel Value.

ci_semi_period_dig_fltr_enable
bool – Specifies whether to apply the pulse width filter to the signal.

ci_semi_period_dig_fltr_min_pulse_width
float – Specifies in seconds the minimum pulse width the filter recognizes.

ci_semi_period_dig_fltr_timebase_rate
float – Specifies in hertz the rate of the pulse width filter timebase. NI-DAQmx uses this value to compute settings for the filter.

ci_semi_period_dig_fltr_timebase_src
str – Specifies the input terminal of the signal to use as the timebase of the pulse width filter.

ci_semi_period_dig_sync_enable
bool – Specifies whether to synchronize recognition of transitions in the signal to the internal timebase of the device.

ci_semi_period_logic_lvl_behavior
nidaqmx.constants.LogicLvlBehavior – Specifies the logic level behavior on the count reset line.

ci_semi_period_starting_edge
nidaqmx.constants.Edge – Specifies on which edge of the input signal to begin semi-period measurement. Semi-period measurements alternate between high time and low time, starting on this edge.

ci_semi_period_term
str – Specifies the input terminal of the signal to measure.

ci_semi_period_term_cfg
nidaqmx.constants.TerminalConfiguration – Specifies the input terminal configuration.

ci_semi_period_units
nidaqmx.constants.TimeUnits – Specifies the units to use to return semi-period measurements.

ci_tc_reached
bool – Indicates whether the counter rolled over. When you query this property, NI-DAQmx resets it to False.

ci_thresh_voltage
float – Specifies the digital threshold value in Volts for high and low input transitions. Some devices do not support this for differential channels.

ci_timestamp_initial_seconds
int – Specifies the number of seconds that elapsed since the beginning of the current year. This value is ignored if **ci_gps_sync_method** is **GpsSignalType1.IRIGB**.

ci_timestamp_units
nidaqmx.constants.TimeUnits – Specifies the units to use to return timestamp measurements.

ci_two_edge_sep_first_dig_fltr_enable
bool – Specifies whether to apply the pulse width filter to the signal.

ci_two_edge_sep_first_dig_fltr_min_pulse_width
float – Specifies in seconds the minimum pulse width the filter recognizes.

ci_two_edge_sep_first_dig_fltr_timebase_rate
float – Specifies in hertz the rate of the pulse width filter timebase. NI-DAQmx uses this value to compute settings for the filter.

ci_two_edge_sep_first_dig_fltr_timebase_src
str – Specifies the input terminal of the signal to use as the timebase of the pulse width filter.

ci_two_edge_sep_first_dig_sync_enable
bool – Specifies whether to synchronize recognition of transitions in the signal to the internal timebase of the device.

ci_two_edge_sep_first_edge
nidaqmx.constants.Edge – Specifies on which edge of the first signal to start each measurement.

ci_two_edge_sep_first_logic_lvl_behavior
nidaqmx.constants.LogicLvlBehavior – Specifies the logic level behavior on the input line.

ci_two_edge_sep_first_term
str – Specifies the source terminal of the digital signal that starts each measurement.

ci_two_edge_sep_first_term_cfg
nidaqmx.constants.TerminalConfiguration – Specifies the input terminal configuration.

ci_two_edge_sep_second_dig_fltr_enable
bool – Specifies whether to apply the pulse width filter to the signal.

ci_two_edge_sep_second_dig_fltr_min_pulse_width
float – Specifies in seconds the minimum pulse width the filter recognizes.

ci_two_edge_sep_second_dig_fltr_timebase_rate
float – Specifies in hertz the rate of the pulse width filter timebase. NI-DAQmx uses this value to compute settings for the filter.

ci_two_edge_sep_second_dig_fltr_timebase_src
str – Specifies the input terminal of the signal to use as the timebase of the pulse width filter.

ci_two_edge_sep_second_dig_sync_enable
bool – Specifies whether to synchronize recognition of transitions in the signal to the internal timebase of the device.

ci_two_edge_sep_second_edge
nidaqmx.constants.Edge – Specifies on which edge of the second signal to stop each measurement.

ci_two_edge_sep_second_logic_lvl_behavior
nidaqmx.constants.LogicLvlBehavior – Specifies the logic level behavior on the count reset line.

ci_two_edge_sep_second_term
str – Specifies the source terminal of the digital signal that stops each measurement.

ci_two_edge_sep_second_term_cfg
nidaqmx.constants.TerminalConfiguration – Specifies the input terminal configuration.

ci_two_edge_sep_units
nidaqmx.constants.TimeUnits – Specifies the units to use to return two-edge separation measurements from the channel.

ci_usb_xfer_req_count
int – Specifies the maximum number of simultaneous USB transfers used to stream data. Modify this value to affect performance under different combinations of operating system and device.

ci_usb_xfer_req_size
int – Specifies the maximum size of a USB transfer request in bytes. Modify this value to affect performance under different combinations of operating system and device.

ci_velocity_a_input_dig_fltr_enable
bool – Specifies whether to apply the pulse width filter to the signal.

ci_velocity_a_input_dig_fltr_min_pulse_width
float – Specifies in seconds the minimum pulse width the digital filter recognizes.

ci_velocity_a_input_dig_fltr_timebase_rate
float – Specifies in hertz the rate of the pulse width filter timebase. NI-DAQmx uses this value to compute settings for the filter.

ci_velocity_a_input_dig_fltr_timebase_src
str – Specifies the input terminal of the signal to use as the timebase of the pulse width filter.

ci_velocity_a_input_logic_lvl_behavior
nidaqmx.constants.LogicLvlBehavior – Specifies the logic level behavior of the input terminal.

ci_velocity_a_input_term
str – Specifies the terminal to which signal A is connected.

ci_velocity_a_input_term_cfg
nidaqmx.constants.TerminalConfiguration – Specifies the input terminal configuration.

ci_velocity_ang_encoder_pulses_per_rev
int – Specifies the number of pulses the encoder generates per revolution. This value is the number of pulses on either signal A or signal B, not the total number of pulses on both signal A and signal B.

ci_velocity_ang_encoder_units
nidaqmx.constants.AngularVelocityUnits – Specifies the units to use to return angular velocity counter measurements.

ci_velocity_b_input_dig_fltr_enable
bool – Specifies whether to apply the pulse width filter to the signal.

ci_velocity_b_input_dig_fltr_min_pulse_width
float – Specifies in seconds the minimum pulse width the digital filter recognizes.

ci_velocity_b_input_dig_fltr_timebase_rate
float – Specifies in hertz the rate of the pulse width filter timebase. NI-DAQmx uses this value to compute settings for the filter.

ci_velocity_b_input_dig_fltr_timebase_src
str – Specifies the input terminal of the signal to use as the timebase of the pulse width filter.

ci_velocity_b_input_logic_lvl_behavior
nidaqmx.constants.LogicLvlBehavior – Specifies the logic level behavior of the input terminal.

ci_velocity_b_input_term
str – Specifies the terminal to which signal B is connected.

ci_velocity_b_input_term_cfg
nidaqmx.constants.TerminalConfiguration – Specifies the input terminal configuration.

ci_velocity_div
int – Specifies the value by which to divide the input signal.

ci_velocity_encoder_decoding_type
nidaqmx.constants.EncoderType – Specifies how to count and interpret the pulses the encoder generates on signal A and signal B. X1, X2, and X4 are valid for quadrature encoders only. Two Pulse Counting is valid for two-pulse encoders only.

ci_velocity_lin_encoder_dist_per_pulse
float – Specifies the distance to measure for each pulse the encoder generates on signal A or signal B. This value is in the units you specify in CI.Velocity.LinEncoder.DistUnits.

ci_velocity_lin_encoder_units
nidaqmx.constants.VelocityUnits – Specifies the units to use to return linear encoder velocity measurements from the channel.

ci_velocity_meas_time
float – Specifies in seconds the length of time to measure the velocity of the signal.

description
str – Specifies a user-defined description for the channel.

is_global
bool – Indicates whether the channel is a global channel.

name
str – Specifies the name of the virtual channel this object represents.

physical_channel
nidaqmx.system.physical_channel.PhysicalChannel – Specifies the name of the physical channel upon which this virtual channel is based.

save (*save_as=u''*, *author=u''*, *overwrite_existing_channel=False*, *allow_interactive_editing=True*, *allow_interactive_deletion=True*)
Saves this local or global channel to MAX as a global channel.

Parameters

- **save_as** (*Optional[str]*) – Is the name to save the task, global channel, or custom scale as. If you do not specify a value for this input, NI-DAQmx uses the name currently assigned to the task, global channel, or custom scale.
- **author** (*Optional[str]*) – Is a name to store with the task, global channel, or custom scale.
- **overwrite_existing_channel** (*Optional[bool]*) – Specifies whether to overwrite a global channel of the same name if one is already saved in MAX. If this input is False and a global channel of the same name is already saved in MAX, this function returns an error.

- **allow_interactive_editing** (*Optional[bool]*) – Specifies whether to allow the task, global channel, or custom scale to be edited in the DAQ Assistant. If allow_interactive_editing is True, the DAQ Assistant must support all task or global channel settings.
- **allow_interactive_deletion** (*Optional[bool]*) – Specifies whether to allow the task, global channel, or custom scale to be deleted through MAX.

nidaqmx.task.co_channel

class nidaqmx._task_modules.channels.co_channel.**COChannel**(*task_handle*, *virtual_or_physical_name*)

Bases: *nidaqmx._task_modules.channels.channel.Channel*

Represents one or more counter output virtual channels and their properties.

chan_type
nidaqmx.constants.ChannelType – Indicates the type of the virtual channel.

channel_names
List[str] – Specifies the unflattened list of the virtual channels.

co_auto_incr_cnt
int – Specifies a number of timebase ticks by which to increase the time spent in the idle state for each successive pulse.

co_constrained_gen_mode
nidaqmx.constants.ConstrainedGenMode – Specifies constraints to apply when the counter generates pulses. Constraining the counter reduces the device resources required for counter operation. Constraining the counter can also allow additional analog or counter tasks on the device to run concurrently. For continuous counter tasks, NI-DAQmx consumes no device resources when the counter is constrained. For finite counter tasks, resource use increases with the frequency regardless of the constraint mode. However, fixed frequency constraints significantly reduce resource usage, and fixed duty cycle constraint marginally reduces it.

co_count
int – Indicates the current value of the count register.

co_ctr_timebase_active_edge
nidaqmx.constants.Edge – Specifies whether a timebase cycle is from rising edge to rising edge or from falling edge to falling edge.

co_ctr_timebase_dig_fltr_enable
bool – Specifies whether to apply the pulse width filter to the signal.

co_ctr_timebase_dig_fltr_min_pulse_width
float – Specifies in seconds the minimum pulse width the filter recognizes.

co_ctr_timebase_dig_fltr_timebase_rate
float – Specifies in hertz the rate of the pulse width filter timebase. NI-DAQmx uses this value to compute settings for the filter.

co_ctr_timebase_dig_fltr_timebase_src
str – Specifies the input terminal of the signal to use as the timebase of the pulse width filter.

co_ctr_timebase_dig_sync_enable
bool – Specifies whether to synchronize recognition of transitions in the signal to the internal timebase of the device.

co_ctr_timebase_master_timebase_div
int – Specifies the divisor for an external counter timebase. You can divide the counter timebase in order to generate slower signals without causing the count register to roll over.

co_ctr_timebase_rate
float – Specifies in Hertz the frequency of the counter timebase. Specifying the rate of a counter timebase allows you to define output pulses in seconds rather than in ticks of the timebase. If you use an external timebase and do not specify the rate, you can define output pulses only in ticks of the timebase.

co_ctr_timebase_src
str – Specifies the terminal of the timebase to use for the counter. Typically, NI-DAQmx uses one of the internal counter timebases when generating pulses. Use this property to specify an external timebase and produce custom pulse widths that are not possible using the internal timebases.

co_data_xfer_mech
`nidaqmx.constants.DataTransferActiveTransferMode` – Specifies the data transfer mode for the device. For buffered operations, use DMA or USB Bulk. For non-buffered operations, use Polled.

co_data_xfer_req_cond
`nidaqmx.constants.OutputDataTransferCondition` – Specifies under what condition to transfer data from the buffer to the onboard memory of the device.

co_enable_initial_delay_on_retrigger
bool – Specifies whether to apply the initial delay to retriggered pulse trains.

co_mem_map_enable
bool – Specifies for NI-DAQmx to map hardware registers to the memory space of the application, if possible. Normally, NI- DAQmx maps hardware registers to memory accessible only to the kernel. Mapping the registers to the memory space of the application increases performance. However, if the application accesses the memory space mapped to the registers, it can adversely affect the operation of the device and possibly result in a system crash.

co_output_state
`nidaqmx.constants.Level` – Indicates the current state of the output terminal of the counter.

co_output_type
`nidaqmx.constants.UsageTypeCO` – Indicates how to define pulses generated on the channel.

co_prescaler
int – Specifies the divisor to apply to the signal you connect to the counter source terminal. Pulse generations defined by frequency or time take this setting into account, but pulse generations defined by ticks do not. You should use a prescaler only when you connect an external signal to the counter source terminal and when that signal has a higher frequency than the fastest onboard timebase.

co_pulse_done
bool – Indicates if the task completed pulse generation. Use this value for retriggerable pulse generation when you need to determine if the device generated the current pulse. For retriggerable tasks, when you query this property, NI-DAQmx resets it to False.

co_pulse_duty_cyc
float – Specifies the duty cycle of the pulses. The duty cycle of a signal is the width of the pulse divided by period. NI- DAQmx uses this ratio and the pulse frequency to determine the width of the pulses and the delay between pulses.

co_pulse_freq
float – Specifies the frequency of the pulses to generate. This value is in the units you specify with **co_pulse_freq_units** or when you create the channel.

co_pulse_freq_initial_delay
float – Specifies in seconds the amount of time to wait before generating the first pulse.

co_pulse_freq_units
nidaqmx.constants.FrequencyUnits – Specifies the units in which to define pulse frequency.

co_pulse_high_ticks
int – Specifies the number of ticks the pulse is high.

co_pulse_high_time
float – Specifies the amount of time that the pulse is at a high voltage. This value is in the units you specify with **co_pulse_time_units** or when you create the channel.

co_pulse_idle_state
nidaqmx.constants.Level – Specifies the resting state of the output terminal.

co_pulse_low_ticks
int – Specifies the number of ticks the pulse is low.

co_pulse_low_time
float – Specifies the amount of time that the pulse is at a low voltage. This value is in the units you specify with **co_pulse_time_units** or when you create the channel.

co_pulse_term
str – Specifies on which terminal to generate pulses.

co_pulse_ticks_initial_delay
int – Specifies the number of ticks to wait before generating the first pulse.

co_pulse_time_initial_delay
float – Specifies in seconds the amount of time to wait before generating the first pulse.

co_pulse_time_units
nidaqmx.constants.TimeUnits – Specifies the units in which to define high and low pulse time.

co_rdy_for_new_val
bool – Indicates whether the counter is ready for new continuous pulse train values.

co_usb_xfer_req_count
int – Specifies the maximum number of simultaneous USB transfers used to stream data. Modify this value to affect performance under different combinations of operating system and device.

co_usb_xfer_req_size
int – Specifies the maximum size of a USB transfer request in bytes. Modify this value to affect performance under different combinations of operating system and device.

co_use_only_on_brd_mem
bool – Specifies whether to write samples directly to the onboard memory of the device, bypassing the memory buffer. Generally, you cannot update onboard memory directly after you start the task. Onboard memory includes data FIFOs.

description
str – Specifies a user-defined description for the channel.

is_global
bool – Indicates whether the channel is a global channel.

name
str – Specifies the name of the virtual channel this object represents.

physical_channel
nidaqmx.system.physical_channel.PhysicalChannel – Specifies the name of the physical channel upon which this virtual channel is based.

save (*save_as=u''*, *author=u''*, *overwrite_existing_channel=False*, *allow_interactive_editing=True*, *allow_interactive_deletion=True*)

Saves this local or global channel to MAX as a global channel.

Parameters

- **save_as** (`Optional[str]`) – Is the name to save the task, global channel, or custom scale as. If you do not specify a value for this input, NI-DAQmx uses the name currently assigned to the task, global channel, or custom scale.
- **author** (`Optional[str]`) – Is a name to store with the task, global channel, or custom scale.
- **overwrite_existing_channel** (`Optional[bool]`) – Specifies whether to overwrite a global channel of the same name if one is already saved in MAX. If this input is False and a global channel of the same name is already saved in MAX, this function returns an error.
- **allow_interactive_editing** (`Optional[bool]`) – Specifies whether to allow the task, global channel, or custom scale to be edited in the DAQ Assistant. If allow_interactive_editing is True, the DAQ Assistant must support all task or global channel settings.
- **allow_interactive_deletion** (`Optional[bool]`) – Specifies whether to allow the task, global channel, or custom scale to be deleted through MAX.

nidaqmx.task.di_channel

class `nidaqmx._task_modules.channels.di_channel.`**DIChannel**(*task_handle*, *virtual_or_physical_name*)

Bases: `nidaqmx._task_modules.channels.channel.Channel`

Represents one or more digital input virtual channels and their properties.

chan_type

`nidaqmx.constants.ChannelType` – Indicates the type of the virtual channel.

channel_names

List[str] – Specifies the unflattened list of the virtual channels.

description

str – Specifies a user-defined description for the channel.

di_acquire_on

`nidaqmx.constants.ActiveOrInactiveEdgeSelection` – Specifies on which edge of the sample clock to acquire samples.

di_data_xfer_mech

`nidaqmx.constants.DataTransferActiveTransferMode` – Specifies the data transfer mode for the device.

di_data_xfer_req_cond

`nidaqmx.constants.InputDataTransferCondition` – Specifies under what condition to transfer data from the onboard memory of the device to the buffer.

di_dig_fltr_enable

bool – Specifies whether to enable the digital filter for the line(s) or port(s). You can enable the filter on a line-by- line basis. You do not have to enable the filter for all lines in a channel.

di_dig_fltr_enable_bus_mode
bool – Specifies whether to enable bus mode for digital filtering. If you set this property to True, NI-DAQmx treats all lines that use common filtering settings as a bus. If any line in the bus has jitter, all lines in the bus hold state until the entire bus stabilizes, or until 2 times the minimum pulse width elapses. If you set this property to False, NI-DAQmx filters all lines individually. Jitter in one line does not affect other lines.

di_dig_fltr_min_pulse_width
float – Specifies in seconds the minimum pulse width the filter recognizes as a valid high or low state transition.

di_dig_fltr_timebase_rate
float – Specifies in hertz the rate of the digital filter timebase. NI-DAQmx uses this value to compute settings for the filter.

di_dig_fltr_timebase_src
str – Specifies the terminal of the signal to use as the timebase of the digital filter.

di_dig_sync_enable
bool – Specifies whether to synchronize recognition of transitions in the signal to the internal timebase of the device.

di_invert_lines
bool – Specifies whether to invert the lines in the channel. If you set this property to True, the lines are at high logic when off and at low logic when on.

di_logic_family
`nidaqmx.constants.LogicFamily` – Specifies the logic family to use for acquisition. A logic family corresponds to voltage thresholds that are compatible with a group of voltage standards. Refer to the device documentation for information on the logic high and logic low voltages for these logic families.

di_mem_map_enable
bool – Specifies for NI-DAQmx to map hardware registers to the memory space of the application, if possible. Normally, NI- DAQmx maps hardware registers to memory accessible only to the kernel. Mapping the registers to the memory space of the application increases performance. However, if the application accesses the memory space mapped to the registers, it can adversely affect the operation of the device and possibly result in a system crash.

di_num_lines
int – Indicates the number of digital lines in the channel.

di_tristate
bool – Specifies whether to tristate the lines in the channel. If you set this property to True, NI-DAQmx tristates the lines in the channel. If you set this property to False, NI-DAQmx does not modify the configuration of the lines even if the lines were previously tristated. Set this property to False to read lines in other tasks or to read output-only lines.

di_usb_xfer_req_count
int – Specifies the maximum number of simultaneous USB transfers used to stream data. Modify this value to affect performance under different combinations of operating system and device.

di_usb_xfer_req_size
int – Specifies the maximum size of a USB transfer request in bytes. Modify this value to affect performance under different combinations of operating system and device.

is_global
bool – Indicates whether the channel is a global channel.

name
str – Specifies the name of the virtual channel this object represents.

physical_channel
: *nidaqmx.system.physical_channel.PhysicalChannel* – Specifies the name of the physical channel upon which this virtual channel is based.

save (*save_as=u''*, *author=u''*, *overwrite_existing_channel=False*, *allow_interactive_editing=True*, *allow_interactive_deletion=True*)
: Saves this local or global channel to MAX as a global channel.

Parameters

- **save_as** (`Optional[str]`) – Is the name to save the task, global channel, or custom scale as. If you do not specify a value for this input, NI-DAQmx uses the name currently assigned to the task, global channel, or custom scale.
- **author** (`Optional[str]`) – Is a name to store with the task, global channel, or custom scale.
- **overwrite_existing_channel** (`Optional[bool]`) – Specifies whether to overwrite a global channel of the same name if one is already saved in MAX. If this input is False and a global channel of the same name is already saved in MAX, this function returns an error.
- **allow_interactive_editing** (`Optional[bool]`) – Specifies whether to allow the task, global channel, or custom scale to be edited in the DAQ Assistant. If allow_interactive_editing is True, the DAQ Assistant must support all task or global channel settings.
- **allow_interactive_deletion** (`Optional[bool]`) – Specifies whether to allow the task, global channel, or custom scale to be deleted through MAX.

nidaqmx.task.do_channel

class `nidaqmx._task_modules.channels.do_channel.`**`DOChannel`** (*task_handle*, *virtual_or_physical_name*)
: Bases: *nidaqmx._task_modules.channels.channel.Channel*

Represents one or more digital output virtual channels and their properties.

chan_type
: *nidaqmx.constants.ChannelType* – Indicates the type of the virtual channel.

channel_names
: *List[str]* – Specifies the unflattened list of the virtual channels.

description
: *str* – Specifies a user-defined description for the channel.

do_data_xfer_mech
: *nidaqmx.constants.DataTransferActiveTransferMode* – Specifies the data transfer mode for the device.

do_data_xfer_req_cond
: *nidaqmx.constants.OutputDataTransferCondition* – Specifies under what condition to transfer data from the buffer to the onboard memory of the device.

do_generate_on
: *nidaqmx.constants.ActiveOrInactiveEdgeSelection* – Specifies on which edge of the sample clock to generate samples.

do_invert_lines
bool – Specifies whether to invert the lines in the channel. If you set this property to True, the lines are at high logic when off and at low logic when on.

do_line_states_done_state
nidaqmx.constants.Level – Specifies the state of the lines in a digital output task when the task completes execution.

do_line_states_paused_state
nidaqmx.constants.Level – Specifies the state of the lines in a digital output task when the task pauses.

do_line_states_start_state
nidaqmx.constants.Level – Specifies the state of the lines in a digital output task when the task starts.

do_logic_family
nidaqmx.constants.LogicFamily – Specifies the logic family to use for generation. A logic family corresponds to voltage thresholds that are compatible with a group of voltage standards. Refer to the device documentation for information on the logic high and logic low voltages for these logic families.

do_mem_map_enable
bool – Specifies for NI-DAQmx to map hardware registers to the memory space of the application, if possible. Normally, NI- DAQmx maps hardware registers to memory accessible only to the kernel. Mapping the registers to the memory space of the application increases performance. However, if the application accesses the memory space mapped to the registers, it can adversely affect the operation of the device and possibly result in a system crash.

do_num_lines
int – Indicates the number of digital lines in the channel.

do_output_drive_type
nidaqmx.constants.DigitalDriveType – Specifies the drive type for digital output channels.

do_overcurrent_auto_reenable
bool – Specifies whether to automatically reenable channels after they no longer exceed the current limit specified by **do_overcurrent_limit**.

do_overcurrent_limit
float – Specifies the current threshold in Amperes for the channel. A value of 0 means the channel observes no limit. Devices can monitor only a finite number of current thresholds simultaneously. If you attempt to monitor additional thresholds, NI-DAQmx returns an error.

do_overcurrent_reenable_period
float – Specifies the delay in seconds between the time a channel no longer exceeds the current limit and the reactivation of that channel, if **do_overcurrent_auto_reenable** is True.

do_tristate
bool – Specifies whether to stop driving the channel and set it to a high-impedance state. You must commit the task for this setting to take effect.

do_usb_xfer_req_count
int – Specifies the maximum number of simultaneous USB transfers used to stream data. Modify this value to affect performance under different combinations of operating system and device.

do_usb_xfer_req_size
int – Specifies the maximum size of a USB transfer request in bytes. Modify this value to affect performance under different combinations of operating system and device.

do_use_only_on_brd_mem
bool – Specifies whether to write samples directly to the onboard memory of the device, bypassing the

memory buffer. Generally, you cannot update onboard memory after you start the task. Onboard memory includes data FIFOs.

is_global
bool – Indicates whether the channel is a global channel.

name
str – Specifies the name of the virtual channel this object represents.

physical_channel
nidaqmx.system.physical_channel.PhysicalChannel – Specifies the name of the physical channel upon which this virtual channel is based.

save (*save_as=u''*, *author=u''*, *overwrite_existing_channel=False*, *allow_interactive_editing=True*, *allow_interactive_deletion=True*)
Saves this local or global channel to MAX as a global channel.

Parameters

- **save_as** (`Optional[str]`) – Is the name to save the task, global channel, or custom scale as. If you do not specify a value for this input, NI-DAQmx uses the name currently assigned to the task, global channel, or custom scale.
- **author** (`Optional[str]`) – Is a name to store with the task, global channel, or custom scale.
- **overwrite_existing_channel** (`Optional[bool]`) – Specifies whether to overwrite a global channel of the same name if one is already saved in MAX. If this input is False and a global channel of the same name is already saved in MAX, this function returns an error.
- **allow_interactive_editing** (`Optional[bool]`) – Specifies whether to allow the task, global channel, or custom scale to be edited in the DAQ Assistant. If allow_interactive_editing is True, the DAQ Assistant must support all task or global channel settings.
- **allow_interactive_deletion** (`Optional[bool]`) – Specifies whether to allow the task, global channel, or custom scale to be deleted through MAX.

nidaqmx.task.channel_collection

class nidaqmx._task_modules.channel_collection.**ChannelCollection**(*task_handle*)
Bases: `_abcoll.Sequence`

Contains the collection of channels for a DAQmx Task.

This class defines methods that implements a container object.

all
nidaqmx._task_modules.channels.channel.Channel – Specifies a channel object that represents the entire list of virtual channels on this channel collection.

channel_names
List[str] – Specifies the entire list of virtual channels on this channel collection.

nidaqmx.task.ai_channel_collection

class nidaqmx._task_modules.ai_channel_collection.**AIChannelCollection**(*task_handle*)
Bases: *nidaqmx._task_modules.channel_collection.ChannelCollection*

Contains the collection of analog input channels for a DAQmx Task.

add_ai_accel_4_wire_dc_voltage_chan (*physical_channel*, *name_to_assign_to_channel=u''*, *terminal_config=<TerminalConfiguration.DEFAULT: -1>*, *min_val=-5.0*, *max_val=5.0*, *units=<AccelUnits.G: 10186>*, *sensitivity=1000.0*, *sensitivity_units=<AccelSensitivityUnits.M_VOLTS_PER_G: 12509>*, *voltage_excit_source=<ExcitationSource.INTERNAL: 10200>*, *voltage_excit_val=0.0*, *use_excit_for_scaling=False*, *custom_scale_name=u''*)

Creates channel(s) to measure acceleration. Use this instance for custom sensors that require excitation. You can use the excitation to scale the measurement.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **terminal_config** (*Optional[nidaqmx.constants.TerminalConfiguration]*) – Specifies the input terminal configuration for the channel.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.AccelUnits]*) – Specifies the units to use to return acceleration measurements from the channel.
- **sensitivity** (*Optional[float]*) – Is the sensitivity of the sensor. This value is in the units you specify with the **sensitivity_units** input. Refer to the sensor documentation to determine this value.
- **sensitivity_units** (*Optional[nidaqmx.constants.AccelSensitivityUnits]*) – Specifies the units of the **sensitivity** input.
- **voltage_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies the source of excitation.
- **voltage_excit_val** (*Optional[float]*) – Specifies in volts the amount of excitation supplied to the sensor. Refer to the sensor documentation to determine appropriate excitation values.
- **use_excit_for_scaling** (*Optional[bool]*) – Specifies if NI- DAQmx divides the measurement by the excitation. You should typically set **use_excit_for_scaling** to True for ratiometric transducers. If you set **use_excit_for_scaling** to True, set **max_val** and **min_val** to reflect the scaling.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_ai_accel_chan (*physical_channel*, *name_to_assign_to_channel=u''*, *terminal_config=<TerminalConfiguration.DEFAULT: -1>*, *min_val=-5.0*, *max_val=5.0*, *units=<AccelUnits.G: 10186>*, *sensitivity=1000.0*, *sensitivity_units=<AccelSensitivityUnits.M_VOLTS_PER_G: 12509>*, *current_excit_source=<ExcitationSource.INTERNAL: 10200>*, *current_excit_val=0.004*, *custom_scale_name=u''*)

Creates channel(s) that use an accelerometer to measure acceleration.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **terminal_config** (*Optional[nidaqmx.constants.TerminalConfiguration]*) – Specifies the input terminal configuration for the channel.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.AccelUnits]*) – Specifies the units to use to return acceleration measurements from the channel.
- **sensitivity** (*Optional[float]*) – Is the sensitivity of the sensor. This value is in the units you specify with the **sensitivity_units** input. Refer to the sensor documentation to determine this value.
- **sensitivity_units** (*Optional[nidaqmx.constants.AccelSensitivityUnits]*) – Specifies the units of the **sensitivity** input.
- **current_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies the source of excitation.
- **current_excit_val** (*Optional[float]*) – Specifies in amperes the amount of excitation to supply to the sensor. Refer to the sensor documentation to determine this value.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_ai_accel_charge_chan (*physical_channel*, *name_to_assign_to_channel=u''*, *terminal_config=<TerminalConfiguration.DEFAULT: -1>*, *min_val=-5.0*, *max_val=5.0*, *units=<AccelUnits.G: 10186>*, *sensitivity=100.0*, *sensitivity_units=<AccelChargeSensitivityUnits.PICO_COULOMBS_PER_G: 16099>*, *custom_scale_name=u''*)

Creates channel(s) that use a charge-based sensor to measure acceleration.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **terminal_config** (*Optional[nidaqmx.constants.TerminalConfiguration]*) – Specifies the input terminal configuration for the channel.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.AccelUnits]*) – Specifies the units to use to return acceleration measurements from the channel.
- **sensitivity** (*Optional[float]*) – Is the sensitivity of the sensor. This value is in the units you specify with the **sensitivity_units** input. Refer to the sensor documentation to determine this value.
- **sensitivity_units** (*Optional[nidaqmx.constants.AccelChargeSensitivityUnits]*) – Specifies the units of the **sensitivity** input.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_ai_bridge_chan(*physical_channel, name_to_assign_to_channel=u'', min_val=-0.002, max_val=0.002, units=<BridgeUnits.VOLTS_PER_VOLTS: 15896>, bridge_config=<BridgeConfiguration.FULL_BRIDGE: 10182>, voltage_excit_source=<ExcitationSource.INTERNAL: 10200>, voltage_excit_val=2.5, nominal_bridge_resistance=350.0, custom_scale_name=u''*)

Creates channel(s) that measure voltage ratios from a Wheatstone bridge. Use this instance with bridge-based sensors that measure phenomena other than strain, force, pressure, or torque, or that scale data to physical units NI-DAQmx does not support.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.

- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.BridgeUnits]*) – Specifies in which unit to return voltage ratios from the channel.
- **bridge_config** (*Optional[nidaqmx.constants.BridgeConfiguration]*) – Specifies information about the bridge configuration and measurement.
- **voltage_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies information about the bridge configuration and measurement.
- **voltage_excit_val** (*Optional[float]*) – Specifies information about the bridge configuration and measurement.
- **nominal_bridge_resistance** (*Optional[float]*) – Specifies information about the bridge configuration and measurement.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_ai_charge_chan(*physical_channel, name_to_assign_to_channel=u'', terminal_config=<TerminalConfiguration.DEFAULT: -1>, min_val=-1e-09, max_val=1e-09, units=<ChargeUnits.COULOMBS: 16102>, custom_scale_name=u''*)

Creates channel(s) that use a sensor with charge output.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **terminal_config** (*Optional[nidaqmx.constants.TerminalConfiguration]*) – Specifies the input terminal configuration for the channel.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.ChargeUnits]*) – Specifies the units to use to return charge measurements from the channel.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_ai_current_chan (*physical_channel*, *name_to_assign_to_channel=u''*, *terminal_config=<TerminalConfiguration.DEFAULT: -1>*, *min_val=-0.01*, *max_val=0.01*, *units=<CurrentUnits.AMPS: 10342>*, *shunt_resistor_loc=<CurrentShuntResistorLocation.LET_DRIVER_CHOOSE: -1>*, *ext_shunt_resistor_val=249.0*, *custom_scale_name=u''*)

Creates channel(s) to measure current.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **terminal_config** (*Optional[nidaqmx.constants.TerminalConfiguration]*) – Specifies the input terminal configuration for the channel.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.CurrentUnits]*) – Specifies the units to use to return current measurements.
- **shunt_resistor_loc** (*Optional[nidaqmx.constants.CurrentShuntResistorLocation]*) – Specifies the location of the shunt resistor. For devices with built-in shunt resistors, specify the location as **INTERNAL**. For devices that do not have built-in shunt resistors, you must attach an external one, set this input to **EXTERNAL** and use the **ext_shunt_resistor_val** input to specify the value of the resistor.
- **ext_shunt_resistor_val** (*Optional[float]*) – Specifies in ohms the resistance of an external shunt resistor.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_ai_current_rms_chan (*physical_channel*, *name_to_assign_to_channel=u''*, *terminal_config=<TerminalConfiguration.DEFAULT: -1>*, *min_val=-0.01*, *max_val=0.01*, *units=<CurrentUnits.AMPS: 10342>*, *shunt_resistor_loc=<CurrentShuntResistorLocation.LET_DRIVER_CHOOSE: -1>*, *ext_shunt_resistor_val=249.0*, *custom_scale_name=u''*)

Creates a channel to measure current RMS, the average (mean) power of the acquired current.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.

- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **terminal_config** (*Optional[nidaqmx.constants.TerminalConfiguration]*) – Specifies the input terminal configuration for the channel.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.CurrentUnits]*) – Specifies the units to use to return current measurements.
- **shunt_resistor_loc** (*Optional[nidaqmx.constants.CurrentShuntResistorLocation]*) – Specifies the location of the shunt resistor. For devices with built-in shunt resistors, specify the location as **INTERNAL**. For devices that do not have built-in shunt resistors, you must attach an external one, set this input to **EXTERNAL** and use the **ext_shunt_resistor_val** input to specify the value of the resistor.
- **ext_shunt_resistor_val** (*Optional[float]*) – Specifies in ohms the resistance of an external shunt resistor.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_ai_force_bridge_polynomial_chan(*physical_channel, name_to_assign_to_channel=u'', min_val=-100.0, max_val=100.0, units=<ForceUnits.POUNDS: 15876>, bridge_config=<BridgeConfiguration.FULL_BRIDGE: 10182>, voltage_excit_source=<ExcitationSource.INTERNAL: 10200>, voltage_excit_val=2.5, nominal_bridge_resistance=350.0, forward_coeffs=None, reverse_coeffs=None, electrical_units=<BridgeElectricalUnits.M_VOLTS_PER_VOLT: 15897>, physical_units=<BridgePhysicalUnits.POUNDS: 15876>, custom_scale_name=u''*)

Creates channel(s) that use a Wheatstone bridge to measure force or load. Use this instance with sensors whose specifications provide a polynomial to convert electrical values to physical values. When you use this scaling type, NI-DAQmx requires coefficients for a polynomial that converts electrical values to physical values (forward), as well as coefficients for a polynomial that converts physical values to electrical values (reverse). If you only know one set of coefficients, use the DAQmx Compute Reverse Polynomial Coefficients function to generate the other set.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input,

NI-DAQmx uses the physical channel name as the virtual channel name.

- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.ForceUnits]*) – Specifies in which unit to return force measurements from the channel.
- **bridge_config** (*Optional[nidaqmx.constants.BridgeConfiguration]*) – Specifies information about the bridge configuration and measurement.
- **voltage_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies information about the bridge configuration and measurement.
- **voltage_excit_val** (*Optional[float]*) – Specifies information about the bridge configuration and measurement.
- **nominal_bridge_resistance** (*Optional[float]*) – Specifies information about the bridge configuration and measurement.
- **forward_coeffs** (*Optional[List[float]]*) – Specifies how to scale electrical values from the sensor to physical units.
- **reverse_coeffs** (*Optional[List[float]]*) – Specifies how to scale electrical values from the sensor to physical units.
- **electrical_units** (*Optional[nidaqmx.constants.BridgeElectricalUnits]*) – Specifies how to scale electrical values from the sensor to physical units.
- **physical_units** (*Optional[nidaqmx.constants.BridgePhysicalUnits]*) – Specifies how to scale electrical values from the sensor to physical units.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_ai_force_bridge_table_chan(*physical_channel*, *name_to_assign_to_channel=u''*, *min_val=-100.0*, *max_val=100.0*, *units=<ForceUnits.POUNDS: 15876>*, *bridge_config=<BridgeConfiguration.FULL_BRIDGE: 10182>*, *voltage_excit_source=<ExcitationSource.INTERNAL: 10200>*, *voltage_excit_val=2.5*, *nominal_bridge_resistance=350.0*, *electrical_vals=None*, *electrical_units=<BridgeElectricalUnits.M_VOLTS_PER_VOLT: 15897>*, *physical_vals=None*, *physical_units=<BridgePhysicalUnits.POUNDS: 15876>*, *custom_scale_name=u''*)

Creates channel(s) that use a Wheatstone bridge to measure force or load. Use this instance with sensors whose specifications provide a table of electrical values and the corresponding physical values. When you use this scaling type, NI-DAQmx performs linear scaling between each pair of electrical and physical values. The input limits specified with **min_val** and **max_val** must fall within the smallest and largest physical values. For any data outside those endpoints, NI-DAQmx coerces that data to the endpoints.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.ForceUnits]*) – Specifies in which unit to return force measurements from the channel.
- **bridge_config** (*Optional[nidaqmx.constants.BridgeConfiguration]*) – Specifies information about the bridge configuration and measurement.
- **voltage_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies information about the bridge configuration and measurement.
- **voltage_excit_val** (*Optional[float]*) – Specifies information about the bridge configuration and measurement.
- **nominal_bridge_resistance** (*Optional[float]*) – Specifies information about the bridge configuration and measurement.
- **electrical_vals** (*Optional[List[float]]*) – Specifies how to scale electrical values from the sensor to physical units.
- **electrical_units** (*Optional[nidaqmx.constants.BridgeElectricalUnits]*) – Specifies how to scale electrical values from the sensor to physical units.
- **physical_vals** (*Optional[List[float]]*) – Specifies how to scale electrical values from the sensor to physical units.
- **physical_units** (*Optional[nidaqmx.constants.BridgePhysicalUnits]*) – Specifies how to scale electrical values from the sensor to physical units.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_ai_force_bridge_two_point_lin_chan(*physical_channel, name_to_assign_to_channel=u'', min_val=-100.0, max_val=100.0, units=<ForceUnits.POUNDS: 15876>, bridge_config=<BridgeConfiguration.FULL_BRIDGE: 10182>, voltage_excit_source=<ExcitationSource.INTERNAL: 10200>, voltage_excit_val=2.5, nominal_bridge_resistance=350.0, first_electrical_val=0.0, second_electrical_val=2.0, electrical_units=<BridgeElectricalUnits.M_VOLTS_PER_VOLT: 15897>, first_physical_val=0.0, second_physical_val=100.0, physical_units=<BridgePhysicalUnits.POUNDS: 15876>, custom_scale_name=u''*)

Creates channel(s) that use a Wheatstone bridge to measure force or load. Use this instance with sensors whose specifications do not provide a polynomial for scaling or a table of electrical and physical values. When you use this scaling type, NI-DAQmx uses two points of electrical and physical values to calculate the slope and y-intercept of a linear equation and uses that equation to scale electrical values to physical values.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.ForceUnits]*) – Specifies in which unit to return force measurements from the channel.
- **bridge_config** (*Optional[nidaqmx.constants.BridgeConfiguration]*) – Specifies information about the bridge configuration and measurement.
- **voltage_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies information about the bridge configuration and measurement.
- **voltage_excit_val** (*Optional[float]*) – Specifies information about the bridge configuration and measurement.
- **nominal_bridge_resistance** (*Optional[float]*) – Specifies information about the bridge configuration and measurement.
- **first_electrical_val** (*Optional[float]*) – Specifies how to scale electrical values from the sensor to physical units.
- **second_electrical_val** (*Optional[float]*) – Specifies how to scale electrical values from the sensor to physical units.

- **electrical_units** (*Optional[nidaqmx.constants.BridgeElectricalUnits]*) – Specifies how to scale electrical values from the sensor to physical units.
- **first_physical_val** (*Optional[float]*) – Specifies how to scale electrical values from the sensor to physical units.
- **second_physical_val** (*Optional[float]*) – Specifies how to scale electrical values from the sensor to physical units.
- **physical_units** (*Optional[nidaqmx.constants.BridgePhysicalUnits]*) – Specifies how to scale electrical values from the sensor to physical units.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_ai_force_iepe_chan(*physical_channel*, *name_to_assign_to_channel=u''*, *terminal_config=<TerminalConfiguration.DEFAULT: -1>*, *min_val=-2000.0*, *max_val=2000.0*, *units=<ForceUnits.NEWTONS: 15875>*, *sensitivity=2.25*, *sensitivity_units=<ForceIEPESensorSensitivityUnits.M_VOLTS_PER_NEWTON: 15891>*, *current_excit_source=<ExcitationSource.INTERNAL: 10200>*, *current_excit_val=0.004*, *custom_scale_name=u''*)

Creates channel(s) that use an IEPE force sensor to measure force or load.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **terminal_config** (*Optional[nidaqmx.constants.TerminalConfiguration]*) – Specifies the input terminal configuration for the channel.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.ForceUnits]*) – Specifies in which unit to return force measurements from the channel.
- **sensitivity** (*Optional[float]*) – Is the sensitivity of the sensor. This value is in the units you specify with the **sensitivity_units** input. Refer to the sensor documentation to determine this value.
- **sensitivity_units** (*Optional[nidaqmx.constants.ForceIEPESensorSensitivityUnits]*) – Specifies the units of the **sensitivity** input.

- **current_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies the source of excitation.
- **current_excit_val** (*Optional[float]*) – Specifies in amperes the amount of excitation to supply to the sensor. Refer to the sensor documentation to determine this value.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_ai_freq_voltage_chan(*physical_channel*, *name_to_assign_to_channel=u''*, *min_val=1*, *max_val=100*, *units=<FrequencyUnits.HZ: 10373>*, *threshold_level=0.0*, *hysteresis=0.0*, *custom_scale_name=u''*)

Creates channel(s) that use a frequency-to-voltage converter to measure frequency.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.FrequencyUnits]*) – Specifies the units to use to return frequency measurements.
- **threshold_level** (*Optional[float]*) – Specifies in volts the level at which to recognize waveform repetitions. You should select a voltage level that occurs only once within the entire period of a waveform. You also can select a voltage that occurs only once while the voltage rises or falls.
- **hysteresis** (*Optional[float]*) – Specifies in volts a window below **level**. The input voltage must pass below **threshold_level** minus **hysteresis** before NI-DAQmx recognizes a waveform repetition. Hysteresis can improve measurement accuracy when the signal contains noise or jitter.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_ai_microphone_chan(*physical_channel*, *name_to_assign_to_channel=u''*, *terminal_config=<TerminalConfiguration.DEFAULT: -1>*, *units=<SoundPressureUnits.PA: 10081>*, *mic_sensitivity=10.0*, *max_snd_press_level=100.0*, *current_excit_source=<ExcitationSource.INTERNAL: 10200>*, *current_excit_val=0.004*, *custom_scale_name=u''*)

Creates channel(s) that use a microphone to measure sound pressure.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **terminal_config** (*Optional[nidaqmx.constants.TerminalConfiguration]*) – Specifies the input terminal configuration for the channel.
- **units** (*Optional[nidaqmx.constants.SoundPressureUnits]*) – Specifies the units to use to return sound pressure measurements.
- **mic_sensitivity** (*Optional[float]*) – Is the sensitivity of the microphone. Specify this value in mV/Pa.
- **max_snd_press_level** (*Optional[float]*) – Is the maximum instantaneous sound pressure level you expect to measure. This value is in decibels, referenced to 20 micropascals.
- **current_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies the source of excitation.
- **current_excit_val** (*Optional[float]*) – Specifies in amperes the amount of excitation to supply to the sensor. Refer to the sensor documentation to determine this value.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_ai_pos_eddy_curr_prox_probe_chan(*physical_channel, name_to_assign_to_channel=u'', min_val=0.0, max_val=0.00254, units=<LengthUnits.METERS: 10219>, sensitivity=200.0, sensitivity_units=<EddyCurrentProxProbeSensitivityUnits.MIL: 14836>, custom_scale_name=u''*)

Creates channel(s) that use an eddy current proximity probe to measure position.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.

- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.LengthUnits]*) – Specifies the units to use to return position measurements from the channel.
- **sensitivity** (*Optional[float]*) – Is the sensitivity of the sensor. This value is in the units you specify with the **sensitivity_units** input. Refer to the sensor documentation to determine this value.
- **sensitivity_units** (*Optional[nidaqmx.constants.EddyCurrentProxProbeSensitivityUnits]*) – Specifies the units of the **sensitivity** input.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_ai_pos_lvdt_chan(*physical_channel*, *name_to_assign_to_channel=u''*, *min_val=-0.1*, *max_val=0.1*, *units=<LengthUnits.METERS: 10219>*, *sensitivity=50.0*, *sensitivity_units=<LVDTSensitivityUnits.M_VOLTS_PER_VOLT_PER_MILLIMETER: 12506>*, *voltage_excit_source=<ExcitationSource.INTERNAL: 10200>*, *voltage_excit_val=1.0*, *voltage_excit_freq=2500.0*, *ac_excit_wire_mode=<ACExcitWireMode.FOUR_WIRE: 4>*, *custom_scale_name=u''*)

Creates channel(s) that use an LVDT to measure linear position.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.LengthUnits]*) – Specifies the units to use to return linear position measurements from the channel.
- **sensitivity** (*Optional[float]*) – Is the sensitivity of the sensor. This value is in the units you specify with the **sensitivity_units** input. Refer to the sensor documentation to determine this value.
- **sensitivity_units** (*Optional[nidaqmx.constants.LVDTSensitivityUnits]*) – Specifies the units of the **sensitivity** input.
- **voltage_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies the source of excitation.
- **voltage_excit_val** (*Optional[float]*) – Specifies in volts the amount of excitation supplied to the sensor. Refer to the sensor documentation to determine appropriate excitation values.

- **voltage_excit_freq** (*Optional[float]*) – Specifies in hertz the excitation frequency that the sensor requires. Refer to the sensor documentation to determine this value.
- **ac_excit_wire_mode** (*Optional[nidaqmx.constants.ACExcitWireMode]*) – Is the number of leads on the sensor. Some sensors require you to tie leads together to create a four- or five- wire sensor. Refer to the sensor documentation for more information.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_ai_pos_rvdt_chan (*physical_channel, name_to_assign_to_channel=u'', min_val=-70.0, max_val=70.0, units=<AngleUnits.DEGREES: 10146>, sensitivity=50.0, sensitivity_units=<RVDTSensitivityUnits.M_VPER_VPER_DEGREE: 12507>, voltage_excit_source=<ExcitationSource.INTERNAL: 10200>, voltage_excit_val=1.0, voltage_excit_freq=2500.0, ac_excit_wire_mode=<ACExcitWireMode.FOUR_WIRE: 4>, custom_scale_name=u''*)

Creates channel(s) that use an RVDT to measure angular position.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.AngleUnits]*) – Specifies the units to use to return angular position measurements from the channel.
- **sensitivity** (*Optional[float]*) – Is the sensitivity of the sensor. This value is in the units you specify with the **sensitivity_units** input. Refer to the sensor documentation to determine this value.
- **sensitivity_units** (*Optional[nidaqmx.constants.RVDTSensitivityUnits]*) – Specifies the units of the **sensitivity** input.
- **voltage_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies the source of excitation.
- **voltage_excit_val** (*Optional[float]*) – Specifies in volts the amount of excitation supplied to the sensor. Refer to the sensor documentation to determine appropriate excitation values.
- **voltage_excit_freq** (*Optional[float]*) – Specifies in hertz the excitation frequency that the sensor requires. Refer to the sensor documentation to determine this value.
- **ac_excit_wire_mode** (*Optional[nidaqmx.constants.ACExcitWireMode]*) – Is the number of leads on the sensor. Some sensors

require you to tie leads together to create a four- or five- wire sensor. Refer to the sensor documentation for more information.

- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_ai_pressure_bridge_polynomial_chan (*physical_channel, name_to_assign_to_channel=u'', min_val=-100.0, max_val=100.0, units=<PressureUnits.POUNDS_PER_SQ_INCH: 15879>, bridge_config=<BridgeConfiguration.FULL_BRIDGE: 10182>, voltage_excit_source=<ExcitationSource.INTERNAL: 10200>, voltage_excit_val=2.5, nominal_bridge_resistance=350.0, forward_coeffs=None, reverse_coeffs=None, electrical_units=<BridgeElectricalUnits.M_VOLTS_PER_VOLT: 15897>, physical_units=<BridgePhysicalUnits.POUNDS_PER_SQ_INCH: 15879>, custom_scale_name=u''*)

Creates channel(s) that use a Wheatstone bridge to measure pressure. Use this instance with sensors whose specifications provide a polynomial to convert electrical values to physical values. When you use this scaling type, NI-DAQmx requires coefficients for a polynomial that converts electrical values to physical values (forward), as well as coefficients for a polynomial that converts physical values to electrical values (reverse). If you only know one set of coefficients, use the DAQmx Compute Reverse Polynomial Coefficients function to generate the other set.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.PressureUnits]*) – Specifies in which unit to return pressure measurements from the channel.
- **bridge_config** (*Optional[nidaqmx.constants.BridgeConfiguration]*) – Specifies information about the bridge configuration and measurement.
- **voltage_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies information about the bridge configuration and measurement.

- **voltage_excit_val** (*Optional[float]*) – Specifies information about the bridge configuration and measurement.
- **nominal_bridge_resistance** (*Optional[float]*) – Specifies information about the bridge configuration and measurement.
- **forward_coeffs** (*Optional[List[float]]*) – Specifies how to scale electrical values from the sensor to physical units.
- **reverse_coeffs** (*Optional[List[float]]*) – Specifies how to scale electrical values from the sensor to physical units.
- **electrical_units** (*Optional[nidaqmx.constants.BridgeElectricalUnits]*) – Specifies how to scale electrical values from the sensor to physical units.
- **physical_units** (*Optional[nidaqmx.constants.BridgePhysicalUnits]*) – Specifies how to scale electrical values from the sensor to physical units.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_ai_pressure_bridge_table_chan(*physical_channel, name_to_assign_to_channel=u'', min_val=-100.0, max_val=100.0, units=<PressureUnits.POUNDS_PER_SQ_INCH: 15879>, bridge_config=<BridgeConfiguration.FULL_BRIDGE: 10182>, voltage_excit_source=<ExcitationSource.INTERNAL: 10200>, voltage_excit_val=2.5, nominal_bridge_resistance=350.0, electrical_vals=None, electrical_units=<BridgeElectricalUnits.M_VOLTS_PER_VOLT: 15897>, physical_vals=None, physical_units=<BridgePhysicalUnits.POUNDS_PER_SQ_INCH: 15879>, custom_scale_name=u''*)

Creates channel(s) that use a Wheatstone bridge to measure pressure. Use this instance with sensors whose specifications provide a table of electrical values and the corresponding physical values. When you use this scaling type, NI-DAQmx performs linear scaling between each pair of electrical and physical values. The input limits specified with **min_val** and **max_val** must fall within the smallest and largest physical values. For any data outside those endpoints, NI-DAQmx coerces that data to the endpoints.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.

- **units** (*Optional[nidaqmx.constants.PressureUnits]*) – Specifies in which unit to return pressure measurements from the channel.
- **bridge_config** (*Optional[nidaqmx.constants.BridgeConfiguration]*) – Specifies information about the bridge configuration and measurement.
- **voltage_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies information about the bridge configuration and measurement.
- **voltage_excit_val** (*Optional[float]*) – Specifies information about the bridge configuration and measurement.
- **nominal_bridge_resistance** (*Optional[float]*) – Specifies information about the bridge configuration and measurement.
- **electrical_vals** (*Optional[List[float]]*) – Specifies how to scale electrical values from the sensor to physical units.
- **electrical_units** (*Optional[nidaqmx.constants.BridgeElectricalUnits]*) – Specifies how to scale electrical values from the sensor to physical units.
- **physical_vals** (*Optional[List[float]]*) – Specifies how to scale electrical values from the sensor to physical units.
- **physical_units** (*Optional[nidaqmx.constants.BridgePhysicalUnits]*) – Specifies how to scale electrical values from the sensor to physical units.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_ai_pressure_bridge_two_point_lin_chan(*physical_channel, name_to_assign_to_channel=u'', min_val=-100.0, max_val=100.0, units=<PressureUnits.POUNDS_PER_SQ_INCH: 15879>, bridge_config=<BridgeConfiguration.FULL_BRIDGE: 10182>, voltage_excit_source=<ExcitationSource.INTERNAL: 10200>, voltage_excit_val=2.5, nominal_bridge_resistance=350.0, first_electrical_val=0.0, second_electrical_val=2.0, electrical_units=<BridgeElectricalUnits.M_VOLTS_PER_VOLT: 15897>, first_physical_val=0.0, second_physical_val=100.0, physical_units=<BridgePhysicalUnits.POUNDS_PER_SQ_INCH: 15879>, custom_scale_name=u''*)

Creates channel(s) that use a Wheatstone bridge to measure pressure. Use this instance with sensors whose specifications do not provide a polynomial for scaling or a table of electrical and physical values. When you use this scaling type, NI-DAQmx uses two points of electrical and physical values to calculate the slope and y-intercept of a linear equation and uses that equation to scale electrical values to physical values.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.PressureUnits]*) – Specifies in which unit to return pressure measurements from the channel.
- **bridge_config** (*Optional[nidaqmx.constants.BridgeConfiguration]*) – Specifies information about the bridge configuration and measurement.
- **voltage_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies information about the bridge configuration and measurement.
- **voltage_excit_val** (*Optional[float]*) – Specifies information about the bridge configuration and measurement.
- **nominal_bridge_resistance** (*Optional[float]*) – Specifies information about the bridge configuration and measurement.
- **first_electrical_val** (*Optional[float]*) – Specifies how to scale electrical values from the sensor to physical units.
- **second_electrical_val** (*Optional[float]*) – Specifies how to scale electrical values from the sensor to physical units.
- **electrical_units** (*Optional[nidaqmx.constants.BridgeElectricalUnits]*) – Specifies how to scale electrical values from the sensor to physical units.
- **first_physical_val** (*Optional[float]*) – Specifies how to scale electrical values from the sensor to physical units.
- **second_physical_val** (*Optional[float]*) – Specifies how to scale electrical values from the sensor to physical units.
- **physical_units** (*Optional[nidaqmx.constants.BridgePhysicalUnits]*) – Specifies how to scale electrical values from the sensor to physical units.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_ai_resistance_chan(*physical_channel*, *name_to_assign_to_channel=u''*, *min_val=100.0*, *max_val=1000.0*, *units=<ResistanceUnits.OHMS: 10384>*, *resistance_config=<ResistanceConfiguration.TWO_WIRE: 2>*, *current_excit_source=<ExcitationSource.EXTERNAL: 10167>*, *current_excit_val=0.001*, *custom_scale_name=u''*)

Creates channel(s) to measure resistance.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.ResistanceUnits]*) – Specifies the units to use to return resistance measurements.
- **resistance_config** (*Optional[nidaqmx.constants.ResistanceConfiguration]*) – Specifies the number of wires to use for resistive measurements.
- **current_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies the source of excitation.
- **current_excit_val** (*Optional[float]*) – Specifies in amperes the amount of excitation to supply to the sensor. Refer to the sensor documentation to determine this value.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_ai_rosette_strain_gage_chan(*physical_channel*, *rosette_type*, *gage_orientation*, *rosette_meas_types*, *name_to_assign_to_channel=u''*, *min_val=-0.001*, *max_val=0.001*, *strain_config=<StrainGageBridgeType.QUARTER_BRIDGE_I: 10271>*, *voltage_excit_source=<ExcitationSource.INTERNAL: 10200>*, *voltage_excit_val=2.5*, *gage_factor=2.0*, *nominal_gage_resistance=350.0*, *poisson_ratio=0.3*, *lead_wire_resistance=0.0*)

Creates channels to measure two-dimensional strain using a rosette strain gage.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create the strain gage virtual channels necessary to calculate the **rosette measurements** channels.

- **rosette_type** (nidaqmx.constants.StrainGageRosetteType) – Specifies information about the rosette configuration and measurements.
- **gage_orientation** (*float*) – Specifies information about the rosette configuration and measurements.
- **rosette_meas_types** (*List[int]*) – Specifies information about the rosette configuration and measurements.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx creates a default channel name.
- **min_val** (*Optional[float]*) – Specifies the minimum strain you expect to measure. This value applies to each strain gage in the rosette.
- **max_val** (*Optional[float]*) – Specifies the maximum strain you expect to measure. This value applies to each strain gage in the rosette.
- **strain_config** (*Optional[nidaqmx.constants.StrainGageBridgeType]*) – Specifies information about the bridge configuration and measurement.
- **voltage_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies information about the bridge configuration and measurement.
- **voltage_excit_val** (*Optional[float]*) – Specifies information about the bridge configuration and measurement.
- **gage_factor** (*Optional[float]*) – Contains information about the strain gage and measurement.
- **nominal_gage_resistance** (*Optional[float]*) – Contains information about the strain gage and measurement.
- **poisson_ratio** (*Optional[float]*) – Contains information about the strain gage and measurement.
- **lead_wire_resistance** (*Optional[float]*) – Specifies information about the bridge configuration and measurement.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_ai_rtd_chan(*physical_channel*, *name_to_assign_to_channel=u''*, *min_val=0.0*, *max_val=100.0*, *units=<TemperatureUnits.DEG_C: 10143>*, *rtd_type=<RTDType.PT_3750: 12481>*, *resistance_config=<ResistanceConfiguration.TWO_WIRE: 2>*, *current_excit_source=<ExcitationSource.EXTERNAL: 10167>*, *current_excit_val=0.0025*, *r_0=100.0*)

Creates channel(s) that use an RTD to measure temperature.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.

- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.TemperatureUnits]*) – Specifies the units to use to return temperature measurements.
- **rtd_type** (*Optional[nidaqmx.constants.RTDType]*) – Specifies the type of RTD connected to the channel.
- **resistance_config** (*Optional[nidaqmx.constants.ResistanceConfiguration]*) – Specifies the number of wires to use for resistive measurements.
- **current_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies the source of excitation.
- **current_excit_val** (*Optional[float]*) – Specifies in amperes the amount of excitation to supply to the sensor. Refer to the sensor documentation to determine this value.
- **r_0** (*Optional[float]*) – Is the sensor resistance in ohms at 0 degrees Celsius. The Callendar-Van Dusen equation requires this value. Refer to the sensor documentation to determine this value.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_ai_strain_gage_chan(*physical_channel*, *name_to_assign_to_channel=u''*, *min_val=-0.001*, *max_val=0.001*, *units=<StrainUnits.STRAIN: 10299>*, *strain_config=<StrainGageBridgeType.FULL_BRIDGE_I: 10183>*, *voltage_excit_source=<ExcitationSource.INTERNAL: 10200>*, *voltage_excit_val=2.5*, *gage_factor=2.0*, *initial_bridge_voltage=0.0*, *nominal_gage_resistance=350.0*, *poisson_ratio=0.3*, *lead_wire_resistance=0.0*, *custom_scale_name=u''*)

Creates channel(s) to measure strain.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.StrainUnits]*) – Specifies the units to use to return strain measurements.
- **strain_config** (*Optional[nidaqmx.constants.StrainGageBridgeType]*) – Specifies information about the bridge configuration and measurement.

- **voltage_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies information about the bridge configuration and measurement.
- **voltage_excit_val** (*Optional[float]*) – Specifies information about the bridge configuration and measurement.
- **gage_factor** (*Optional[float]*) – Contains information about the strain gage and measurement.
- **initial_bridge_voltage** (*Optional[float]*) – Specifies information about the bridge configuration and measurement.
- **nominal_gage_resistance** (*Optional[float]*) – Contains information about the strain gage and measurement.
- **poisson_ratio** (*Optional[float]*) – Contains information about the strain gage and measurement.
- **lead_wire_resistance** (*Optional[float]*) – Specifies information about the bridge configuration and measurement.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_ai_temp_built_in_sensor_chan(*physical_channel*, *name_to_assign_to_channel=u''*, *units=<TemperatureUnits.DEG_C: 10143>*)

Creates channel(s) that use the built-in sensor of a terminal block or device to measure temperature. On SCXI modules, for example, the built-in sensor could be the CJC sensor.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **units** (*Optional[nidaqmx.constants.TemperatureUnits]*) – Specifies the units to use to return temperature measurements.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_ai_thrmcpl_chan(*physical_channel*, *name_to_assign_to_channel=u''*, *min_val=0.0*, *max_val=100.0*, *units=<TemperatureUnits.DEG_C: 10143>*, *thermocouple_type=<ThermocoupleType.J: 10072>*, *cjc_source=<CJCSource.CONSTANT_USER_VALUE: 10116>*, *cjc_val=25.0*, *cjc_channel=u''*)

Creates channel(s) that use a thermocouple to measure temperature.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.

- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.TemperatureUnits]*) – Specifies the units to use to return temperature measurements.
- **thermocouple_type** (*Optional[nidaqmx.constants.ThermocoupleType]*) – Specifies the type of thermocouple connected to the channel. Thermocouple types differ in composition and measurement range.
- **cjc_source** (*Optional[nidaqmx.constants.CJCSource]*) – Specifies the source of cold-junction compensation.
- **cjc_val** (*Optional[float]*) – Specifies in **units** the temperature of the cold junction if you set **cjc_source** to **CONSTANT_VALUE**.
- **cjc_channel** (*Optional[str]*) – Specifies the channel that acquires the temperature of the thermocouple cold- junction if you set **cjc_source** to **CHANNEL**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_ai_thrmstr_chan_iex(*physical_channel, name_to_assign_to_channel=u'', min_val=0.0, max_val=100.0, units=<TemperatureUnits.DEG_C: 10143>, resistance_config=<ResistanceConfiguration.FOUR_WIRE: 4>, current_excit_source=<ExcitationSource.EXTERNAL: 10167>, current_excit_val=0.00015, a=0.001295361, b=0.0002343159, c=1.018703e-07*)

Creates channel(s) that use a thermistor to measure temperature. Use this instance when the thermistor requires current excitation.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.TemperatureUnits]*) – Specifies the units to use to return temperature measurements.
- **resistance_config** (*Optional[nidaqmx.constants.ResistanceConfiguration]*) – Specifies the number of wires to use for resistive measurements.

- **current_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies the source of excitation.
- **current_excit_val** (*Optional[float]*) – Specifies in amperes the amount of excitation to supply to the sensor. Refer to the sensor documentation to determine this value.
- **a** (*Optional[float]*) – Contains the constants for the Steinhart-Hart thermistor equation. Refer to the sensor documentation to determine values for these constants.
- **b** (*Optional[float]*) – Contains the constants for the Steinhart-Hart thermistor equation. Refer to the sensor documentation to determine values for these constants.
- **c** (*Optional[float]*) – Contains the constants for the Steinhart-Hart thermistor equation. Refer to the sensor documentation to determine values for these constants.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_ai_thrmstr_chan_vex(*physical_channel*, *name_to_assign_to_channel=u''*, *min_val=0.0*, *max_val=100.0*, *units=<TemperatureUnits.DEG_C: 10143>*, *resistance_config=<ResistanceConfiguration.FOUR_WIRE: 4>*, *voltage_excit_source=<ExcitationSource.EXTERNAL: 10167>*, *voltage_excit_val=2.5*, *a=0.001295361*, *b=0.0002343159*, *c=1.018703e-07*, *r_1=5000.0*)

Creates channel(s) that use a thermistor to measure temperature. Use this instance when the thermistor requires voltage excitation.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.TemperatureUnits]*) – Specifies the units to use to return temperature measurements.
- **resistance_config** (*Optional[nidaqmx.constants.ResistanceConfiguration]*) – Specifies the number of wires to use for resistive measurements.
- **voltage_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies the source of excitation.
- **voltage_excit_val** (*Optional[float]*) – Specifies in volts the amount of excitation supplied to the sensor. Refer to the sensor documentation to determine appropriate excitation values.
- **a** (*Optional[float]*) – Contains the constants for the Steinhart-Hart thermistor equation. Refer to the sensor documentation to determine values for these constants.

- **b** (*Optional[float]*) – Contains the constants for the Steinhart-Hart thermistor equation. Refer to the sensor documentation to determine values for these constants.
- **c** (*Optional[float]*) – Contains the constants for the Steinhart-Hart thermistor equation. Refer to the sensor documentation to determine values for these constants.
- **r_1** (*Optional[float]*) – Specifies in ohms the value of the reference resistor.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_ai_torque_bridge_polynomial_chan(*physical_channel*, *name_to_assign_to_channel=u''*, *min_val=-100.0*, *max_val=100.0*, *units=<TorqueUnits.INCH_POUNDS: 15883>*, *bridge_config=<BridgeConfiguration.FULL_BRIDGE: 10182>*, *voltage_excit_source=<ExcitationSource.INTERNAL: 10200>*, *voltage_excit_val=2.5*, *nominal_bridge_resistance=350.0*, *forward_coeffs=None*, *reverse_coeffs=None*, *electrical_units=<BridgeElectricalUnits.M_VOLTS_PER_VOLT: 15897>*, *physical_units=<BridgePhysicalUnits.INCH_POUNDS: 15883>*, *custom_scale_name=u''*)

Creates channel(s) that use a Wheatstone bridge to measure torque. Use this instance with sensors whose specifications provide a polynomial to convert electrical values to physical values. When you use this scaling type, NI-DAQmx requires coefficients for a polynomial that converts electrical values to physical values (forward), as well as coefficients for a polynomial that converts physical values to electrical values (reverse). If you only know one set of coefficients, use the DAQmx Compute Reverse Polynomial Coefficients function to generate the other set.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.TorqueUnits]*) – Specifies in which unit to return torque measurements from the channel.
- **bridge_config** (*Optional[nidaqmx.constants.BridgeConfiguration]*) – Specifies information about the bridge configuration and measurement.
- **voltage_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies information about the bridge configuration and measurement.
- **voltage_excit_val** (*Optional[float]*) – Specifies information about the bridge configuration and measurement.

- **nominal_bridge_resistance** (*Optional[float]*) – Specifies information about the bridge configuration and measurement.
- **forward_coeffs** (*Optional[List[float]]*) – Specifies how to scale electrical values from the sensor to physical units.
- **reverse_coeffs** (*Optional[List[float]]*) – Specifies how to scale electrical values from the sensor to physical units.
- **electrical_units** (*Optional[nidaqmx.constants.BridgeElectricalUnits]*) – Specifies how to scale electrical values from the sensor to physical units.
- **physical_units** (*Optional[nidaqmx.constants.BridgePhysicalUnits]*) – Specifies how to scale electrical values from the sensor to physical units.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_ai_torque_bridge_table_chan(*physical_channel*, *name_to_assign_to_channel=u''*, *min_val=-100.0*, *max_val=100.0*, *units=<TorqueUnits.INCH_POUNDS: 15883>*, *bridge_config=<BridgeConfiguration.FULL_BRIDGE: 10182>*, *voltage_excit_source=<ExcitationSource.INTERNAL: 10200>*, *voltage_excit_val=2.5*, *nominal_bridge_resistance=350.0*, *electrical_vals=None*, *electrical_units=<BridgeElectricalUnits.M_VOLTS_PER_VOLT: 15897>*, *physical_vals=None*, *physical_units=<BridgePhysicalUnits.INCH_POUNDS: 15883>*, *custom_scale_name=u''*)

Creates channel(s) that use a Wheatstone bridge to measure torque. Use this instance with sensors whose specifications provide a table of electrical values and the corresponding physical values. When you use this scaling type, NI-DAQmx performs linear scaling between each pair of electrical and physical values. The input limits specified with **min_val** and **max_val** must fall within the smallest and largest physical values. For any data outside those endpoints, NI-DAQmx coerces that data to the endpoints.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.TorqueUnits]*) – Specifies in which unit to return torque measurements from the channel.

- **bridge_config** (*Optional[*nidaqmx.constants.BridgeConfiguration*]*) – Specifies information about the bridge configuration and measurement.
- **voltage_excit_source** (*Optional[*nidaqmx.constants.ExcitationSource*]*) – Specifies information about the bridge configuration and measurement.
- **voltage_excit_val** (*Optional[float]*) – Specifies information about the bridge configuration and measurement.
- **nominal_bridge_resistance** (*Optional[float]*) – Specifies information about the bridge configuration and measurement.
- **electrical_vals** (*Optional[List[float]]*) – Specifies how to scale electrical values from the sensor to physical units.
- **electrical_units** (*Optional[*nidaqmx.constants.BridgeElectricalUnits*]*) – Specifies how to scale electrical values from the sensor to physical units.
- **physical_vals** (*Optional[List[float]]*) – Specifies how to scale electrical values from the sensor to physical units.
- **physical_units** (*Optional[*nidaqmx.constants.BridgePhysicalUnits*]*) – Specifies how to scale electrical values from the sensor to physical units.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_ai_torque_bridge_two_point_lin_chan(*physical_channel, name_to_assign_to_channel=u'', min_val=-100.0, max_val=100.0, units=<TorqueUnits.INCH_POUNDS: 15883>, bridge_config=<BridgeConfiguration.FULL_BRIDGE: 10182>, voltage_excit_source=<ExcitationSource.INTERNAL: 10200>, voltage_excit_val=2.5, nominal_bridge_resistance=350.0, first_electrical_val=0.0, second_electrical_val=2.0, electrical_units=<BridgeElectricalUnits.M_VOLTS_PER_VOLT: 15897>, first_physical_val=0.0, second_physical_val=100.0, physical_units=<BridgePhysicalUnits.INCH_POUNDS: 15883>, custom_scale_name=u''*)

Creates channel(s) that use a Wheatstone bridge to measure torque. Use this instance with sensors whose specifications do not provide a polynomial for scaling or a table of electrical and physical values. When you use this scaling type, NI-DAQmx uses two points of electrical and physical values to calculate the slope and y-intercept of a linear equation and uses that equation to scale electrical values to physical values.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.

- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.TorqueUnits]*) – Specifies in which unit to return torque measurements from the channel.
- **bridge_config** (*Optional[nidaqmx.constants.BridgeConfiguration]*) – Specifies information about the bridge configuration and measurement.
- **voltage_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies information about the bridge configuration and measurement.
- **voltage_excit_val** (*Optional[float]*) – Specifies information about the bridge configuration and measurement.
- **nominal_bridge_resistance** (*Optional[float]*) – Specifies information about the bridge configuration and measurement.
- **first_electrical_val** (*Optional[float]*) – Specifies how to scale electrical values from the sensor to physical units.
- **second_electrical_val** (*Optional[float]*) – Specifies how to scale electrical values from the sensor to physical units.
- **electrical_units** (*Optional[nidaqmx.constants.BridgeElectricalUnits]*) – Specifies how to scale electrical values from the sensor to physical units.
- **first_physical_val** (*Optional[float]*) – Specifies how to scale electrical values from the sensor to physical units.
- **second_physical_val** (*Optional[float]*) – Specifies how to scale electrical values from the sensor to physical units.
- **physical_units** (*Optional[nidaqmx.constants.BridgePhysicalUnits]*) – Specifies how to scale electrical values from the sensor to physical units.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_ai_velocity_iepe_chan (*physical_channel*, *name_to_assign_to_channel=u''*, *terminal_config=<TerminalConfiguration.DEFAULT: -1>*, *min_val=-50.0*, *max_val=50.0*, *units=<VelocityUnits.INCHES_PER_SECOND: 15960>*, *sensitivity=100.0*, *sensitivity_units=<VelocityIEPESensorSensitivityUnits.M_VOLTS_PER_INCH_PER_SECOND: 15964>*, *current_excit_source=<ExcitationSource.INTERNAL: 10200>*, *current_excit_val=0.002*, *custom_scale_name=u''*)

Creates channel(s) that use an IEPE velocity sensor to measure velocity.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **terminal_config** (*Optional[nidaqmx.constants.TerminalConfiguration]*) – Specifies the input terminal configuration for the channel.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.VelocityUnits]*) – Specifies in which unit to return velocity measurements from the channel.
- **sensitivity** (*Optional[float]*) – Is the sensitivity of the sensor. This value is in the units you specify with the **sensitivity_units** input. Refer to the sensor documentation to determine this value.
- **sensitivity_units** (*Optional[nidaqmx.constants.VelocityIEPESensorSensitivityUnits]*) – Specifies the units of the **sensitivity** input.
- **current_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies the source of excitation.
- **current_excit_val** (*Optional[float]*) – Specifies in amperes the amount of excitation to supply to the sensor. Refer to the sensor documentation to determine this value.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_ai_voltage_chan (*physical_channel*, *name_to_assign_to_channel=u''*, *terminal_config=<TerminalConfiguration.DEFAULT: -1>*, *min_val=-5.0*, *max_val=5.0*, *units=<VoltageUnits.VOLTS: 10348>*, *custom_scale_name=u''*)

Creates channel(s) to measure voltage. If the measurement requires the use of internal excitation or you need excitation to scale the voltage, use the AI Custom Voltage with Excitation instance of this function.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **terminal_config** (*Optional[nidaqmx.constants.TerminalConfiguration]*) – Specifies the input terminal configuration for the channel.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.VoltageUnits]*) – Specifies the units to use to return voltage measurements.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_ai_voltage_chan_with_excit(*physical_channel*, *name_to_assign_to_channel=u''*, *terminal_config=<TerminalConfiguration.DEFAULT: -1>*, *min_val=-10.0*, *max_val=10.0*, *units=<VoltageUnits.VOLTS: 10348>*, *bridge_config=<BridgeConfiguration.NO_BRIDGE: 10228>*, *voltage_excit_source=<ExcitationSource.INTERNAL: 10200>*, *voltage_excit_val=0.0*, *use_excit_for_scaling=False*, *custom_scale_name=u''*)

Creates channel(s) to measure voltage. Use this instance for custom sensors that require excitation. You can use the excitation to scale the measurement.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **terminal_config** (*Optional[nidaqmx.constants.TerminalConfiguration]*) – Specifies the input terminal configuration for the channel.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.

- **units** (*Optional[nidaqmx.constants.VoltageUnits]*) – Specifies the units to use to return voltage measurements.
- **bridge_config** (*Optional[nidaqmx.constants.BridgeConfiguration]*) – Specifies what type of Wheatstone bridge the sensor is.
- **voltage_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies the source of excitation.
- **voltage_excit_val** (*Optional[float]*) – Specifies in volts the amount of excitation supplied to the sensor. Refer to the sensor documentation to determine appropriate excitation values.
- **use_excit_for_scaling** (*Optional[bool]*) – Specifies if NI- DAQmx divides the measurement by the excitation. You should typically set **use_excit_for_scaling** to True for ratiometric transducers. If you set **use_excit_for_scaling** to True, set **max_val** and **min_val** to reflect the scaling.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_ai_voltage_rms_chan (*physical_channel, name_to_assign_to_channel=u'', terminal_config=<TerminalConfiguration.DEFAULT: -1>, min_val=-5.0, max_val=5.0, units=<VoltageUnits.VOLTS: 10348>, custom_scale_name=u''*)

Creates channel(s) to measure voltage RMS, the average (mean) power of the acquired voltage.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **terminal_config** (*Optional[nidaqmx.constants.TerminalConfiguration]*) – Specifies the input terminal configuration for the channel.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.VoltageUnits]*) – Specifies the units to use to return voltage measurements.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_teds_ai_accel_chan(*physical_channel*, *name_to_assign_to_channel=u''*, *terminal_config=<TerminalConfiguration.DEFAULT: -1>*, *min_val=-5.0*, *max_val=5.0*, *units=<AccelUnits.G: 10186>*, *current_excit_source=<ExcitationSource.INTERNAL: 10200>*, *current_excit_val=0.004*, *custom_scale_name=u''*)

Creates channel(s) that use an accelerometer to measure acceleration. You must configure the physical channel(s) with TEDS information to use this function.

Parameters

- **physical_channel** (`str`) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (`Optional[str]`) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **terminal_config** (`Optional[nidaqmx.constants.TerminalConfiguration]`) – Specifies the input terminal configuration for the channel.
- **min_val** (`Optional[float]`) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (`Optional[float]`) – Specifies in **units** the maximum value you expect to measure.
- **units** (`Optional[nidaqmx.constants.AccelUnits]`) – Specifies the units to use to return acceleration measurements from the channel.
- **current_excit_source** (`Optional[nidaqmx.constants.ExcitationSource]`) – Specifies the source of excitation.
- **current_excit_val** (`Optional[float]`) – Specifies in amperes the amount of excitation to supply to the sensor. Refer to the sensor documentation to determine this value.
- **custom_scale_name** (`Optional[str]`) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_teds_ai_bridge_chan(*physical_channel*, *name_to_assign_to_channel=u''*, *min_val=-0.002*, *max_val=0.002*, *units=<TEDSUnits.FROM_TEDS: 12516>*, *voltage_excit_source=<ExcitationSource.INTERNAL: 10200>*, *voltage_excit_val=2.5*, *custom_scale_name=u''*)

Creates channel(s) that measure a Wheatstone bridge. You must configure the physical channel(s) with TEDS information to use this function. Use this instance with bridge-based sensors that measure phenomena other than strain, force, pressure, or torque, or that scale data to physical units NI-DAQmx does not support.

Parameters

- **physical_channel** (`str`) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.

- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.TEDSUnits]*) – Specifies in which unit to return measurements from the channel.
- **voltage_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies the source of excitation.
- **voltage_excit_val** (*Optional[float]*) – Specifies in volts the amount of excitation supplied to the sensor. Refer to the sensor documentation to determine appropriate excitation values.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_teds_ai_current_chan(*physical_channel, name_to_assign_to_channel=u'', terminal_config=<TerminalConfiguration.DEFAULT: -1>, min_val=-0.01, max_val=0.01, units=<TEDSUnits.FROM_TEDS: 12516>, shunt_resistor_loc=<CurrentShuntResistorLocation.LET_DRIVER_CHOOSE: -1>, ext_shunt_resistor_val=249.0, custom_scale_name=u''*)

Creates channel(s) to measure current. You must configure the physical channel(s) with TEDS information to use this function.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **terminal_config** (*Optional[nidaqmx.constants.TerminalConfiguration]*) – Specifies the input terminal configuration for the channel.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.TEDSUnits]*) – Specifies the units to use to return measurements.
- **shunt_resistor_loc** (*Optional[nidaqmx.constants.CurrentShuntResistorLocation]*) – Specifies the location of the shunt resistor. For devices with built-in shunt resistors, specify the location as **INTERNAL**. For devices that do not have built-in shunt resistors, you must attach an external one, set

this input to **EXTERNAL** and use the **ext_shunt_resistor_val** input to specify the value of the resistor.

- **ext_shunt_resistor_val** (*Optional[float]*) – Specifies in ohms the resistance of an external shunt resistor.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_teds_ai_force_bridge_chan(*physical_channel*, *name_to_assign_to_channel=u''*, *min_val=-100.0*, *max_val=100.0*, *units=<ForceUnits.POUNDS: 15876>*, *voltage_excit_source=<ExcitationSource.INTERNAL: 10200>*, *voltage_excit_val=2.5*, *custom_scale_name=u''*)

Creates channel(s) that use a Wheatstone bridge to measure force or load. You must configure the physical channel(s) with TEDS information to use this function. NI-DAQmx scales electrical values to physical values according to that TEDS information.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.ForceUnits]*) – Specifies in which unit to return force measurements from the channel.
- **voltage_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies the source of excitation.
- **voltage_excit_val** (*Optional[float]*) – Specifies in volts the amount of excitation supplied to the sensor. Refer to the sensor documentation to determine appropriate excitation values.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_teds_ai_force_iepe_chan(*physical_channel*, *name_to_assign_to_channel=u''*, *terminal_config=<TerminalConfiguration.DEFAULT: -1>*, *min_val=-2000.0*, *max_val=2000.0*, *units=<ForceUnits.NEWTONS: 15875>*, *current_excit_source=<ExcitationSource.INTERNAL: 10200>*, *current_excit_val=0.001*, *custom_scale_name=u''*)

Creates channel(s) that use an IEPE force sensor to measure force or load. You must configure the physical channel(s) with TEDS information to use this function.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **terminal_config** (*Optional[nidaqmx.constants.TerminalConfiguration]*) – Specifies the input terminal configuration for the channel.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.ForceUnits]*) – Specifies in which unit to return force measurements from the channel.
- **current_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies the source of excitation.
- **current_excit_val** (*Optional[float]*) – Specifies in amperes the amount of excitation to supply to the sensor. Refer to the sensor documentation to determine this value.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_teds_ai_microphone_chan(*physical_channel*, *name_to_assign_to_channel=u''*, *terminal_config=<TerminalConfiguration.DEFAULT: -1>*, *units=<SoundPressureUnits.PA: 10081>*, *max_snd_press_level=100.0*, *current_excit_source=<ExcitationSource.INTERNAL: 10200>*, *current_excit_val=0.004*, *custom_scale_name=u''*)

Creates channel(s) that use a microphone to measure sound pressure. You must configure the physical channel(s) with TEDS information to use this function.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. You must use physical channels that you configured with TEDS information. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.

- **terminal_config** (*Optional[nidaqmx.constants.TerminalConfiguration]*) – Specifies the input terminal configuration for the channel.
- **units** (*Optional[nidaqmx.constants.SoundPressureUnits]*) – Specifies the units to use to return sound pressure measurements.
- **max_snd_press_level** (*Optional[float]*) – Is the maximum instantaneous sound pressure level you expect to measure. This value is in decibels, referenced to 20 micropascals.
- **current_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies the source of excitation.
- **current_excit_val** (*Optional[float]*) – Specifies in amperes the amount of excitation to supply to the sensor. Refer to the sensor documentation to determine this value.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_teds_ai_pos_lvdt_chan(*physical_channel, name_to_assign_to_channel=u'', min_val=-0.1, max_val=0.1, units=<LengthUnits.METERS: 10219>, voltage_excit_source=<ExcitationSource.INTERNAL: 10200>, voltage_excit_val=1.0, voltage_excit_freq=2500.0, ac_excit_wire_mode=<ACExcitWireMode.FOUR_WIRE: 4>, custom_scale_name=u''*)

Creates channel(s) that use an LVDT to measure linear position. You must configure the physical channel(s) with TEDS information to use this function.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.LengthUnits]*) – Specifies the units to use to return linear position measurements from the channel.
- **voltage_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies the source of excitation.
- **voltage_excit_val** (*Optional[float]*) – Specifies in volts the amount of excitation supplied to the sensor. Refer to the sensor documentation to determine appropriate excitation values.
- **voltage_excit_freq** (*Optional[float]*) – Specifies in hertz the excitation frequency that the sensor requires. Refer to the sensor documentation to determine this value.

- **ac_excit_wire_mode** (*Optional[nidaqmx.constants.ACExcitWireMode]*) – Is the number of leads on the sensor. Some sensors require you to tie leads together to create a four- or five- wire sensor. Refer to the sensor documentation for more information.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_teds_ai_pos_rvdt_chan(*physical_channel, name_to_assign_to_channel=u'', min_val=-70.0, max_val=70.0, units=<AngleUnits.DEGREES: 10146>, voltage_excit_source=<ExcitationSource.INTERNAL: 10200>, voltage_excit_val=1.0, voltage_excit_freq=2500.0, ac_excit_wire_mode=<ACExcitWireMode.FOUR_WIRE: 4>, custom_scale_name=u''*)

Creates channel(s) that use an RVDT to measure angular position. You must configure the physical channel(s) with TEDS information to use this function.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.AngleUnits]*) – Specifies the units to use to return angular position measurements from the channel.
- **voltage_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies the source of excitation.
- **voltage_excit_val** (*Optional[float]*) – Specifies in volts the amount of excitation supplied to the sensor. Refer to the sensor documentation to determine appropriate excitation values.
- **voltage_excit_freq** (*Optional[float]*) – Specifies in hertz the excitation frequency that the sensor requires. Refer to the sensor documentation to determine this value.
- **ac_excit_wire_mode** (*Optional[nidaqmx.constants.ACExcitWireMode]*) – Is the number of leads on the sensor. Some sensors require you to tie leads together to create a four- or five- wire sensor. Refer to the sensor documentation for more information.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_teds_ai_pressure_bridge_chan(*physical_channel*, *name_to_assign_to_channel=u''*, *min_val=-100.0*, *max_val=100.0*, *units=<PressureUnits.POUNDS_PER_SQ_INCH: 15879>*, *voltage_excit_source=<ExcitationSource.INTERNAL: 10200>*, *voltage_excit_val=2.5*, *custom_scale_name=u''*)

Creates channel(s) that use a Wheatstone bridge to measure pressure. You must configure the physical channel(s) with TEDS information to use this function. NI-DAQmx scales electrical values to physical values according to that TEDS information.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.PressureUnits]*) – Specifies in which unit to return pressure measurements from the channel.
- **voltage_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies the source of excitation.
- **voltage_excit_val** (*Optional[float]*) – Specifies in volts the amount of excitation supplied to the sensor. Refer to the sensor documentation to determine appropriate excitation values.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_teds_ai_resistance_chan(*physical_channel*, *name_to_assign_to_channel=u''*, *min_val=100.0*, *max_val=1000.0*, *units=<TEDSUnits.FROM_TEDS: 12516>*, *resistance_config=<ResistanceConfiguration.TWO_WIRE: 2>*, *current_excit_source=<ExcitationSource.EXTERNAL: 10167>*, *current_excit_val=0.001*, *custom_scale_name=u''*)

Creates channel(s) to measure resistance. You must configure the physical channel(s) with TEDS information to use this function.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.

- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.TEDSUnits]*) – Specifies the units to use to return measurements.
- **resistance_config** (*Optional[nidaqmx.constants.ResistanceConfiguration]*) – Specifies the number of wires to use for resistive measurements.
- **current_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies the source of excitation.
- **current_excit_val** (*Optional[float]*) – Specifies in amperes the amount of excitation to supply to the sensor. Refer to the sensor documentation to determine this value.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_teds_ai_rtd_chan(*physical_channel*, *name_to_assign_to_channel=u''*, *min_val=0.0*, *max_val=100.0*, *units=<TemperatureUnits.DEG_C: 10143>*, *resistance_config=<ResistanceConfiguration.TWO_WIRE: 2>*, *current_excit_source=<ExcitationSource.EXTERNAL: 10167>*, *current_excit_val=0.0025*)

Creates channel(s) that use an RTD to measure temperature. You must configure the physical channel(s) with TEDS information to use this function.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.TemperatureUnits]*) – Specifies the units to use to return temperature measurements.
- **resistance_config** (*Optional[nidaqmx.constants.ResistanceConfiguration]*) – Specifies the number of wires to use for resistive measurements.
- **current_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies the source of excitation.

- **current_excit_val** (*Optional[float]*) – Specifies in amperes the amount of excitation to supply to the sensor. Refer to the sensor documentation to determine this value.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_teds_ai_strain_gage_chan(*physical_channel*, *name_to_assign_to_channel=u''*, *min_val=-0.001*, *max_val=0.001*, *units=<StrainUnits.STRAIN: 10299>*, *voltage_excit_source=<ExcitationSource.INTERNAL: 10200>*, *voltage_excit_val=2.5*, *initial_bridge_voltage=0.0*, *lead_wire_resistance=0.0*, *custom_scale_name=u''*)

Creates channel(s) to measure strain. You must configure the physical channel(s) with TEDS information to use this function.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.StrainUnits]*) – Specifies the units to use to return strain measurements.
- **voltage_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies information about the bridge configuration and measurement.
- **voltage_excit_val** (*Optional[float]*) – Specifies information about the bridge configuration and measurement.
- **initial_bridge_voltage** (*Optional[float]*) – Specifies information about the bridge configuration and measurement.
- **lead_wire_resistance** (*Optional[float]*) – Specifies information about the bridge configuration and measurement.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_teds_ai_thrmcpl_chan(*physical_channel*, *name_to_assign_to_channel=u''*, *min_val=0.0*, *max_val=100.0*, *units=<TemperatureUnits.DEG_C: 10143>*, *cjc_source=<CJCSource.CONSTANT_USER_VALUE: 10116>*, *cjc_val=25.0*, *cjc_channel=u''*)

Creates channel(s) that use a thermocouple to measure temperature. You must configure the physical channel(s) with TEDS information to use this function.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.TemperatureUnits]*) – Specifies the units to use to return temperature measurements.
- **cjc_source** (*Optional[nidaqmx.constants.CJCSource]*) – Specifies the source of cold-junction compensation.
- **cjc_val** (*Optional[float]*) – Specifies in **units** the temperature of the cold junction if you set **cjc_source** to **CONSTANT_VALUE**.
- **cjc_channel** (*Optional[str]*) – Specifies the channel that acquires the temperature of the thermocouple cold- junction if you set **cjc_source** to **CHANNEL**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_teds_ai_thrmstr_chan_iex(*physical_channel*, *name_to_assign_to_channel=u''*, *min_val=0.0*, *max_val=100.0*, *units=<TemperatureUnits.DEG_C: 10143>*, *resistance_config=<ResistanceConfiguration.FOUR_WIRE: 4>*, *current_excit_source=<ExcitationSource.EXTERNAL: 10167>*, *current_excit_val=0.00015*)

Creates channel(s) that use a thermistor to measure temperature. Use this instance when the thermistor requires current excitation. You must configure the physical channel(s) with TEDS information to use this function.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.TemperatureUnits]*) – Specifies the units to use to return temperature measurements.

- **resistance_config** (*Optional[nidaqmx.constants.ResistanceConfiguration]*) – Specifies the number of wires to use for resistive measurements.
- **current_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies the source of excitation.
- **current_excit_val** (*Optional[float]*) – Specifies in amperes the amount of excitation to supply to the sensor. Refer to the sensor documentation to determine this value.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_teds_ai_thrmstr_chan_vex(*physical_channel, name_to_assign_to_channel=u'', min_val=0.0, max_val=100.0, units=<TemperatureUnits.DEG_C: 10143>, resistance_config=<ResistanceConfiguration.FOUR_WIRE: 4>, voltage_excit_source=<ExcitationSource.EXTERNAL: 10167>, voltage_excit_val=2.5, r_1=5000.0*)

Creates channel(s) that use a thermistor to measure temperature. Use this instance when the thermistor requires voltage excitation. You must configure the physical channel(s) with TEDS information to use this function.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.TemperatureUnits]*) – Specifies the units to use to return temperature measurements.
- **resistance_config** (*Optional[nidaqmx.constants.ResistanceConfiguration]*) – Specifies the number of wires to use for resistive measurements.
- **voltage_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies the source of excitation.
- **voltage_excit_val** (*Optional[float]*) – Specifies in volts the amount of excitation supplied to the sensor. Refer to the sensor documentation to determine appropriate excitation values.
- **r_1** (*Optional[float]*) – Specifies in ohms the value of the reference resistor.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_teds_ai_torque_bridge_chan(*physical_channel*, *name_to_assign_to_channel=u''*, *min_val=-100.0*, *max_val=100.0*, *units=<TorqueUnits.INCH_POUNDS: 15883>*, *voltage_excit_source=<ExcitationSource.INTERNAL: 10200>*, *voltage_excit_val=2.5*, *custom_scale_name=u''*)

Creates channel(s) that use a Wheatstone bridge to measure torque. You must configure the physical channel(s) with TEDS information to use this function. NI-DAQmx scales electrical values to physical values according to that TEDS information.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.TorqueUnits]*) – Specifies in which unit to return torque measurements from the channel.
- **voltage_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies the source of excitation.
- **voltage_excit_val** (*Optional[float]*) – Specifies in volts the amount of excitation supplied to the sensor. Refer to the sensor documentation to determine appropriate excitation values.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_teds_ai_voltage_chan(*physical_channel*, *name_to_assign_to_channel=u''*, *terminal_config=<TerminalConfiguration.DEFAULT: -1>*, *min_val=-5.0*, *max_val=5.0*, *units=<TEDSUnits.FROM_TEDS: 12516>*, *custom_scale_name=u''*)

Creates channel(s) to measure voltage. You must configure the physical channel(s) with TEDS information to use this function. If the measurement requires the use of internal excitation or you need excitation to scale the voltage, use the TEDS AI Custom Voltage with Excitation instance of this function.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.

- **terminal_config** (*Optional[nidaqmx.constants.TerminalConfiguration]*) – Specifies the input terminal configuration for the channel.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.TEDSUnits]*) – Specifies the units to use to return measurements.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

add_teds_ai_voltage_chan_with_excit(*physical_channel*, *name_to_assign_to_channel=u''*, *terminal_config=<TerminalConfiguration.DEFAULT: -1>*, *min_val=-10.0*, *max_val=10.0*, *units=<TEDSUnits.FROM_TEDS: 12516>*, *voltage_excit_source=<ExcitationSource.INTERNAL: 10200>*, *voltage_excit_val=0.0*, *custom_scale_name=u''*)

Creates channel(s) to measure voltage. Use this instance for custom sensors that require excitation. You can use the excitation to scale the measurement. You must configure the physical channel(s) with TEDS information to use this function.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **terminal_config** (*Optional[nidaqmx.constants.TerminalConfiguration]*) – Specifies the input terminal configuration for the channel.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.TEDSUnits]*) – Specifies the units to use to return measurements.
- **voltage_excit_source** (*Optional[nidaqmx.constants.ExcitationSource]*) – Specifies the source of excitation.
- **voltage_excit_val** (*Optional[float]*) – Specifies in volts the amount of excitation supplied to the sensor. Refer to the sensor documentation to determine appropriate excitation values.

- **custom_scale_name** (`Optional[str]`) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ai_channel.AIChannel*

all
`nidaqmx._task_modules.channels.channel.Channel` – Specifies a channel object that represents the entire list of virtual channels on this channel collection.

channel_names
List[str] – Specifies the entire list of virtual channels on this channel collection.

count (*value*) → integer – return number of occurrences of value

index (*value*) → integer – return first index of value.
Raises ValueError if the value is not present.

nidaqmx.task.ao_channel_collection

class nidaqmx._task_modules.ao_channel_collection.**AOChannelCollection** (*task_handle*)
Bases: `nidaqmx._task_modules.channel_collection.ChannelCollection`

Contains the collection of analog output channels for a DAQmx Task.

add_ao_current_chan (*physical_channel*, *name_to_assign_to_channel=u''*, *min_val=0.0*, *max_val=0.02*, *units=<CurrentUnits.AMPS: 10342>*, *custom_scale_name=u''*)
Creates channel(s) to generate current.

Parameters

- **physical_channel** (`str`) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (`Optional[str]`) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (`Optional[float]`) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (`Optional[float]`) – Specifies in **units** the maximum value you expect to measure.
- **units** (`Optional[nidaqmx.constants.CurrentUnits]`) – Specifies the units to use to generate current.
- **custom_scale_name** (`Optional[str]`) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ao_channel.AOChannel*

add_ao_func_gen_chan (*physical_channel*, *name_to_assign_to_channel=u''*, *type=<FuncGenType.SINE: 14751>*, *freq=1000.0*, *amplitude=5.0*, *offset=0.0*)
Creates a channel for continually generating a waveform on the selected physical channel.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **type** (*Optional[nidaqmx.constants.FuncGenType]*) – Specifies the kind of waveform to generate.
- **freq** (*Optional[float]*) – Is the frequency of the waveform to generate in hertz.
- **amplitude** (*Optional[float]*) – Is the zero-to-peak amplitude of the waveform to generate in volts. Zero and negative values are valid.
- **offset** (*Optional[float]*) – Is the voltage offset of the waveform to generate.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ao_channel.AOChannel*

add_ao_voltage_chan (*physical_channel, name_to_assign_to_channel=u'', min_val=-10.0, max_val=10.0, units=<VoltageUnits.VOLTS: 10348>, custom_scale_name=u''*)

Creates channel(s) to generate voltage.

Parameters

- **physical_channel** (*str*) – Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to generate.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to generate.
- **units** (*Optional[nidaqmx.constants.VoltageUnits]*) – Specifies the units to use to generate voltage.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ao_channel.AOChannel*

all

nidaqmx._task_modules.channels.channel.Channel – Specifies a channel object that represents the entire list of virtual channels on this channel collection.

channel_names

List[str] – Specifies the entire list of virtual channels on this channel collection.

count (*value*) → integer – return number of occurrences of value

index (*value*) → integer – return first index of value.
Raises ValueError if the value is not present.

nidaqmx.task.ci_channel_collection

class nidaqmx._task_modules.ci_channel_collection.**CIChannelCollection** (*task_handle*)
Bases: *nidaqmx._task_modules.channel_collection.ChannelCollection*

Contains the collection of counter input channels for a DAQmx Task.

add_ci_ang_encoder_chan (*counter, name_to_assign_to_channel=u'', decoding_type=<EncoderType.X_4: 10092>, zidx_enable=False, zidx_val=0, zidx_phase=<EncoderZIndexPhase.AHIGH_BHIGH: 10040>, units=<AngleUnits.DEGREES: 10146>, pulses_per_rev=24, initial_angle=0.0, custom_scale_name=u''*)

Creates a channel that uses an angular encoder to measure angular position. With the exception of devices that support multi-counter tasks, you can create only one counter input channel at a time with this function because a task can contain only one counter input channel. To read from multiple counters simultaneously, use a separate task for each counter. Connect the input signals to the default input terminals of the counter unless you select different input terminals.

Parameters

- **counter** (*str*) – Specifies the name of the counter to use to create the virtual channel. The DAQmx physical channel constant lists all physical channels, including counters, for devices installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **decoding_type** (*Optional[nidaqmx.constants.EncoderType]*) – Specifies how to count and interpret the pulses the encoder generates on signal A and signal B. **X_1**, **X_2**, and **X_4** are valid for quadrature encoders only. **TWO_PULSE_COUNTING** is valid only for two-pulse encoders.
- **zidx_enable** (*Optional[bool]*) – Specifies whether to use Z indexing for the channel.
- **zidx_val** (*Optional[float]*) – Specifies in **units** the value to which to reset the measurement when signal Z is high and signal A and signal B are at the states you specify with **zidx_phase**.
- **zidx_phase** (*Optional[nidaqmx.constants.EncoderZIndexPhase]*) – Specifies the states at which signal A and signal B must be while signal Z is high for NI-DAQmx to reset the measurement. If signal Z is never high while signal A and signal B are high, for example, you must choose a phase other than **A_HIGH_B_HIGH**.
- **units** (*Optional[nidaqmx.constants.AngleUnits]*) – Specifies the units to use to return angular position measurements from the channel.
- **pulses_per_rev** (*Optional[int]*) – Is the number of pulses the encoder generates per revolution. This value is the number of pulses on either signal A or signal B, not the total number of pulses on both signal A and signal B.
- **initial_angle** (*Optional[float]*) – Is the starting angle of the encoder. This value is in the units you specify with the **units** input.

- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ci_channel.CIChannel*

add_ci_ang_velocity_chan(*counter*, *name_to_assign_to_channel=u''*, *min_val=0.0*, *max_val=1.0*, *decoding_type=<EncoderType.X_4: 10092>*, *units=<AngularVelocityUnits.RPM: 16080>*, *pulses_per_rev=24*, *custom_scale_name=u''*)

Creates a channel to measure the angular velocity of a digital signal. With the exception of devices that support multi-counter tasks, you can create only one counter input channel at a time with this function because a task can contain only one counter input channel. To read from multiple counters simultaneously, use a separate task for each counter. Connect the input signal to the default input terminal of the counter unless you select a different input terminal.

Parameters

- **counter** (*str*) – Specifies the name of the counter to use to create the virtual channel. The DAQmx physical channel constant lists all physical channels, including counters, for devices installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **decoding_type** (*Optional[nidaqmx.constants.EncoderType]*) – Specifies how to count and interpret the pulses the encoder generates on signal A and signal B. **X_1**, **X_2**, and **X_4** are valid for quadrature encoders only. **TWO_PULSE_COUNTING** is valid only for two-pulse encoders.
- **units** (*Optional[nidaqmx.constants.AngularVelocityUnits]*) – Specifies in which unit to return velocity measurements from the channel.
- **pulses_per_rev** (*Optional[int]*) – Is the number of pulses the encoder generates per revolution. This value is the number of pulses on either signal A or signal B, not the total number of pulses on both signal A and signal B.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ci_channel.CIChannel*

add_ci_count_edges_chan(*counter*, *name_to_assign_to_channel=u''*, *edge=<Edge.RISING: 10280>*, *initial_count=0*, *count_direction=<CountDirection.COUNT_UP: 10128>*)

Creates a channel to count the number of rising or falling edges of a digital signal. With the exception of devices that support multi-counter tasks, you can create only one counter input channel at a time with this function because a task can contain only one counter input channel. To read from multiple counters simultaneously, use a separate task for each counter. Connect the input signal to the default input terminal of the counter unless you select a different input terminal.

Parameters

- **counter** (*str*) – Specifies the name of the counter to use to create the virtual channel. The DAQmx physical channel constant lists all physical channels, including counters, for devices installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **edge** (*Optional[nidaqmx.constants.Edge]*) – Specifies on which edges of the input signal to increment or decrement the count.
- **initial_count** (*Optional[int]*) – Is the value from which to start counting.
- **count_direction** (*Optional[nidaqmx.constants.CountDirection]*) – Specifies whether to increment or decrement the counter on each edge.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ci_channel.CIChannel*

add_ci_duty_cycle_chan(*counter*, *name_to_assign_to_channel=u''*, *min_freq=2.0*, *max_freq=10000.0*, *edge=<Edge.RISING: 10280>*, *custom_scale_name=u''*)

Creates channel(s) to duty cycle of a digital pulse. Connect the input signal to the default input terminal of the counter unless you select a different input terminal. With the exception of devices that support multi-counter tasks, you can create only one counter input channel at a time with this function because a task can contain only one counter input channel. To read from multiple counters simultaneously, use a separate task for each counter.

Parameters

- **counter** (*str*) – Specifies the name of the counter to use to create the virtual channel. The DAQmx physical channel constant lists all physical channels, including counters, for devices installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_freq** (*Optional[float]*) – Specifies the minimum frequency you expect to measure.
- **max_freq** (*Optional[float]*) – Specifies the maximum frequency you expect to measure.
- **edge** (*Optional[nidaqmx.constants.Edge]*) – Specifies between which edges to measure the frequency or period of the signal.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ci_channel.CIChannel*

add_ci_freq_chan(*counter*, *name_to_assign_to_channel=u''*, *min_val=2.0*, *max_val=100.0*, *units=<FrequencyUnits.HZ: 10373>*, *edge=<Edge.RISING: 10280>*, *meas_method=<CounterFrequencyMethod.LOW_FREQUENCY_1_COUNTER: 10105>*, *meas_time=0.001*, *divisor=4*, *custom_scale_name=u''*)

Creates a channel to measure the frequency of a digital signal. With the exception of devices that support multi-counter tasks, you can create only one counter input channel at a time with this function because a task can contain only one counter input channel. To read from multiple counters simultaneously, use a separate task for each counter. Connect the input signal to the default input terminal of the counter unless you select a different input terminal.

Parameters

- **counter** (*str*) – Specifies the name of the counter to use to create the virtual channel. The DAQmx physical channel constant lists all physical channels, including counters, for devices installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.FrequencyUnits]*) – Specifies the units to use to return frequency measurements.
- **edge** (*Optional[nidaqmx.constants.Edge]*) – Specifies between which edges to measure the frequency or period of the signal.
- **meas_method** (*Optional[nidaqmx.constants.CounterFrequencyMethod]*) – Specifies the method to use to calculate the period or frequency of the signal.
- **meas_time** (*Optional[float]*) – Is the length of time in seconds to measure the frequency or period of the signal if **meas_method** is **HIGH_FREQUENCYWITH_2_COUNTERS**. Leave this input unspecified if **meas_method** is not **HIGH_FREQUENCYWITH_2_COUNTERS**.
- **divisor** (*Optional[int]*) – Is the value by which to divide the input signal when **meas_method** is **LARGE_RANGEWITH_2_COUNTERS**. Leave this input unspecified if **meas_method** is not **LARGE_RANGEWITH_2_COUNTERS**.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ci_channel.CIChannel*

add_ci_gps_timestamp_chan(*counter*, *name_to_assign_to_channel=u''*, *units=<TimeUnits.SECONDS: 10364>*, *sync_method=<GpsSignalType.IRIGB: 10070>*, *custom_scale_name=u''*)

Creates a channel that uses a special purpose counter to take a timestamp and synchronizes that counter to a GPS receiver. With the exception of devices that support multi-counter tasks, you can create only one counter input channel at a time with this function because a task can contain only one counter input channel. To read from multiple counters simultaneously, use a separate task for each counter. Connect the input signals to the default input terminals of the counter unless you select different input terminals.

Parameters

- **counter** (*str*) – Specifies the name of the counter to use to create the virtual channel. The DAQmx physical channel constant lists all physical channels, including counters, for devices installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **units** (*Optional[nidaqmx.constants.TimeUnits]*) – Specifies the units to use to return the timestamp.
- **sync_method** (*Optional[nidaqmx.constants.GpsSignalType]*) – Specifies the method to use to synchronize the counter to a GPS receiver.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ci_channel.CIChannel*

add_ci_lin_encoder_chan(*counter*, *name_to_assign_to_channel=u''*, *decoding_type=<EncoderType.X_4: 10092>*, *zidx_enable=False*, *zidx_val=0*, *zidx_phase=<EncoderZIndexPhase.AHIGH_BHIGH: 10040>*, *units=<LengthUnits.METERS: 10219>*, *dist_per_pulse=0.001*, *initial_pos=0.0*, *custom_scale_name=u''*)

Creates a channel that uses a linear encoder to measure linear position. With the exception of devices that support multi- counter tasks, you can create only one counter input channel at a time with this function because a task can contain only one counter input channel. To read from multiple counters simultaneously, use a separate task for each counter. Connect the input signals to the default input terminals of the counter unless you select different input terminals.

Parameters

- **counter** (*str*) – Specifies the name of the counter to use to create the virtual channel. The DAQmx physical channel constant lists all physical channels, including counters, for devices installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **decoding_type** (*Optional[nidaqmx.constants.EncoderType]*) – Specifies how to count and interpret the pulses the encoder generates on signal A and signal B. **X_1**, **X_2**, and **X_4** are valid for quadrature encoders only. **TWO_PULSE_COUNTING** is valid only for two-pulse encoders.
- **zidx_enable** (*Optional[bool]*) – Specifies whether to use Z indexing for the channel.
- **zidx_val** (*Optional[float]*) – Specifies in **units** the value to which to reset the measurement when signal Z is high and signal A and signal B are at the states you specify with **zidx_phase**.
- **zidx_phase** (*Optional[nidaqmx.constants.EncoderZIndexPhase]*) – Specifies the states at which signal A and signal B must be while signal Z is high for NI-DAQmx to reset the measurement. If signal Z is never high while signal A and signal B are high, for example, you must choose a phase other than **A_HIGH_B_HIGH**.
- **units** (*Optional[nidaqmx.constants.LengthUnits]*) – Specifies the units to use to return linear position measurements from the channel.

- **dist_per_pulse** (*Optional[float]*) – Is the distance to measure for each pulse the encoder generates on signal A or signal B. This value is in the units you specify with the **units** input.
- **initial_pos** (*Optional[float]*) – Is the position of the encoder when you begin the measurement. This value is in the units you specify with the **units** input.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ci_channel.CIChannel*

add_ci_lin_velocity_chan(*counter*, *name_to_assign_to_channel=u''*, *min_val=0.0*, *max_val=1.0*, *decoding_type=<EncoderType.X_4: 10092>*, *units=<VelocityUnits.METERS_PER_SECOND: 15959>*, *dist_per_pulse=0.001*, *custom_scale_name=u''*)

Creates a channel that uses a linear encoder to measure linear velocity. With the exception of devices that support multi- counter tasks, you can create only one counter input channel at a time with this function because a task can contain only one counter input channel. To read from multiple counters simultaneously, use a separate task for each counter. Connect the input signal to the default input terminal of the counter unless you select a different input terminal.

Parameters

- **counter** (*str*) – Specifies the name of the counter to use to create the virtual channel. The DAQmx physical channel constant lists all physical channels, including counters, for devices installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **decoding_type** (*Optional[nidaqmx.constants.EncoderType]*) – Specifies how to count and interpret the pulses the encoder generates on signal A and signal B. **X_1**, **X_2**, and **X_4** are valid for quadrature encoders only. **TWO_PULSE_COUNTING** is valid only for two-pulse encoders.
- **units** (*Optional[nidaqmx.constants.VelocityUnits]*) – Specifies in which unit to return velocity measurements from the channel.
- **dist_per_pulse** (*Optional[float]*) – Is the distance to measure for each pulse the encoder generates on signal A or signal B. This value is in the units you specify with the **units** input.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ci_channel.CIChannel*

add_ci_period_chan(*counter*, *name_to_assign_to_channel=u''*, *min_val=1e-06*, *max_val=0.1*, *units=<TimeUnits.SECONDS: 10364>*, *edge=<Edge.RISING: 10280>*, *meas_method=<CounterFrequencyMethod.LOW_FREQUENCY_1_COUNTER: 10105>*, *meas_time=0.001*, *divisor=4*, *custom_scale_name=u''*)

Creates a channel to measure the period of a digital signal. With the exception of devices that support multi-counter tasks, you can create only one counter input channel at a time with this function because a task can contain only one counter input channel. To read from multiple counters simultaneously, use a separate task for each counter. Connect the input signal to the default input terminal of the counter unless you select a different input terminal.

Parameters

- **counter** (*str*) – Specifies the name of the counter to use to create the virtual channel. The DAQmx physical channel constant lists all physical channels, including counters, for devices installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.TimeUnits]*) – Specifies the units to use to return time or period measurements.
- **edge** (*Optional[nidaqmx.constants.Edge]*) – Specifies between which edges to measure the frequency or period of the signal.
- **meas_method** (*Optional[nidaqmx.constants.CounterFrequencyMethod]*) – Specifies the method to use to calculate the period or frequency of the signal.
- **meas_time** (*Optional[float]*) – Is the length of time in seconds to measure the frequency or period of the signal if **meas_method** is **HIGH_FREQUENCYWITH_2_COUNTERS**. Leave this input unspecified if **meas_method** is not **HIGH_FREQUENCYWITH_2_COUNTERS**.
- **divisor** (*Optional[int]*) – Is the value by which to divide the input signal when **meas_method** is **LARGE_RANGEWITH_2_COUNTERS**. Leave this input unspecified if **meas_method** is not **LARGE_RANGEWITH_2_COUNTERS**.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ci_channel.CIChannel*

add_ci_pulse_chan_freq(*counter*, *name_to_assign_to_channel=u''*, *min_val=1000*, *max_val=1000000*, *units=<FrequencyUnits.HZ: 10373>*)

Creates a channel to measure pulse specifications, returning the measurements as pairs of frequency and duty cycle. With the exception of devices that support multi-counter tasks, you can create only one counter input channel at a time with this function because a task can contain only one counter input channel. To read from multiple counters simultaneously, use a separate task for each counter. Connect the input signal to the default input terminal of the counter unless you select a different input terminal.

Parameters

- **counter** (*str*) – Specifies the name of the counter to use to create the virtual channel. The DAQmx physical channel constant lists all physical channels, including counters, for devices installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.FrequencyUnits]*) – Specifies the units to use to return pulse specifications in terms of frequency.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ci_channel.CIChannel*

add_ci_pulse_chan_ticks(*counter*, *name_to_assign_to_channel=u''*, *source_terminal=u'OnboardClock'*, *min_val=1000*, *max_val=1000000*)

Creates a channel to measure pulse specifications, returning the measurements as pairs of high ticks and low ticks. With the exception of devices that support multi-counter tasks, you can create only one counter input channel at a time with this function because a task can contain only one counter input channel. To read from multiple counters simultaneously, use a separate task for each counter. Connect the input signal to the default input terminal of the counter unless you select a different input terminal.

Parameters

- **counter** (*str*) – Specifies the name of the counter to use to create the virtual channel. The DAQmx physical channel constant lists all physical channels, including counters, for devices installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **source_terminal** (*Optional[str]*) – Is the terminal to which you connect a signal to use as the source of ticks. A DAQmx terminal constant lists all terminals available on devices installed in the system. You also can specify a source terminal by specifying a string that contains a terminal name. If you specify OnboardClock, or do not specify any terminal, NI-DAQmx selects the fastest onboard timebase available on the device.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ci_channel.CIChannel*

add_ci_pulse_chan_time(*counter*, *name_to_assign_to_channel=u''*, *min_val=1e-06*, *max_val=0.001*, *units=<TimeUnits.SECONDS: 10364>*)

Creates a channel to measure pulse specifications, returning the measurements as pairs of high time and low time. With the exception of devices that support multi-counter tasks, you can create only one counter input channel at a time with this function because a task can contain only one counter input channel. To read from multiple counters simultaneously, use a separate task for each counter. Connect the input signal to the default input terminal of the counter unless you select a different input terminal.

Parameters

- **counter** (*str*) – Specifies the name of the counter to use to create the virtual channel. The DAQmx physical channel constant lists all physical channels, including counters, for devices installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.TimeUnits]*) – Specifies the units to use to return pulse specifications in terms of high time and low time.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ci_channel.CIChannel*

add_ci_pulse_width_chan(*counter*, *name_to_assign_to_channel=u''*, *min_val=1e-06*, *max_val=0.1*, *units=<TimeUnits.SECONDS: 10364>*, *starting_edge=<Edge.RISING: 10280>*, *custom_scale_name=u''*)

Creates a channel to measure the width of a digital pulse. **starting_edge** determines whether to measure a high pulse or low pulse. With the exception of devices that support multi- counter tasks, you can create only one counter input channel at a time with this function because a task can contain only one counter input channel. To read from multiple counters simultaneously, use a separate task for each counter. Connect the input signal to the default input terminal of the counter unless you select a different input terminal.

Parameters

- **counter** (*str*) – Specifies the name of the counter to use to create the virtual channel. The DAQmx physical channel constant lists all physical channels, including counters, for devices installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.TimeUnits]*) – Specifies the units to use to return time or period measurements.
- **starting_edge** (*Optional[nidaqmx.constants.Edge]*) – Specifies on which edge to begin measuring pulse width.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ci_channel.CIChannel*

add_ci_semi_period_chan(*counter*, *name_to_assign_to_channel=u''*, *min_val=1e-06*, *max_val=0.1*, *units=<TimeUnits.SECONDS: 10364>*, *custom_scale_name=u''*)

Creates a channel to measure the time between state transitions of a digital signal. With the exception of devices that support multi-counter tasks, you can create only one counter input channel at a time with this function because a task can contain only one counter input channel. To read from multiple counters simultaneously, use a separate task for each counter. Connect the input signal to the default input terminal of the counter unless you select a different input terminal.

Parameters

- **counter** (*str*) – Specifies the name of the counter to use to create the virtual channel. The DAQmx physical channel constant lists all physical channels, including counters, for devices installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.
- **units** (*Optional[nidaqmx.constants.TimeUnits]*) – Specifies the units to use to return time or period measurements.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ci_channel.CIChannel*

add_ci_two_edge_sep_chan(*counter*, *name_to_assign_to_channel=u''*, *min_val=1e-06*, *max_val=1.0*, *units=<TimeUnits.SECONDS: 10364>*, *first_edge=<Edge.RISING: 10280>*, *second_edge=<Edge.FALLING: 10171>*, *custom_scale_name=u''*)

Creates a channel that measures the amount of time between the rising or falling edge of one digital signal and the rising or falling edge of another digital signal. With the exception of devices that support multi-counter tasks, you can create only one counter input channel at a time with this function because a task can contain only one counter input channel. To read from multiple counters simultaneously, use a separate task for each counter. Connect the input signals to the default input terminals of the counter unless you select different input terminals.

Parameters

- **counter** (*str*) – Specifies the name of the counter to use to create the virtual channel. The DAQmx physical channel constant lists all physical channels, including counters, for devices installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **min_val** (*Optional[float]*) – Specifies in **units** the minimum value you expect to measure.
- **max_val** (*Optional[float]*) – Specifies in **units** the maximum value you expect to measure.

- **units** (*Optional[nidaqmx.constants.TimeUnits]*) – Specifies the units to use to return time or period measurements.
- **first_edge** (*Optional[nidaqmx.constants.Edge]*) – Specifies on which edge of the first signal to start each measurement.
- **second_edge** (*Optional[nidaqmx.constants.Edge]*) – Specifies on which edge of the second signal to stop each measurement.
- **custom_scale_name** (*Optional[str]*) – Specifies the name of a custom scale for the channel. If you want the channel to use a custom scale, specify the name of the custom scale to this input and set **units** to **FROM_CUSTOM_SCALE**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.ci_channel.CIChannel*

all

nidaqmx._task_modules.channels.channel.Channel – Specifies a channel object that represents the entire list of virtual channels on this channel collection.

channel_names

List[str] – Specifies the entire list of virtual channels on this channel collection.

count (*value*) → integer – return number of occurrences of value

index (*value*) → integer – return first index of value.

Raises ValueError if the value is not present.

nidaqmx.task.co_channel_collection

class nidaqmx._task_modules.co_channel_collection.**COChannelCollection**(*task_handle*)

Bases: *nidaqmx._task_modules.channel_collection.ChannelCollection*

Contains the collection of counter output channels for a DAQmx Task.

add_co_pulse_chan_freq(*counter*, *name_to_assign_to_channel=u''*, *units=<FrequencyUnits.HZ: 10373>*, *idle_state=<Level.LOW: 10214>*, *initial_delay=0.0*, *freq=1.0*, *duty_cycle=0.5*)

Creates channel(s) to generate digital pulses that **freq** and **duty_cycle** define. The pulses appear on the default output terminal of the counter unless you select a different output terminal.

Parameters

- **counter** (*str*) – Specifies the names of the counters to use to create the virtual channels. The DAQmx physical channel constant lists all physical channels, including counters, for devices installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **units** (*Optional[nidaqmx.constants.FrequencyUnits]*) – Specifies the units in which to define pulse frequency.
- **idle_state** (*Optional[nidaqmx.constants.Level]*) – Specifies the resting state of the output terminal.
- **initial_delay** (*Optional[float]*) – Is the amount of time in seconds to wait before generating the first pulse.
- **freq** (*Optional[float]*) – Specifies at what frequency to generate pulses.

- **duty_cycle** (*Optional[float]*) – Is the width of the pulse divided by the pulse period. NI-DAQmx uses this ratio combined with frequency to determine pulse width and the interval between pulses.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.co_channel.COChannel*

add_co_pulse_chan_ticks(*counter*, *source_terminal*, *name_to_assign_to_channel=u''*, *idle_state=<Level.LOW: 10214>*, *initial_delay=0*, *low_ticks=100*, *high_ticks=100*)

Creates channel(s) to generate digital pulses defined by the number of timebase ticks that the pulse is at a high state and the number of timebase ticks that the pulse is at a low state. The pulses appear on the default output terminal of the counter unless you select a different output terminal.

Parameters

- **counter** (*str*) – Specifies the names of the counters to use to create the virtual channels. The DAQmx physical channel constant lists all physical channels, including counters, for devices installed in the system.
- **source_terminal** (*str*) – Is the terminal to which you connect an external timebase. A DAQmx terminal constant lists all terminals available on devices installed in the system. You also can specify a source terminal by specifying a string that contains a terminal name.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **idle_state** (*Optional[nidaqmx.constants.Level]*) – Specifies the resting state of the output terminal.
- **initial_delay** (*Optional[int]*) – Is the number of timebase ticks to wait before generating the first pulse.
- **low_ticks** (*Optional[int]*) – Is the number of ticks the pulse is low.
- **high_ticks** (*Optional[int]*) – Is the number of ticks the pulse is high.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.co_channel.COChannel*

add_co_pulse_chan_time(*counter*, *name_to_assign_to_channel=u''*, *units=<TimeUnits.SECONDS: 10364>*, *idle_state=<Level.LOW: 10214>*, *initial_delay=0.0*, *low_time=0.01*, *high_time=0.01*)

Creates channel(s) to generate digital pulses defined by the amount of time the pulse is at a high state and the amount of time the pulse is at a low state. The pulses appear on the default output terminal of the counter unless you select a different output terminal.

Parameters

- **counter** (*str*) – Specifies the names of the counters to use to create the virtual channels. The DAQmx physical channel constant lists all physical channels, including counters, for devices installed in the system.
- **name_to_assign_to_channel** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **units** (*Optional[nidaqmx.constants.TimeUnits]*) – Specifies the units in which to define pulse high and low time.

- **idle_state** (*Optional[nidaqmx.constants.Level]*) – Specifies the resting state of the output terminal.
- **initial_delay** (*Optional[float]*) – Is the amount of time in seconds to wait before generating the first pulse.
- **low_time** (*Optional[float]*) – Is the amount of time the pulse is low.
- **high_time** (*Optional[float]*) – Is the amount of time the pulse is high.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.co_channel.COChannel*

all
nidaqmx._task_modules.channels.channel.Channel – Specifies a channel object that represents the entire list of virtual channels on this channel collection.

channel_names
List[str] – Specifies the entire list of virtual channels on this channel collection.

count (*value*) → integer – return number of occurrences of value

index (*value*) → integer – return first index of value.
Raises ValueError if the value is not present.

nidaqmx.task.di_channel_collection

class nidaqmx._task_modules.di_channel_collection.**DIChannelCollection** (*task_handle*)
Bases: *nidaqmx._task_modules.channel_collection.ChannelCollection*

Contains the collection of digital input channels for a DAQmx Task.

add_di_chan (*lines*, *name_to_assign_to_lines=u''*, *line_grouping=<LineGrouping.CHAN_FOR_ALL_LINES: 1>*)
Creates channel(s) to measure digital signals. You can group digital lines into one digital channel or separate them into multiple digital channels. If you specify one or more entire ports in the **lines** input by using port physical channel names, you cannot separate the ports into multiple channels. To separate ports into multiple channels, use this function multiple times with a different port each time.

Parameters

- **lines** (*str*) – Specifies the names of the digital lines or ports to use to create virtual channels. The DAQmx physical channel constant lists all lines and ports for devices installed in the system.
- **name_to_assign_to_lines** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **line_grouping** (*Optional[nidaqmx.constants.LineGrouping]*) – Specifies how to group digital lines into one or more virtual channels. If you specify one or more entire ports with the **lines** input, you must set this input to **one channel for all lines**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.di_channel.DIChannel*

all
nidaqmx._task_modules.channels.channel.Channel – Specifies a channel object that represents the entire list of virtual channels on this channel collection.

channel_names
List[str] – Specifies the entire list of virtual channels on this channel collection.

count (*value*) → integer – return number of occurrences of value

index (*value*) → integer – return first index of value.
Raises ValueError if the value is not present.

nidaqmx.task.do_channel_collection

class nidaqmx._task_modules.do_channel_collection.**DOChannelCollection** (*task_handle*)
Bases: *nidaqmx._task_modules.channel_collection.ChannelCollection*

Contains the collection of digital output channels for a DAQmx Task.

add_do_chan (*lines*, *name_to_assign_to_lines=u''*, *line_grouping=<LineGrouping.CHAN_FOR_ALL_LINES: 1>*)
Creates channel(s) to generate digital signals. You can group digital lines into one digital channel or separate them into multiple digital channels. If you specify one or more entire ports in **lines** input by using port physical channel names, you cannot separate the ports into multiple channels. To separate ports into multiple channels, use this function multiple times with a different port each time.

Parameters

- **lines** (*str*) – Specifies the names of the digital lines or ports to use to create virtual channels. The DAQmx physical channel constant lists all lines and ports for devices installed in the system.
- **name_to_assign_to_lines** (*Optional[str]*) – Specifies a name to assign to the virtual channel this function creates. If you do not specify a value for this input, NI-DAQmx uses the physical channel name as the virtual channel name.
- **line_grouping** (*Optional[nidaqmx.constants.LineGrouping]*) – Specifies how to group digital lines into one or more virtual channels. If you specify one or more entire ports with the **lines** input, you must set this input to **one channel for all lines**.

Returns Indicates the newly created channel object.

Return type *nidaqmx._task_modules.channels.do_channel.DOChannel*

all
nidaqmx._task_modules.channels.channel.Channel – Specifies a channel object that represents the entire list of virtual channels on this channel collection.

channel_names
List[str] – Specifies the entire list of virtual channels on this channel collection.

count (*value*) → integer – return number of occurrences of value

index (*value*) → integer – return first index of value.
Raises ValueError if the value is not present.

nidaqmx.task.export_signals

class nidaqmx._task_modules.export_signals.**ExportSignals** (*task_handle*)
Bases: object

Represents the exported signal configurations for a DAQmx task.

adv_cmplt_event_delay
float – Specifies the output signal delay in periods of the sample clock.

adv_cmplt_event_output_term
str – Specifies the terminal to which to route the Advance Complete Event.

adv_cmplt_event_pulse_polarity
nidaqmx.constants.Polarity – Specifies the polarity of the exported Advance Complete Event.

adv_cmplt_event_pulse_width
float – Specifies the width of the exported Advance Complete Event pulse.

adv_trig_output_term
str – Specifies the terminal to which to route the Advance Trigger.

adv_trig_pulse_polarity
nidaqmx.constants.Polarity – Indicates the polarity of the exported Advance Trigger.

adv_trig_pulse_width
float – Specifies the width of an exported Advance Trigger pulse. Specify this value in the units you specify with **adv_trig_pulse_width_units**.

adv_trig_pulse_width_units
nidaqmx.constants.DigitalWidthUnits – Specifies the units of **adv_trig_pulse_width**.

ai_conv_clk_output_term
str – Specifies the terminal to which to route the AI Convert Clock.

ai_conv_clk_pulse_polarity
nidaqmx.constants.Polarity – Indicates the polarity of the exported AI Convert Clock. The polarity is fixed and independent of the active edge of the source of the AI Convert Clock.

ai_hold_cmplt_event_output_term
str – Specifies the terminal to which to route the AI Hold Complete Event.

ai_hold_cmplt_event_pulse_polarity
nidaqmx.constants.Polarity – Specifies the polarity of an exported AI Hold Complete Event pulse.

change_detect_event_output_term
str – Specifies the terminal to which to route the Change Detection Event.

change_detect_event_pulse_polarity
nidaqmx.constants.Polarity – Specifies the polarity of an exported Change Detection Event pulse.

ctr_out_event_output_behavior
nidaqmx.constants.ExportAction – Specifies whether the exported Counter Output Event pulses or changes from one state to the other when the counter reaches terminal count.

ctr_out_event_output_term
str – Specifies the terminal to which to route the Counter Output Event.

ctr_out_event_pulse_polarity
nidaqmx.constants.Polarity – Specifies the polarity of the pulses at the output terminal of the counter when **ctr_out_event_output_behavior** is **ExportActions2.PULSE**. NI-DAQmx ignores this property if **ctr_out_event_output_behavior** is **ExportActions2.TOGGLE**.

ctr_out_event_toggle_idle_state
nidaqmx.constants.Level – Specifies the initial state of the output terminal of the counter when **ctr_out_event_output_behavior** is **ExportActions2.TOGGLE**. The terminal enters this state when NI-DAQmx commits the task.

data_active_event_lvl_active_lvl
nidaqmx.constants.Polarity – Specifies the polarity of the exported Data Active Event.

data_active_event_output_term
str – Specifies the terminal to which to export the Data Active Event.

divided_samp_clk_timebase_output_term
str – Specifies the terminal to which to route the Divided Sample Clock Timebase.

export_signal (*signal_id*, *output_terminal*)
Routes a control signal to the terminal you specify. The output terminal can reside on the device that generates the control signal or on a different device. You can use this function to share clocks and triggers among multiple tasks and devices. The routes this function creates are task-based routes.

Parameters

- **signal_id** (`nidaqmx.constants.Signal`) – Is the name of the trigger, clock, or event to export.
- **output_terminal** (`str`) – Is the destination of the exported signal. A DAQmx terminal constant lists all terminals on installed devices. You can also specify a string containing a comma-delimited list of terminal names.

exported_10_m_hz_ref_clk_output_term
str – Specifies the terminal to which to route the 10MHz Clock.

exported_20_m_hz_timebase_output_term
str – Specifies the terminal to which to route the 20MHz Timebase.

hshk_event_delay
float – Specifies the number of seconds to delay after the Handshake Trigger deasserts before asserting the Handshake Event.

hshk_event_interlocked_assert_on_start
bool – Specifies to assert the Handshake Event when the task starts if **hshk_event_output_behavior** is **ExportActions5.INTERLOCKED**.

hshk_event_interlocked_asserted_lvl
`nidaqmx.constants.Level` – Specifies the asserted level of the exported Handshake Event if **hshk_event_output_behavior** is **ExportActions5.INTERLOCKED**.

hshk_event_interlocked_deassert_delay
float – Specifies in seconds the amount of time to wait after the Handshake Trigger asserts before deasserting the Handshake Event if **hshk_event_output_behavior** is **ExportActions5.INTERLOCKED**.

hshk_event_output_behavior
`nidaqmx.constants.ExportAction` – Specifies the output behavior of the Handshake Event.

hshk_event_output_term
str – Specifies the terminal to which to route the Handshake Event.

hshk_event_pulse_polarity
`nidaqmx.constants.Polarity` – Specifies the polarity of the exported Handshake Event if **hshk_event_output_behavior** is **ExportActions5.PULSE**.

hshk_event_pulse_width
float – Specifies in seconds the pulse width of the exported Handshake Event if **hshk_event_output_behavior** is **ExportActions5.PULSE**.

pause_trig_lvl_active_lvl
`nidaqmx.constants.Polarity` – Specifies the active level of the exported Pause Trigger.

pause_trig_output_term
str – Specifies the terminal to which to route the Pause Trigger.

rdy_for_start_event_lvl_active_lvl
`nidaqmx.constants.Polarity` – Specifies the polarity of the exported Ready for Start Event.

rdy_for_start_event_output_term
str – Specifies the terminal to which to route the Ready for Start Event.

rdy_for_xfer_event_deassert_cond
`nidaqmx.constants.DeassertCondition` – Specifies when the ready for transfer event deasserts.

rdy_for_xfer_event_deassert_cond_custom_threshold
int – Specifies in samples the threshold below which the Ready for Transfer Event deasserts. This threshold is an amount of space available in the onboard memory of the device. **rdy_for_xfer_event_deassert_cond** must be **DeassertCondition.ONBOARD_MEMORY_CUSTOM_THRESHOLD** to use a custom threshold.

rdy_for_xfer_event_lvl_active_lvl
`nidaqmx.constants.Polarity` – Specifies the active level of the exported Ready for Transfer Event.

rdy_for_xfer_event_output_term
str – Specifies the terminal to which to route the Ready for Transfer Event.

ref_trig_output_term
str – Specifies the terminal to which to route the Reference Trigger.

ref_trig_pulse_polarity
`nidaqmx.constants.Polarity` – Specifies the polarity of the exported Reference Trigger.

samp_clk_delay_offset
float – Specifies in seconds the amount of time to offset the exported Sample clock. Refer to timing diagrams for generation applications in the device documentation for more information about this value.

samp_clk_output_behavior
`nidaqmx.constants.ExportAction` – Specifies whether the exported Sample Clock issues a pulse at the beginning of a sample or changes to a high state for the duration of the sample.

samp_clk_output_term
str – Specifies the terminal to which to route the Sample Clock.

samp_clk_pulse_polarity
`nidaqmx.constants.Polarity` – Specifies the polarity of the exported Sample Clock if **samp_clk_output_behavior** is **ExportActions3.PULSE**.

samp_clk_timebase_output_term
str – Specifies the terminal to which to route the Sample Clock Timebase.

start_trig_output_term
str – Specifies the terminal to which to route the Start Trigger.

start_trig_pulse_polarity
`nidaqmx.constants.Polarity` – Specifies the polarity of the exported Start Trigger.

sync_pulse_event_output_term
str – Specifies the terminal to which to route the Synchronization Pulse Event.

watchdog_expired_event_output_term
str – Specifies the terminal to which to route the Watchdog Timer Expired Event.

nidaqmx.task.in_stream

class `nidaqmx._task_modules.in_stream.`**`InStream`**(*task*)

Bases: `object`

Exposes an input data stream on a DAQmx task.

The input data stream be used to control reading behavior and can be used in conjunction with reader classes to read samples from an NI-DAQmx task.

`accessory_insertion_or_removal_detected`

bool – Indicates if any device(s) in the task detected the insertion or removal of an accessory since the task started. Reading this property clears the accessory change status for all channels in the task. You must read this property before you read **devs_with_inserted_or_removed_accessories**. Otherwise, you will receive an error.

`auto_start`

bool – Specifies if DAQmx Read automatically starts the task if you did not start the task explicitly by using DAQmx Start. The default value is True. When DAQmx Read starts a finite acquisition task, it also stops the task after reading the last sample.

`avail_samp_per_chan`

int – Indicates the number of samples available to read per channel. This value is the same for all channels in the task.

`change_detect_overflowed`

bool – Indicates if samples were missed because change detection events occurred faster than the device could handle them. Some devices detect overflows differently than others.

`channels_to_read`

`nidaqmx._task_modules.channels.channel.Channel` – Specifies a subset of channels in the task from which to read.

`common_mode_range_error_chans`

List[str] – Indicates a list of names of any virtual channels in the task for which the device(s) detected a common mode range violation. You must read **common_mode_range_error_chans_exist** before you read this property. Otherwise, you will receive an error.

`common_mode_range_error_chans_exist`

bool – Indicates if the device(s) detected a common mode range violation for any virtual channel in the task. Common mode range violation occurs when the voltage of either the positive terminal or negative terminal to ground are out of range. Reading this property clears the common mode range violation status for all channels in the task. You must read this property before you read **common_mode_range_error_chans**. Otherwise, you will receive an error.

`configure_logging`(*file_path, logging_mode=<LoggingMode.LOG_AND_READ: 15842>, group_name=u'', operation=<LoggingOperation.OPEN_OR_CREATE: 15846>*)

Configures TDMS file logging for the task.

Parameters

- **`file_path`** (*`str`*) – Specifies the path to the TDMS file to which you want to log data.
- **`logging_mode`** (*`Optional[nidaqmx.constants.LoggingMode]`*) – Specifies whether to enable logging and whether to allow reading data while logging. "log" mode allows for the best performance. However, you cannot read data while logging if you specify this mode. If you want to read data while logging, specify "LOG_AND_READ" mode.

- **group_name** (*Optional[str]*) – Specifies the name of the group to create within the TDMS file for data from this task. If you append data to an existing file and the specified group already exists, NI-DAQmx appends a number symbol and a number to the group name, incrementing that number until finding a group name that does not exist. For example, if you specify a group name of Voltage Task, and that group already exists, NI-DAQmx assigns the group name Voltage Task #1, then Voltage Task #2. If you do not specify a group name, NI-DAQmx uses the name of the task.
- **operation** (*Optional[nidaqmx.constants.LoggingOperation]*) – Specifies how to open the TDMS file.

curr_read_pos
float – Indicates in samples per channel the current position in the buffer.

devs_with_inserted_or_removed_accessories
List[str] – Indicates the names of any devices that detected the insertion or removal of an accessory since the task started. You must read **accessory_insertion_or_removal_detected** before you read this property. Otherwise, you will receive an error.

di_num_booleans_per_chan
int – Indicates the number of booleans per channel that NI-DAQmx returns in a sample for line-based reads. If a channel has fewer lines than this number, the extra booleans are False.

excit_fault_chans
List[str] – Indicates a list of names of any virtual channels in the task for which the device(s) detected an excitation fault condition. You must read **excit_fault_chans_exist** before you read this property. Otherwise, you will receive an error.

excit_fault_chans_exist
bool – Indicates if the device(s) detected an excitation fault condition for any virtual channel in the task. Reading this property clears the excitation fault status for all channels in the task. You must read this property before you read **excit_fault_chans**. Otherwise, you will receive an error.

input_buf_size
int – Specifies the number of samples the input buffer can hold for each channel in the task. Zero indicates to allocate no buffer. Use a buffer size of 0 to perform a hardware-timed operation without using a buffer. Setting this property overrides the automatic input buffer allocation that NI- DAQmx performs.

input_onbrd_buf_size
int – Indicates in samples per channel the size of the onboard input buffer of the device.

logging_file_path
str – Specifies the path to the TDMS file to which you want to log data. If the file path is changed while the task is running, this takes effect on the next sample interval (if Logging.SampsPerFile has been set) or when DAQmx Start New File is called. New file paths can be specified by ending with "" or "/". Files created after specifying a new file path retain the same name and numbering sequence.

logging_file_preallocation_size
long – Specifies a size in samples to be used to pre-allocate space on disk. Pre-allocation can improve file I/O performance, especially in situations where multiple files are being written to disk. For finite tasks, the default behavior is to pre-allocate the file based on the number of samples you configure the task to acquire.

logging_file_write_size
int – Specifies the size, in samples, in which data will be written to disk. The size must be evenly divisible by the volume sector size, in bytes.

logging_mode
nidaqmx.constants.LoggingMode – Specifies whether to enable logging and whether to allow reading data while logging. Log mode allows for the best performance. However, you cannot read data

while logging if you specify this mode. If you want to read data while logging, specify Log and Read mode.

logging_pause
bool – Specifies whether logging is paused while a task is executing. If **logging_mode** is set to Log and Read mode, this value is taken into consideration on the next call to DAQmx Read, where data is written to disk. If **logging_mode** is set to Log Only mode, this value is taken into consideration the next time that data is written to disk. A new TDMS group is written when logging is resumed from a paused state.

logging_samps_per_file
long – Specifies how many samples to write to each file. When the file reaches the number of samples specified, a new file is created with the naming convention of <filename>_####.tdms, where #### starts at 0001 and increments automatically with each new file. For example, if the file specified is C:data.tdms, the next file name used is C:data_0001.tdms. To disable file spanning behavior, set this attribute to 0. If **logging_file_path** is changed while this attribute is set, the new file path takes effect on the next file created.

logging_tdms_group_name
str – Specifies the name of the group to create within the TDMS file for data from this task. If you append data to an existing file and the specified group already exists, NI- DAQmx appends a number symbol and a number to the group name, incrementing that number until finding a group name that does not exist. For example, if you specify a group name of Voltage Task, and that group already exists, NI- DAQmx assigns the group name Voltage Task #1, then Voltage Task #2.

logging_tdms_operation
`nidaqmx.constants.LoggingOperation` – Specifies how to open the TDMS file.

num_chans
int – Indicates the number of channels that DAQmx Read reads from the task. This value is the number of channels in the task or the number of channels you specify with **channels_to_read**.

offset
int – Specifies an offset in samples per channel at which to begin a read operation. This offset is relative to the location you specify with **relative_to**.

open_chans
List[str] – Indicates a list of names of any open virtual channels. You must read **open_chans_exist** before you read this property. Otherwise you will receive an error.

open_chans_details
List[str] – Indicates a list of details of any open virtual channels. You must read **open_chans_exist** before you read this property. Otherwise you will receive an error.

open_chans_exist
bool – Indicates if the device or devices detected an open channel condition in any virtual channel in the task. Reading this property clears the open channel status for all channels in this task. You must read this property before you read **open_chans**. Otherwise, you will receive an error.

open_current_loop_chans
List[str] – Indicates a list of names of any virtual channels in the task for which the device(s) detected an open current loop. You must read **open_current_loop_chans_exist** before you read this property. Otherwise, you will receive an error.

open_current_loop_chans_exist
bool – Indicates if the device(s) detected an open current loop for any virtual channel in the task. Reading this property clears the open current loop status for all channels in the task. You must read this property before you read **open_current_loop_chans**. Otherwise, you will receive an error.

open_thrmcpl_chans
List[str] – Indicates a list of names of any virtual channels in the task for which the device(s) detected an

open thermcouple. You must read **open_thrmcpl_chans_exist** before you read this property. Otherwise, you will receive an error.

open_thrmcpl_chans_exist
bool – Indicates if the device(s) detected an open thermocouple connected to any virtual channel in the task. Reading this property clears the open thermocouple status for all channels in the task. You must read this property before you read **open_thrmcpl_chans**. Otherwise, you will receive an error.

over_write
nidaqmx.constants.OverwriteMode – Specifies whether to overwrite samples in the buffer that you have not yet read.

overcurrent_chans
List[str] – Indicates a list of names of any virtual channels in the task for which the device(s) detected an overcurrent condition. You must read **overcurrent_chans_exist** before you read this property. Otherwise, you will receive an error. On some devices, you must restart the task for all overcurrent channels to recover.

overcurrent_chans_exist
bool – Indicates if the device(s) detected an overcurrent condition for any virtual channel in the task. Reading this property clears the overcurrent status for all channels in the task. You must read this property before you read **overcurrent_chans**. Otherwise, you will receive an error.

overloaded_chans
List[str] – Indicates a list of names of any overloaded virtual channels in the task. You must read **overloaded_chans_exist** before you read this property. Otherwise, you will receive an error.

overloaded_chans_exist
bool – Indicates if the device(s) detected an overload in any virtual channel in the task. Reading this property clears the overload status for all channels in the task. You must read this property before you read **overloaded_chans**. Otherwise, you will receive an error.

overtemperature_chans
List[str] – Indicates a list of names of any overtemperature virtual channels. You must read **overtemperature_chans_exist** before you read this property. Otherwise, you will receive an error.

overtemperature_chans_exist
bool – Indicates if the device(s) detected an overtemperature condition in any virtual channel in the task. Reading this property clears the overtemperature status for all channels in the task. You must read this property before you read **overtemperature_chans**. Otherwise, you will receive an error.

raw_data_width
int – Indicates in bytes the size of a raw sample from the task.

read(*number_of_samples_per_channel=-1*)
Reads raw samples from the task or virtual channels you specify.

Raw samples constitute the internal representation of samples in a device, read directly from the device or buffer without scaling or reordering. The native format of a device can be an 8-, 16-, or 32-bit integer, signed or unsigned.

NI-DAQmx does not separate raw data into channels. It returns data in an interleaved or non-interleaved 1D array, depending on the raw ordering of the device. Refer to your device documentation for more information.

This method determines a NumPy array of appropriate size and data type to create and return based on your device specifications.

Use the "timeout" property on the stream to specify the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

Parameters **number_of_samples_per_channel** (*int*) – Specifies the number of samples to read.

If you set this input to nidaqmx.READ_ALL_AVAILABLE, NI-DAQmx determines how many samples to read based on if the task acquires samples continuously or acquires a finite number of samples.

If the task acquires samples continuously and you set this input to nidaqmx.READ_ALL_AVAILABLE, this method reads all the samples currently available in the buffer.

If the task acquires a finite number of samples and you set this input to nidaqmx.READ_ALL_AVAILABLE, the method waits for the task to acquire all requested samples, then reads those samples. If you set the "read_all_avail_samp" property to TRUE, the method reads the samples currently available in the buffer and does not wait for the task to acquire all requested samples.

Returns The samples requested in the form of a 1D NumPy array. This method determines a NumPy array of appropriate size and data type to create and return based on your device specifications.

Return type numpy.ndarray

read_all_avail_samp

bool – Specifies whether subsequent read operations read all samples currently available in the buffer or wait for the buffer to become full before reading. NI-DAQmx uses this setting for finite acquisitions and only when the number of samples to read is -1. For continuous acquisitions when the number of samples to read is -1, a read operation always reads all samples currently available in the buffer.

readall()

Reads all available raw samples from the task or virtual channels you specify.

NI-DAQmx determines how many samples to read based on if the task acquires samples continuously or acquires a finite number of samples.

If the task acquires samples continuously, this method reads all the samples currently available in the buffer.

If the task acquires a finite number of samples, the method waits for the task to acquire all requested samples, then reads those samples. If you set the "read_all_avail_samp" property to TRUE, the method reads the samples currently available in the buffer and does not wait for the task to acquire all requested samples.

Raw samples constitute the internal representation of samples in a device, read directly from the device or buffer without scaling or reordering. The native format of a device can be an 8-, 16-, or 32-bit integer, signed or unsigned.

NI-DAQmx does not separate raw data into channels. It returns data in an interleaved or non-interleaved 1D array, depending on the raw ordering of the device. Refer to your device documentation for more information.

This method determines a NumPy array of appropriate size and data type to create and return based on your device specifications.

Use the "timeout" property on the stream to specify the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

Returns The samples requested in the form of a 1D NumPy array. This method determines a NumPy array of appropriate size and data type to create and return based on your device specifications.

Return type numpy.ndarray

readinto (*numpy_array*)

Reads raw samples from the task or virtual channels you specify into numpy_array.

The object numpy_array should be a pre-allocated, writable 1D numpy array.

The number of samples per channel to read is determined using the following equation:

number_of_samples_per_channel = math.floor(

numpy_array_size_in_bytes / (number_of_channels_to_read * raw_sample_size_in_bytes))

Raw samples constitute the internal representation of samples in a device, read directly from the device or buffer without scaling or reordering. The native format of a device can be an 8-, 16-, or 32-bit integer, signed or unsigned.

If you use a different integer size than the native format of the device, one integer can contain multiple samples or one sample can stretch across multiple integers. For example, if you use 32-bit integers, but the device uses 8-bit samples, one integer contains up to four samples. If you use 8-bit integers, but the device uses 16-bit samples, a sample might require two integers. This behavior varies from device to device. Refer to your device documentation for more information.

NI-DAQmx does not separate raw data into channels. It returns data in an interleaved or non-interleaved 1D array, depending on the raw ordering of the device. Refer to your device documentation for more information.

Use the "timeout" property on the stream to specify the amount of time in seconds to wait for samples to become available. If the time elapses, the method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to -1, the method waits indefinitely. If you set timeout to 0, the method tries once to read the requested samples and returns an error if it is unable to.

Parameters **numpy_array** – Specifies the 1D NumPy array object into which the samples requested are read.

Returns Indicates the total number of samples read.

Return type int

relative_to

nidaqmx.constants.ReadRelativeTo – Specifies the point in the buffer at which to begin a read operation. If you also specify an offset with **offset**, the read operation begins at that offset relative to the point you select with this property. The default value is **ReadRelativeTo.CURRENT_READ_POSITION** unless you configure a Reference Trigger for the task. If you configure a Reference Trigger, the default value is **ReadRelativeTo.FIRST_PRETRIGGER_SAMPLE**.

sleep_time

float – Specifies in seconds the amount of time to sleep after checking for available samples if **wait_mode** is **WaitMode.SLEEP**.

start_new_file (*file_path*)

Starts a new TDMS file the next time data is written to disk.

Parameters **file_path** (*str*) – Specifies the path to the TDMS file to which you want to log data.

timeout
float – Specifies the amount of time in seconds to wait for samples to become available. If the time elapses, the read method returns an error and any samples read before the timeout elapsed. The default timeout is 10 seconds. If you set timeout to nidaqmx.WAIT_INFINITELY, the read method waits indefinitely. If you set timeout to 0, the read method tries once to read the requested samples and returns an error if it is unable to.

total_samp_per_chan_acquired
float – Indicates the total number of samples acquired by each channel. NI-DAQmx returns a single value because this value is the same for all channels. For retriggered acquisitions, this value is the cumulative number of samples across all retriggered acquisitions.

wait_mode
`nidaqmx.constants.WaitMode` – Specifies how DAQmx Read waits for samples to become available.

nidaqmx.task.out_stream

class `nidaqmx._task_modules.out_stream.`**`OutStream`**(*task*)
Bases: `object`

Exposes an output data stream on a DAQmx task.

The output data stream be used to control writing behavior and can be used in conjunction with writer classes to write samples to an NI-DAQmx task.

accessory_insertion_or_removal_detected
bool – Indicates if any devices in the task detected the insertion or removal of an accessory since the task started. Reading this property clears the accessory change status for all channels in the task. You must read this property before you read **devs_with_inserted_or_removed_accessories**. Otherwise, you will receive an error.

auto_start
bool – Specifies if the “write” method automatically starts the stream’s owning task if you did not explicitly start it with the DAQmx Start Task method.

curr_write_pos
float – Indicates the position in the buffer of the next sample to generate. This value is identical for all channels in the task.

devs_with_inserted_or_removed_accessories
List[str] – Indicates the names of any devices that detected the insertion or removal of an accessory since the task started. You must read **accessory_insertion_or_removal_detected** before you read this property. Otherwise, you will receive an error.

do_num_booleans_per_chan
int – Indicates the number of Boolean values expected per channel in a sample for line-based writes. This property is determined by the channel in the task with the most digital lines. If a channel has fewer lines than this number, NI- DAQmx ignores the extra Boolean values.

external_overvoltage_chans
List[str] – Indicates a list of names of any virtual channels in the task for which an External Overvoltage condition has been detected. You must read External OvervoltageChansExist before you read this property. Otherwise, you will receive an error.

external_overvoltage_chans_exist
bool – Indicates if the device(s) detected an External Overvoltage condition for any channel in the task. Reading this property clears the External Overvoltage status for all channels in the task. You must read this property before you read External OvervoltageChans. Otherwise, you will receive an error.

num_chans
int – Indicates the number of channels that DAQmx Write writes to the task. This value is the number of channels in the task.

offset
int – Specifies in samples per channel an offset at which a write operation begins. This offset is relative to the location you specify with **relative_to**.

open_current_loop_chans
List[str] – Indicates a list of names of any virtual channels in the task for which the device(s) detected an open current loop. You must read **open_current_loop_chans_exist** before you read this property. Otherwise, you will receive an error.

open_current_loop_chans_exist
bool – Indicates if the device(s) detected an open current loop for any channel in the task. Reading this property clears the open current loop status for all channels in the task. You must read this property before you read **open_current_loop_chans**. Otherwise, you will receive an error.

output_buf_size
int – Specifies the number of samples the output buffer can hold for each channel in the task. Zero indicates to allocate no buffer. Use a buffer size of 0 to perform a hardware-timed operation without using a buffer. Setting this property overrides the automatic output buffer allocation that NI- DAQmx performs.

output_onbrd_buf_size
int – Specifies in samples per channel the size of the onboard output buffer of the device.

overcurrent_chans
List[str] – Indicates a list of names of any virtual channels in the task for which an overcurrent condition has been detected. You must read **overcurrent_chans_exist** before you read this property. Otherwise, you will receive an error.

overcurrent_chans_exist
bool – Indicates if the device(s) detected an overcurrent condition for any channel in the task. Reading this property clears the overcurrent status for all channels in the task. You must read this property before you read **overcurrent_chans**. Otherwise, you will receive an error.

overloaded_chans
List[str] – Indicates a list of names of any overloaded virtual channels in the task. You must read **overloaded_chans_exist** before you read this property. Otherwise, you will receive an error.

overloaded_chans_exist
bool – Indicates if the device(s) detected an overload in any virtual channel in the task. Reading this property clears the overload status for all channels in the task. You must read this property before you read **overloaded_chans**. Otherwise, you will receive an error.

overtemperature_chans
List[str] – Indicates a list of names of any overtemperature virtual channels. You must read **overtemperature_chans_exist** before you read this property. Otherwise, you will receive an error. The list of names may be empty if the device cannot determine the source of the overtemperature.

overtemperature_chans_exist
bool – Indicates if the device(s) detected an overtemperature condition in any virtual channel in the task. Reading this property clears the overtemperature status for all channels in the task. You must read this property before you read **overtemperature_chans**. Otherwise, you will receive an error.

power_supply_fault_chans
List[str] – Indicates a list of names of any virtual channels in the task that have a power supply fault. You must read **power_supply_fault_chans_exist** before you read this property. Otherwise, you will receive an error.

power_supply_fault_chans_exist
bool – Indicates if the device(s) detected a power supply fault for any channel in the task. Reading this property clears the power supply fault status for all channels in the task. You must read this property before you read **power_supply_fault_chans**. Otherwise, you will receive an error.

raw_data_width
int – Indicates in bytes the required size of a raw sample to write to the task.

regen_mode
nidaqmx.constants.RegenerationMode – Specifies whether to allow NI-DAQmx to generate the same data multiple times.

relative_to
nidaqmx.constants.WriteRelativeTo – Specifies the point in the buffer at which to write data. If you also specify an offset with **offset**, the write operation begins at that offset relative to this point you select with this property.

sleep_time
float – Specifies in seconds the amount of time to sleep after checking for available buffer space if **wait_mode** is **WaitMode2.SLEEP**.

space_avail
int – Indicates in samples per channel the amount of available space in the buffer.

timeout
float – Specifies the amount of time in seconds to wait for the write method to write all samples. NI-DAQmx performs a timeout check only if the write method must wait before it writes data. The write method returns an error if the time elapses. The default timeout is 10 seconds. If you set "timeout" to nidaqmx.WAIT_INFINITELY, the write method waits indefinitely. If you set timeout to 0, the write method tries once to write the submitted samples. If the write method could not write all the submitted samples, it returns an error and the number of samples successfully written in the number of samples written per channel output.

total_samp_per_chan_generated
float – Indicates the total number of samples generated by each channel in the task. This value is identical for all channels in the task.

wait_mode
nidaqmx.constants.WaitMode – Specifies how DAQmx Write waits for space to become available in the buffer.

write(*numpy_array*)
Writes raw samples to the task or virtual channels you specify.

The number of samples per channel to write is determined using the following equation:

number_of_samples_per_channel = math.floor(

numpy_array_size_in_bytes / (number_of_channels_to_write * raw_sample_size_in_bytes))

Raw samples constitute the internal representation of samples in a device, read directly from the device or buffer without scaling or reordering. The native format of a device can be an 8-, 16-, or 32-bit integer, signed or unsigned.

If you use a different integer size than the native format of the device, one integer can contain multiple samples or one sample can stretch across multiple integers. For example, if you use 32-bit integers, but the device uses 8-bit samples, one integer contains up to four samples. If you use 8-bit integers, but the device uses 16-bit samples, a sample might require two integers. This behavior varies from device to device. Refer to your device documentation for more information.

NI-DAQmx does not separate raw data into channels. It accepts data in an interleaved or non-interleaved 1D array, depending on the raw ordering of the device. Refer to your device documentation for more information.

If the task uses on-demand timing, this method returns only after the device generates all samples. On-demand is the default timing type if you do not use the timing property on the task to configure a sample timing type. If the task uses any timing type other than on-demand, this method returns immediately and does not wait for the device to generate all samples. Your application must determine if the task is done to ensure that the device generated all samples.

Use the "auto_start" property on the stream to specify if this method automatically starts the stream's owning task if you did not explicitly start it with the DAQmx Start Task method.

Use the "timeout" property on the stream to specify the amount of time in seconds to wait for the method to write all samples. NI-DAQmx performs a timeout check only if the method must wait before it writes data. This method returns an error if the time elapses. The default timeout is 10 seconds. If you set timeout to nidaqmx.WAIT_INFINITELY, the method waits indefinitely. If you set timeout to 0, the method tries once to write the submitted samples. If the method could not write all the submitted samples, it returns an error and the number of samples successfully written.

Parameters **`numpy_array`** (*`numpy.ndarray`*) – Specifies a 1D NumPy array that contains the raw samples to write to the task.

Returns Specifies the actual number of samples per channel successfully written to the buffer.

Return type int

nidaqmx.task.timing

class `nidaqmx._task_modules.timing.`**`Timing`**(*task_handle*)

Bases: `object`

Represents the timing configurations for a DAQmx task.

`ai_conv_active_edge`

`nidaqmx.constants.Edge` – Specifies on which edge of the clock pulse an analog-to-digital conversion takes place.

`ai_conv_dig_fltr_enable`

bool – Specifies whether to apply a digital filter to the AI Convert Clock.

`ai_conv_dig_fltr_min_pulse_width`

float – Specifies in seconds the minimum pulse width the filter recognizes.

`ai_conv_dig_fltr_timebase_rate`

float – Specifies in hertz the rate of the digital filter timebase. NI-DAQmx uses this value to compute settings for the filter.

`ai_conv_dig_fltr_timebase_src`

str – Specifies the terminal of the signal to use as the timebase of the digital filter.

`ai_conv_dig_sync_enable`

bool – Specifies whether to synchronize recognition of transitions in the signal to the internal timebase of the device.

`ai_conv_max_rate`

float – Indicates the maximum convert rate supported by the task, given the current devices and channel count.

ai_conv_rate
float – Specifies in Hertz the rate at which to clock the analog- to-digital converter. This clock is specific to the analog input section of multiplexed devices.

ai_conv_src
str – Specifies the terminal of the signal to use as the AI Convert Clock.

ai_conv_timebase_div
int – Specifies the number of AI Convert Clock Timebase pulses needed to produce a single AI Convert Clock pulse.

ai_conv_timebase_src
nidaqmx.constants.MIOAIConvertTimebaseSource – Specifies the terminal of the signal to use as the AI Convert Clock Timebase.

cfg_burst_handshaking_timing_export_clock(*sample_clk_rate*, *sample_clk_outp_term*, *sample_mode=<AcquisitionType.FINITE: 10178>*, *samps_per_chan=1000*, *sample_clk_pulse_polarity=<Polarity.ACTIVE_HIGH: 10095>*, *pause_when=<Level.HIGH: 10192>*, *ready_event_active_level=<Polarity.ACTIVE_HIGH: 10095>*)
Configures when the DAQ device transfers data to a peripheral device, using the onboard Sample Clock of the DAQ device to control burst handshake timing and exporting that clock for use by the peripheral device.

Parameters

- **sample_clk_rate** (*float*) – Specifies in hertz the rate of the Sample Clock.
- **sample_clk_outp_term** (*str*) – Specifies the terminal to which to export the Sample Clock.
- **sample_mode** (*Optional[nidaqmx.constants.AcquisitionType]*) – Specifies if the task acquires or generates samples continuously or if it acquires or generates a finite number of samples.
- **samps_per_chan** (*Optional[long]*) – Specifies the number of samples to acquire or generate for each channel in the task if **sample_mode** is **FINITE_SAMPLES**. If **sample_mode** is **CONTINUOUS_SAMPLES**, NI-DAQmx uses this value to determine the buffer size. This function returns an error if the specified value is negative.
- **sample_clk_pulse_polarity** (*Optional[nidaqmx.constants.Polarity]*) – Specifies the polarity of the exported Sample Clock.
- **pause_when** (*Optional[nidaqmx.constants.Level]*) – Specifies whether the task pauses while the trigger signal is high or low.
- **ready_event_active_level** (*Optional[nidaqmx.constants.Polarity]*) – Specifies the polarity of the Ready for Transfer Event.

cfg_burst_handshaking_timing_import_clock(*sample_clk_rate*, *sample_clk_src*, *sample_mode=<AcquisitionType.FINITE: 10178>*, *samps_per_chan=1000*, *sample_clk_active_edge=<Edge.RISING: 10280>*, *pause_when=<Level.HIGH: 10192>*, *ready_event_active_level=<Polarity.ACTIVE_HIGH: 10095>*)
Configures when the DAQ device transfers data to a peripheral device, using an imported sample clock to control burst handshake timing.

Parameters

- **sample_clk_rate** (*float*) – Specifies in hertz the rate of the Sample Clock.
- **sample_clk_src** (*str*) – Specifies the source terminal of the Sample Clock. Leave this input unspecified to use the default onboard clock of the device.
- **sample_mode** (*Optional[nidaqmx.constants.AcquisitionType]*) – Specifies if the task acquires or generates samples continuously or if it acquires or generates a finite number of samples.
- **samps_per_chan** (*Optional[long]*) – Specifies the number of samples to acquire or generate for each channel in the task if **sample_mode** is **FINITE_SAMPLES**. If **sample_mode** is **CONTINUOUS_SAMPLES**, NI-DAQmx uses this value to determine the buffer size. This function returns an error if the specified value is negative.
- **sample_clk_active_edge** (*Optional[nidaqmx.constants.Edge]*) – Specifies on which edges of Sample Clock pulses to acquire or generate samples.
- **pause_when** (*Optional[nidaqmx.constants.Level]*) – Specifies whether the task pauses while the trigger signal is high or low.
- **ready_event_active_level** (*Optional[nidaqmx.constants.Polarity]*) – Specifies the polarity of the Ready for Transfer Event.

cfg_change_detection_timing(*rising_edge_chan=u'', falling_edge_chan=u'', sample_mode=<AcquisitionType.FINITE: 10178>, samps_per_chan=1000*)

Configures the task to acquire samples on the rising and/or falling edges of the lines or ports you specify. To detect both rising and falling edges on a line or port, specify the name of that line or port to both **rising_edge_chan** and **falling_edge_chan**.

Parameters

- **rising_edge_chan** (*Optional[str]*) – Specifies the names of the digital lines or ports on which to detect rising edges. The DAQmx physical channel constant lists all lines and ports for devices installed in your system.
- **falling_edge_chan** (*Optional[str]*) – Specifies the names of the digital lines or ports on which to detect falling edges. The DAQmx physical channel constant lists all lines and ports for devices installed in your system.
- **sample_mode** (*Optional[nidaqmx.constants.AcquisitionType]*) – Specifies if the task acquires samples continuously or if it acquires a finite number of samples.
- **samps_per_chan** (*Optional[long]*) – Specifies the number of samples to acquire from each channel in the task if **sample_mode** is **FINITE_SAMPLES**. This function returns an error if the specified value is negative.

cfg_handshaking_timing(*sample_mode=<AcquisitionType.FINITE: 10178>, samps_per_chan=1000*)

Determines the number of digital samples to acquire or generate using digital handshaking between the device and a peripheral device.

Parameters

- **sample_mode** (*Optional[nidaqmx.constants.AcquisitionType]*) – Specifies if the task acquires or generates samples continuously or if it acquires or generates a finite number of samples.

- **samps_per_chan** (*Optional[long]*) – Specifies the number of samples to acquire or generate for each channel in the task if **sample_mode** is **FINITE_SAMPLES**. If **sample_mode** is **CONTINUOUS_SAMPLES**, NI-DAQmx uses this value to determine the buffer size. This function returns an error if the specified value is negative.

cfg_implicit_timing(*sample_mode=<AcquisitionType.FINITE: 10178>, samps_per_chan=1000*)

Sets only the number of samples to acquire or generate without specifying timing. Typically, you should use this instance when the task does not require sample timing, such as tasks that use counters for buffered frequency measurement, buffered period measurement, or pulse train generation. For finite counter output tasks, **samps_per_chan** is the number of pulses to generate.

Parameters

- **sample_mode** (*Optional[nidaqmx.constants.AcquisitionType]*) – Specifies if the task acquires or generates samples continuously or if it acquires or generates a finite number of samples.
- **samps_per_chan** (*Optional[long]*) – Specifies the number of samples to acquire or generate for each channel in the task if **sample_mode** is **FINITE_SAMPLES**. If **sample_mode** is **CONTINUOUS_SAMPLES**, NI-DAQmx uses this value to determine the buffer size. This function returns an error if the specified value is negative.

cfg_pipelined_samp_clk_timing(*rate, source=u'', active_edge=<Edge.RISING: 10280>, sample_mode=<AcquisitionType.FINITE: 10178>, samps_per_chan=1000*)

"Sets the source of the Sample Clock, the rate of the Sample Clock, and the number of samples to acquire or generate. The device acquires or generates samples on each Sample Clock edge, but it does not respond to certain triggers until a few Sample Clock edges later. Pipelining allows higher data transfer rates at the cost of increased trigger response latency. Refer to the device documentation for information about which triggers pipelining affects.

This timing type allows handshaking using the Pause trigger and either the Ready for Transfer event or the Data Active event. Refer to the device documentation for more information.

This timing type is supported only by the NI 6536 and NI 6537."

Args:

rate (float): Specifies the sampling rate in samples per channel per second. If you use an external source for the Sample Clock, set this input to the maximum expected rate of that clock.

source (Optional[str]): Specifies the source terminal of the Sample Clock. Leave this input unspecified to use the default onboard clock of the device.

active_edge (Optional[nidaqmx.constants.Edge]): Specifies on which edges of Sample Clock pulses to acquire or generate samples.

sample_mode (Optional[nidaqmx.constants.AcquisitionType]): Specifies if the task acquires or generates samples continuously or if it acquires or generates a finite number of samples.

samps_per_chan (Optional[long]): Specifies the number of samples to acquire or generate for each channel in the task if **sample_mode** is **FINITE_SAMPLES**. If **sample_mode** is **CONTINUOUS_SAMPLES**, NI-DAQmx uses this value to determine the buffer size. This function returns an error if the specified value is negative.

cfg_samp_clk_timing(*rate, source=u'', active_edge=<Edge.RISING: 10280>, sample_mode=<AcquisitionType.FINITE: 10178>, samps_per_chan=1000*)

Sets the source of the Sample Clock, the rate of the Sample Clock, and the number of samples to acquire or generate.

Parameters

- **rate** (*float*) – Specifies the sampling rate in samples per channel per second. If you use an external source for the Sample Clock, set this input to the maximum expected rate of that clock.
- **source** (*Optional[str]*) – Specifies the source terminal of the Sample Clock. Leave this input unspecified to use the default onboard clock of the device.
- **active_edge** (*Optional[nidaqmx.constants.Edge]*) – Specifies on which edges of Sample Clock pulses to acquire or generate samples.
- **sample_mode** (*Optional[nidaqmx.constants.AcquisitionType]*) – Specifies if the task acquires or generates samples continuously or if it acquires or generates a finite number of samples.
- **samps_per_chan** (*Optional[long]*) – Specifies the number of samples to acquire or generate for each channel in the task if **sample_mode** is **FINITE_SAMPLES**. If **sample_mode** is **CONTINUOUS_SAMPLES**, NI-DAQmx uses this value to determine the buffer size. This function returns an error if the specified value is negative.

change_detect_di_falling_edge_physical_chans

nidaqmx.system.physical_channel.PhysicalChannel – Specifies the names of the digital lines or ports on which to detect falling edges. The lines or ports must be used by virtual channels in the task. You also can specify a string that contains a list or range of digital lines or ports.

change_detect_di_rising_edge_physical_chans

nidaqmx.system.physical_channel.PhysicalChannel – Specifies the names of the digital lines or ports on which to detect rising edges. The lines or ports must be used by virtual channels in the task. You also can specify a string that contains a list or range of digital lines or ports.

change_detect_di_tristate

bool – Specifies whether to tristate lines specified with **change_detect_di_rising_edge_physical_chans** and **change_detect_di_falling_edge_physical_chans** that are not in a virtual channel in the task. If you set this property to True, NI-DAQmx tristates rising/falling edge lines that are not in a virtual channel in the task. If you set this property to False, NI-DAQmx does not modify the configuration of rising/falling edge lines that are not in a virtual channel in the task, even if the lines were previously tristated. Set this property to False to detect changes on lines in other tasks or to detect changes on output-only lines.

delay_from_samp_clk_delay

float – Specifies the amount of time to wait after receiving a Sample Clock edge before beginning to acquire the sample. This value is in the units you specify with **delay_from_samp_clk_delay_units**.

delay_from_samp_clk_delay_units

nidaqmx.constants.DigitalWidthUnits – Specifies the units of **delay_from_samp_clk_delay**.

hshk_delay_after_xfer

float – Specifies the number of seconds to wait after a handshake cycle before starting a new handshake cycle.

hshk_sample_input_data_when

nidaqmx.constants.SampleInputDataWhen – Specifies on which edge of the Handshake Trigger an input task latches the data from the peripheral device.

hshk_start_cond

nidaqmx.constants.HandshakeStartCondition – Specifies the point in the handshake cycle that the device is in when the task starts.

implicit_underflow_behavior

nidaqmx.constants.UnderflowBehavior – Specifies the action to take when the onboard memory of the device becomes empty.

master_timebase_rate
float – Specifies the rate of the Master Timebase.

master_timebase_src
str – Specifies the terminal of the signal to use as the Master Timebase. On an E Series device, you can choose only between the onboard 20MHz Timebase or the RTSI7 terminal.

ref_clk_rate
float – Specifies the frequency of the Reference Clock.

ref_clk_src
str – Specifies the terminal of the signal to use as the Reference Clock.

samp_clk_active_edge
nidaqmx.constants.Edge – Specifies on which edge of a clock pulse sampling takes place. This property is useful primarily when the signal you use as the Sample Clock is not a periodic clock.

samp_clk_dig_fltr_enable
bool – Specifies whether to apply the pulse width filter to the signal.

samp_clk_dig_fltr_min_pulse_width
float – Specifies in seconds the minimum pulse width the filter recognizes.

samp_clk_dig_fltr_timebase_rate
float – Specifies in hertz the rate of the pulse width filter timebase. NI-DAQmx uses this value to compute settings for the filter.

samp_clk_dig_fltr_timebase_src
str – Specifies the input terminal of the signal to use as the timebase of the pulse width filter.

samp_clk_dig_sync_enable
bool – Specifies whether to synchronize recognition of transitions in the signal to the internal timebase of the device.

samp_clk_max_rate
float – Indicates the maximum Sample Clock rate supported by the task, based on other timing settings. For output tasks, the maximum Sample Clock rate is the maximum rate of the DAC. For input tasks, NI-DAQmx calculates the maximum sampling rate differently for multiplexed devices than simultaneous sampling devices.

samp_clk_overrun_behavior
nidaqmx.constants.OverflowBehavior – Specifies the action to take if Sample Clock edges occur faster than the device can handle them.

samp_clk_rate
float – Specifies the sampling rate in samples per channel per second. If you use an external source for the Sample Clock, set this input to the maximum expected rate of that clock.

samp_clk_src
str – Specifies the terminal of the signal to use as the Sample Clock.

samp_clk_term
str – Indicates the name of the internal Sample Clock terminal for the task. This property does not return the name of the Sample Clock source terminal specified with **samp_clk_src**.

samp_clk_timebase_active_edge
nidaqmx.constants.Edge – Specifies on which edge to recognize a Sample Clock Timebase pulse. This property is useful primarily when the signal you use as the Sample Clock Timebase is not a periodic clock.

samp_clk_timebase_div
int – Specifies the number of Sample Clock Timebase pulses needed to produce a single Sample Clock pulse.

samp_clk_timebase_master_timebase_div
int – Specifies the number of pulses of the Master Timebase needed to produce a single pulse of the Sample Clock Timebase.

samp_clk_timebase_rate
float – Specifies the rate of the Sample Clock Timebase. Some applications require that you specify a rate when you use any signal other than the onboard Sample Clock Timebase. NI- DAQmx requires this rate to calculate other timing parameters.

samp_clk_timebase_src
str – Specifies the terminal of the signal to use as the Sample Clock Timebase.

samp_clk_timebase_term
str – Indicates the name of the internal Sample Clock Timebase terminal for the task. This property does not return the name of the Sample Clock Timebase source terminal specified with **samp_clk_timebase_src**.

samp_clk_underflow_behavior
`nidaqmx.constants.UnderflowBehavior` – Specifies the action to take when the onboard memory of the device becomes empty. In either case, the sample clock does not stop.

samp_clk_write_wfm_use_initial_wfm_dt
bool – Specifies that the value of **samp_clk_rate** will be determined by the dt component of the initial DAQmx Write waveform input for Output tasks.

samp_quant_samp_mode
`nidaqmx.constants.AcquisitionType` – Specifies if a task acquires or generates a finite number of samples or if it continuously acquires or generates samples.

samp_quant_samp_per_chan
float – Specifies the number of samples to acquire or generate for each channel if **samp_quant_samp_mode** is **AcquisitionType.FINITE**. If **samp_quant_samp_mode** is **AcquisitionType.CONTINUOUS**, NI-DAQmx uses this value to determine the buffer size.

samp_timing_engine
int – Specifies which timing engine to use for the task.

samp_timing_type
`nidaqmx.constants.SampleTimingType` – Specifies the type of sample timing to use for the task.

simultaneous_ao_enable
bool – Specifies whether to update all channels in the task simultaneously, rather than updating channels independently when you write a sample to that channel.

sync_clk_interval
int – Specifies the interval, in Sample Clock periods, between each internal Synchronization Clock pulse. NI-DAQmx uses this pulse for synchronization of triggers between multiple devices at different rates. Refer to device documentation for information about how to calculate this value.

sync_pulse_min_delay_to_start
float – Specifies in seconds the amount of time that elapses after the master device issues the synchronization pulse before the task starts.

sync_pulse_reset_delay
float – Specifies in seconds the amount of time to wait after the Synchronization Pulse before resetting the ADCs or DACs on the device. When synchronizing devices, query **sync_pulse_reset_time** on all devices

and note the largest reset time. Then, for each device, subtract the reset time from the largest reset time and set this property to the resulting value.

sync_pulse_reset_time
float – Indicates in seconds the amount of time required for the ADCs or DACs on the device to reset. When synchronizing devices, query this property on all devices and note the largest reset time. Then, for each device, subtract the value of this property from the largest reset time and set **sync_pulse_reset_delay** to the resulting value.

sync_pulse_src
str – Specifies the terminal of the signal to use as the synchronization pulse. The synchronization pulse resets the clock dividers and the ADCs/DACs on the device.

sync_pulse_sync_time
float – Indicates in seconds the delay required to reset the ADCs/DACs after the device receives the synchronization pulse.

sync_pulse_term
str – Indicates the name of the internal Synchronization Pulse terminal for the task. This property does not return the name of the source terminal.

nidaqmx.task.triggers

class nidaqmx._task_modules.triggers.**Triggers**(*task_handle*)
Bases: object

Represents the trigger configurations for a DAQmx task.

arm_start_trigger
nidaqmx._task_modules.triggering.arm_start_trigger.ArmStartTrigger – Gets the arm start trigger configurations for the task.

handshake_trigger
nidaqmx._task_modules.triggering.handshake_trigger.HandshakeTrigger – Gets the handshake trigger configurations for the task.

pause_trigger
nidaqmx._task_modules.triggering.pause_trigger.PauseTrigger – Gets the pause trigger configurations for the task.

reference_trigger
nidaqmx._task_modules.triggering.reference_trigger.ReferenceTrigger – Gets the reference trigger configurations for the task.

start_trigger
nidaqmx._task_modules.triggering.start_trigger.StartTrigger – Gets the start trigger configurations for the task.

sync_type
nidaqmx.constants.SyncType – Specifies the role of the device in a synchronized system. Setting this value to **SyncType.MASTER** or **SyncType.SLAVE** enables trigger skew correction. If you enable trigger skew correction, set this property to **SyncType.MASTER** on only one device, and set this property to **SyncType.SLAVE** on the other devices.

nidaqmx.task.arm_start_trigger

class nidaqmx._task_modules.triggering.arm_start_trigger.**ArmStartTrigger**(*task_handle*)
Bases: object

Represents the arm start trigger configurations for a DAQmx task.

dig_edge_dig_fltr_enable
bool – Specifies whether to apply the pulse width filter to the signal.

dig_edge_dig_fltr_min_pulse_width
float – Specifies in seconds the minimum pulse width the filter recognizes.

dig_edge_dig_fltr_timebase_rate
float – Specifies in hertz the rate of the pulse width filter timebase. NI-DAQmx uses this value to compute settings for the filter.

dig_edge_dig_fltr_timebase_src
str – Specifies the input terminal of the signal to use as the timebase of the pulse width filter.

dig_edge_dig_sync_enable
bool – Specifies whether to synchronize recognition of transitions in the signal to the internal timebase of the device.

dig_edge_edge
`nidaqmx.constants.Edge` – Specifies on which edge of a digital signal to arm the task for a Start Trigger.

dig_edge_src
str – Specifies the name of a terminal where there is a digital signal to use as the source of the Arm Start Trigger.

term
str – Indicates the name of the internal Arm Start Trigger terminal for the task. This property does not return the name of the trigger source terminal.

trig_type
`nidaqmx.constants.TriggerType` – Specifies the type of trigger to use to arm the task for a Start Trigger. If you configure an Arm Start Trigger, the task does not respond to a Start Trigger until the device receives the Arm Start Trigger.

nidaqmx.task.handshake_trigger

class `nidaqmx._task_modules.triggering.handshake_trigger.`**`HandshakeTrigger`**(*task_handle*)
Bases: `object`

Represents the handshake trigger configurations for a DAQmx task.

interlocked_asserted_lvl
`nidaqmx.constants.Level` – Specifies the asserted level of the Handshake Trigger.

interlocked_src
str – Specifies the source terminal of the Handshake Trigger.

trig_type
`nidaqmx.constants.TriggerType` – Specifies the type of Handshake Trigger to use.

nidaqmx.task.pause_trigger

class `nidaqmx._task_modules.triggering.pause_trigger.`**`PauseTrigger`**(*task_handle*)
Bases: `object`

Represents the pause trigger configurations for a DAQmx task.

anlg_lvl_coupling
nidaqmx.constants.Coupling – Specifies the coupling for the source signal of the trigger if the source is a terminal rather than a virtual channel.

anlg_lvl_dig_fltr_enable
bool – Specifies whether to apply a digital filter to the digital output of the analog triggering circuitry (the Analog Comparison Event). When enabled, the analog signal must stay above or below the trigger level for the minimum pulse width before being recognized. Use filtering for noisy trigger signals that transition in and out of the hysteresis window rapidly.

anlg_lvl_dig_fltr_min_pulse_width
float – Specifies in seconds the minimum pulse width the filter recognizes.

anlg_lvl_dig_fltr_timebase_rate
float – Specifies in hertz the rate of the digital filter timebase. NI-DAQmx uses this value to compute settings for the filter.

anlg_lvl_dig_fltr_timebase_src
str – Specifies the terminal of the signal to use as the timebase of the digital filter.

anlg_lvl_dig_sync_enable
bool – Specifies whether to synchronize recognition of transitions in the signal to the internal timebase of the device.

anlg_lvl_hyst
float – Specifies a hysteresis level in the units of the measurement or generation. If **anlg_lvl_when** is **ActiveLevel.ABOVE**, the trigger does not deassert until the source signal passes below **anlg_lvl_lvl** minus the hysteresis. If **anlg_lvl_when** is **ActiveLevel.BELOW**, the trigger does not deassert until the source signal passes above **anlg_lvl_lvl** plus the hysteresis. Hysteresis is always enabled. Set this property to a non-zero value to use hysteresis.

anlg_lvl_lvl
float – Specifies the threshold at which to pause the task. Specify this value in the units of the measurement or generation. Use **anlg_lvl_when** to specify whether the task pauses above or below this threshold.

anlg_lvl_src
str – Specifies the name of a virtual channel or terminal where there is an analog signal to use as the source of the trigger.

anlg_lvl_when
nidaqmx.constants.ActiveLevel – Specifies whether the task pauses above or below the threshold you specify with **anlg_lvl_lvl**.

anlg_win_btm
float – Specifies the lower limit of the window. Specify this value in the units of the measurement or generation.

anlg_win_coupling
nidaqmx.constants.Coupling – Specifies the coupling for the source signal of the terminal if the source is a terminal rather than a virtual channel.

anlg_win_dig_fltr_enable
bool – Specifies whether to apply a digital filter to the digital output of the analog triggering circuitry (the Analog Comparison Event). When enabled, the analog signal must stay within the trigger window for the minimum pulse width before being recognized. Use filtering for noisy trigger signals that transition in and out of the window rapidly.

anlg_win_dig_fltr_min_pulse_width
float – Specifies in seconds the minimum pulse width the filter recognizes.

anlg_win_dig_fltr_timebase_rate
float – Specifies in hertz the rate of the digital filter timebase. NI-DAQmx uses this value to compute settings for the filter.

anlg_win_dig_fltr_timebase_src
str – Specifies the terminal of the signal to use as the timebase of the digital filter.

anlg_win_dig_sync_enable
bool – Specifies whether to synchronize recognition of transitions in the signal to the internal timebase of the device.

anlg_win_src
str – Specifies the name of a virtual channel or terminal where there is an analog signal to use as the source of the trigger.

anlg_win_top
float – Specifies the upper limit of the window. Specify this value in the units of the measurement or generation.

anlg_win_when
`nidaqmx.constants.WindowTriggerCondition2` – Specifies whether the task pauses while the trigger signal is inside or outside the window you specify with **anlg_win_btm** and **anlg_win_top**.

dig_lvl_dig_fltr_enable
bool – Specifies whether to apply a digital filter to the trigger signal.

dig_lvl_dig_fltr_min_pulse_width
float – Specifies in seconds the minimum pulse width the filter recognizes.

dig_lvl_dig_fltr_timebase_rate
float – Specifies in hertz the rate of the pulse width filter timebase. NI-DAQmx uses this value to compute settings for the filter.

dig_lvl_dig_fltr_timebase_src
str – Specifies the input terminal of the signal to use as the timebase of the pulse width filter.

dig_lvl_dig_sync_enable
bool – Specifies whether to synchronize recognition of transitions in the signal to the internal timebase of the device.

dig_lvl_src
str – Specifies the name of a terminal where there is a digital signal to use as the source of the Pause Trigger.

dig_lvl_when
`nidaqmx.constants.Level` – Specifies whether the task pauses while the signal is high or low.

dig_pattern_pattern
str – Specifies the digital pattern that must be met for the Pause Trigger to occur.

dig_pattern_src
`nidaqmx.system.physical_channel.PhysicalChannel` – Specifies the physical channels to use for pattern matching. The order of the physical channels determines the order of the pattern. If a port is included, the lines within the port are in ascending order.

dig_pattern_when
`nidaqmx.constants.DigitalPatternCondition` – Specifies if the Pause Trigger occurs when the physical channels specified with **dig_pattern_src** match or differ from the digital pattern specified with **dig_pattern_pattern**.

term
str – Indicates the name of the internal Pause Trigger terminal for the task. This property does not return the name of the trigger source terminal.

trig_type
nidaqmx.constants.TriggerType – Specifies the type of trigger to use to pause a task.

nidaqmx.task.reference_trigger

class nidaqmx._task_modules.triggering.reference_trigger.**ReferenceTrigger**(*task_handle*)
Bases: object

Represents the reference trigger configurations for a DAQmx task.

anlg_edge_coupling
nidaqmx.constants.Coupling – Specifies the coupling for the source signal of the trigger if the source is a terminal rather than a virtual channel.

anlg_edge_dig_fltr_enable
bool – Specifies whether to apply a digital filter to the digital output of the analog triggering circuitry (the Analog Comparison Event). When enabled, the analog signal must stay above or below the trigger level for the minimum pulse width before being recognized. Use filtering for noisy trigger signals that transition in and out of the hysteresis window rapidly.

anlg_edge_dig_fltr_min_pulse_width
float – Specifies in seconds the minimum pulse width thefilter recognizes.

anlg_edge_dig_fltr_timebase_rate
float – Specifies in hertz the rate of the digital filter timebase. NI-DAQmx uses this value to compute settings for the filter.

anlg_edge_dig_fltr_timebase_src
str – Specifies the terminal of the signal to use as the timebase of the digital filter.

anlg_edge_dig_sync_enable
bool – Specifies whether to synchronize recognition of transitions in the signal to the internal timebase of the device.

anlg_edge_hyst
float – Specifies a hysteresis level in the units of the measurement. If **anlg_edge_slope** is **Slope1.RISING**, the trigger does not deassert until the source signal passes below **anlg_edge_lvl** minus the hysteresis. If **anlg_edge_slope** is **Slope1.FALLING**, the trigger does not deassert until the source signal passes above **anlg_edge_lvl** plus the hysteresis. Hysteresis is always enabled. Set this property to a non-zero value to use hysteresis.

anlg_edge_lvl
float – Specifies in the units of the measurement the threshold at which the Reference Trigger occurs. Use **anlg_edge_slope** to specify on which slope to trigger at this threshold.

anlg_edge_slope
nidaqmx.constants.Slope – Specifies on which slope of the source signal the Reference Trigger occurs.

anlg_edge_src
str – Specifies the name of a virtual channel or terminal where there is an analog signal to use as the source of the Reference Trigger.

anlg_win_btm
float – Specifies the lower limit of the window. Specify this value in the units of the measurement.

anlg_win_coupling
nidaqmx.constants.Coupling – Specifies the coupling for the source signal of the trigger if the source is a terminal rather than a virtual channel.

anlg_win_dig_fltr_enable
bool – Specifies whether to apply a digital filter to the digital output of the analog triggering circuitry (the Analog Comparison Event). When enabled, the analog signal must stay within the trigger window for the minimum pulse width before being recognized. Use filtering for noisy trigger signals that transition in and out of the window rapidly.

anlg_win_dig_fltr_min_pulse_width
float – Specifies in seconds the minimum pulse width the filter recognizes.

anlg_win_dig_fltr_timebase_rate
float – Specifies in hertz the rate of the digital filter timebase. NI-DAQmx uses this value to compute settings for the filter.

anlg_win_dig_fltr_timebase_src
str – Specifies the terminal of the signal to use as the timebase of the digital filter.

anlg_win_dig_sync_enable
bool – Specifies whether to synchronize recognition of transitions in the signal to the internal timebase of the device.

anlg_win_src
str – Specifies the name of a virtual channel or terminal where there is an analog signal to use as the source of the Reference Trigger.

anlg_win_top
float – Specifies the upper limit of the window. Specify this value in the units of the measurement.

anlg_win_trig_when
nidaqmx.constants.WindowTriggerCondition1 – Specifies whether the Reference Trigger occurs when the source signal enters the window or when it leaves the window. Use **anlg_win_btm** and **anlg_win_top** to specify the window.

auto_trig_enable
bool – Specifies whether to send a software trigger to the device when a hardware trigger is no longer active in order to prevent a timeout.

auto_triggered
bool – Indicates whether a completed acquisition was triggered by the auto trigger. If an acquisition has not completed after the task starts, this property returns False. This property is only applicable when **auto_trig_enable** is True.

cfg_anlg_edge_ref_trig(*trigger_source*, *pretrigger_samples*, *trigger_slope=<Slope.RISING: 10280>*, *trigger_level=0.0*)
Configures the task to stop the acquisition when the device acquires all pretrigger samples; an analog signal reaches the level you specify; and the device acquires all post-trigger samples. When you use a Reference Trigger, the default for the read RelativeTo property is **first_pretrigger_sample** with a read Offset of 0.

Parameters

- **trigger_source** (*str*) – Is the name of a virtual channel or terminal where there is an analog signal to use as the source of the trigger.
- **pretrigger_samples** (*int*) – Specifies the minimum number of samples to acquire per channel before recognizing the Reference Trigger. The number of post-trigger samples per channel is equal to **number of samples per channel** in the DAQmx Timing function minus **pretrigger_samples**.

- **trigger_slope** (*Optional[nidaqmx.constants.Slope]*) – Specifies on which slope of the signal the Reference Trigger occurs.
- **trigger_level** (*Optional[float]*) – Specifies at what threshold to trigger. Specify this value in the units of the measurement or generation. Use **trigger_slope** to specify on which slope to trigger at this threshold.

cfg_anlg_window_ref_trig (*trigger_source*, *window_top*, *window_bottom*, *pretrigger_samples*, *trigger_when=<WindowTriggerCondition1.ENTERING_WINDOW: 10163>*)

Configures the task to stop the acquisition when the device acquires all pretrigger samples; an analog signal enters or leaves a range you specify; and the device acquires all post- trigger samples. When you use a Reference Trigger, the default for the read RelativeTo property is **first_pretrigger_sample** with a read Offset of 0.

Parameters

- **trigger_source** (*str*) – Is the name of a virtual channel or terminal where there is an analog signal to use as the source of the trigger.
- **window_top** (*float*) – Is the upper limit of the window. Specify this value in the units of the measurement or generation.
- **window_bottom** (*float*) – Is the lower limit of the window. Specify this value in the units of the measurement or generation.
- **pretrigger_samples** (*int*) – Specifies the minimum number of samples to acquire per channel before recognizing the Reference Trigger. The number of post-trigger samples per channel is equal to **number of samples per channel** in the DAQmx Timing function minus **pretrigger_samples**.
- **trigger_when** (*Optional[nidaqmx.constants.WindowTriggerCondition1]*) – Specifies whether the Reference Trigger occurs when the signal enters the window or when it leaves the window. Use **window_bottom** and **window_top** to specify the limits of the window.

cfg_dig_edge_ref_trig (*trigger_source*, *pretrigger_samples*, *trigger_edge=<Edge.RISING: 10280>*)

Configures the task to stop the acquisition when the device acquires all pretrigger samples, detects a rising or falling edge of a digital signal, and acquires all posttrigger samples. When you use a Reference Trigger, the default for the read RelativeTo property is **first_pretrigger_sample** with a read Offset of 0.

Parameters

- **trigger_source** (*str*) – Specifies the name of a terminal where there is a digital signal to use as the source of the trigger.
- **pretrigger_samples** (*int*) – Specifies the minimum number of samples to acquire per channel before recognizing the Reference Trigger. The number of post-trigger samples per channel is equal to **number of samples per channel** in the DAQmx Timing function minus **pretrigger_samples**.
- **trigger_edge** (*Optional[nidaqmx.constants.Edge]*) – Specifies on which edge of the digital signal the Reference Trigger occurs.

cfg_dig_pattern_ref_trig (*trigger_source*, *trigger_pattern*, *pretrigger_samples*, *trigger_when=<DigitalPatternCondition.PATTERN_MATCHES: 10254>*)

Configures the task to stop the acquisition when the device acquires all pretrigger samples, matches a digital pattern, and acquires all posttrigger samples. When you use a Reference Trigger, the default for the read RelativeTo property is First PretriggerSample with a read Offset of zero.

Parameters

- **trigger_source** (*str*) – Specifies the physical channels to use for pattern matching. The order of the physical channels determines the order of the pattern. If a port is included, the order of the physical channels within the port is in ascending order.
- **trigger_pattern** (*str*) – Specifies the digital pattern that must be met for the trigger to occur.
- **pretrigger_samples** (*int*) – Specifies the minimum number of samples to acquire per channel before recognizing the Reference Trigger. The number of post-trigger samples per channel is equal to **number of samples per channel** in the DAQmx Timing function minus **pretrigger_samples**.
- **trigger_when** (*Optional[nidaqmx.constants.DigitalPatternCondition]*) – Specifies the condition under which the trigger occurs.

delay
float – Specifies in seconds the time to wait after the device receives the Reference Trigger before switching from pretrigger to posttrigger samples.

dig_edge_dig_fltr_enable
bool – Specifies whether to apply a digital filter to the trigger signal.

dig_edge_dig_fltr_min_pulse_width
float – Specifies in seconds the minimum pulse width the filter recognizes.

dig_edge_dig_fltr_timebase_rate
float – Specifies in hertz the rate of the digital filter timebase. NI-DAQmx uses this value to compute settings for the filter.

dig_edge_dig_fltr_timebase_src
str – Specifies the terminal of the signal to use as the timebase of the digital filter.

dig_edge_dig_sync_enable
bool – Specifies whether to synchronize recognition of transitions in the signal to the internal timebase of the device.

dig_edge_edge
nidaqmx.constants.Edge – Specifies on what edge of a digital pulse the Reference Trigger occurs.

dig_edge_src
str – Specifies the name of a terminal where there is a digital signal to use as the source of the Reference Trigger.

dig_pattern_pattern
str – Specifies the digital pattern that must be met for the Reference Trigger to occur.

dig_pattern_src
nidaqmx.system.physical_channel.PhysicalChannel – Specifies the physical channels to use for pattern matching. The order of the physical channels determines the order of the pattern. If a port is included, the order of the physical channels within the port is in ascending order.

dig_pattern_trig_when
nidaqmx.constants.DigitalPatternCondition – Specifies whether the Reference Trigger occurs when the physical channels specified with **dig_pattern_src** match or differ from the digital pattern specified with **dig_pattern_pattern**.

disable_ref_trig()
Disables reference triggering for the measurement.

pretrig_samples
int – Specifies the minimum number of pretrigger samples to acquire from each channel before recognizing the reference trigger. Post-trigger samples per channel are equal to **samp_quant_samp_per_chan** minus the number of pretrigger samples per channel.

term
str – Indicates the name of the internal Reference Trigger terminal for the task. This property does not return the name of the trigger source terminal.

trig_type
`nidaqmx.constants.TriggerType` – Specifies the type of trigger to use to mark a reference point for the measurement.

nidaqmx.task.start_trigger

class `nidaqmx._task_modules.triggering.start_trigger.`**StartTrigger**(*task_handle*)
Bases: `object`

Represents the start trigger configurations for a DAQmx task.

anlg_edge_coupling
`nidaqmx.constants.Coupling` – Specifies the coupling for the source signal of the trigger if the source is a terminal rather than a virtual channel.

anlg_edge_dig_fltr_enable
bool – Specifies whether to apply a digital filter to the digital output of the analog triggering circuitry (the Analog Comparison Event). When enabled, the analog signal must stay above or below the trigger level for the minimum pulse width before being recognized. Use filtering for noisy trigger signals that transition in and out of the hysteresis window rapidly.

anlg_edge_dig_fltr_min_pulse_width
float – Specifies in seconds the minimum pulse width the filter recognizes.

anlg_edge_dig_fltr_timebase_rate
float – Specifies in hertz the rate of the digital filter timebase. NI-DAQmx uses this value to compute settings for the filter.

anlg_edge_dig_fltr_timebase_src
str – Specifies the terminal of the signal to use as the timebase of the digital filter.

anlg_edge_dig_sync_enable
bool – Specifies whether to synchronize recognition of transitions in the signal to the internal timebase of the device.

anlg_edge_hyst
float – Specifies a hysteresis level in the units of the measurement or generation. If **anlg_edge_slope** is **Slope1.RISING**, the trigger does not deassert until the source signal passes below **anlg_edge_lvl** minus the hysteresis. If **anlg_edge_slope** is **Slope1.FALLING**, the trigger does not deassert until the source signal passes above **anlg_edge_lvl** plus the hysteresis. Hysteresis is always enabled. Set this property to a non-zero value to use hysteresis.

anlg_edge_lvl
float – Specifies at what threshold in the units of the measurement or generation to start acquiring or generating samples. Use **anlg_edge_slope** to specify on which slope to trigger on this threshold.

anlg_edge_slope
`nidaqmx.constants.Slope` – Specifies on which slope of the trigger signal to start acquiring or generating samples.

anlg_edge_src
str – Specifies the name of a virtual channel or terminal where there is an analog signal to use as the source of the Start Trigger.

anlg_win_btm
float – Specifies the lower limit of the window. Specify this value in the units of the measurement or generation.

anlg_win_coupling
nidaqmx.constants.Coupling – Specifies the coupling for the source signal of the trigger if the source is a terminal rather than a virtual channel.

anlg_win_dig_fltr_enable
bool – Specifies whether to apply a digital filter to the digital output of the analog triggering circuitry (the Analog Comparison Event). When enabled, the analog signal must stay within the trigger window for the minimum pulse width before being recognized. Use filtering for noisy trigger signals that transition in and out of the window rapidly.

anlg_win_dig_fltr_min_pulse_width
float – Specifies in seconds the minimum pulse width the filter recognizes.

anlg_win_dig_fltr_timebase_rate
float – Specifies in hertz the rate of the digital filter timebase. NI-DAQmx uses this value to compute settings for the filter.

anlg_win_dig_fltr_timebase_src
str – Specifies the terminal of the signal to use as the timebase of the digital filter.

anlg_win_dig_sync_enable
bool – Specifies whether to synchronize recognition of transitions in the signal to the internal timebase of the device.

anlg_win_src
str – Specifies the name of a virtual channel or terminal where there is an analog signal to use as the source of the Start Trigger.

anlg_win_top
float – Specifies the upper limit of the window. Specify this value in the units of the measurement or generation.

anlg_win_trig_when
nidaqmx.constants.WindowTriggerCondition1 – Specifies whether the task starts acquiring or generating samples when the signal enters or leaves the window you specify with **anlg_win_btm** and **anlg_win_top**.

cfg_anlg_edge_start_trig(*trigger_source=u'', trigger_slope=<Slope.RISING: 10280>, trigger_level=0.0*)
Configures the task to start acquiring or generating samples when an analog signal crosses the level you specify.

Parameters

- **trigger_source** (*Optional[str]*) – Is the name of a virtual channel or terminal where there is an analog signal to use as the source of the trigger.
- **trigger_slope** (*Optional[nidaqmx.constants.Slope]*) – Specifies on which slope of the signal to start acquiring or generating samples when the signal crosses **trigger_level**.
- **trigger_level** (*Optional[float]*) – Specifies at what threshold to start acquiring or generating samples. Specify this value in the units of the measurement or generation. Use **trigger_slope** to specify on which slope to trigger at this threshold.

cfg_anlg_window_start_trig(*window_top*, *window_bottom*, *trigger_source=u''*, *trigger_when=<WindowTriggerCondition1.ENTERING_WINDOW: 10163>*)

Configures the task to start acquiring or generating samples when an analog signal enters or leaves a range you specify.

Parameters

- **window_top** (*float*) – Is the upper limit of the window. Specify this value in the units of the measurement or generation.
- **window_bottom** (*float*) – Is the lower limit of the window. Specify this value in the units of the measurement or generation.
- **trigger_source** (*Optional[str]*) – Is the name of a virtual channel or terminal where there is an analog signal to use as the source of the trigger.
- **trigger_when** (*Optional[nidaqmx.constants.WindowTriggerCondition1]*) – Specifies whether the task starts measuring or generating samples when the signal enters the window or when it leaves the window. Use **window_bottom** and **window_top** to specify the limits of the window.

cfg_dig_edge_start_trig(*trigger_source*, *trigger_edge=<Edge.RISING: 10280>*)

Configures the task to start acquiring or generating samples on a rising or falling edge of a digital signal.

Parameters

- **trigger_source** (*str*) – Specifies the name of a terminal where there is a digital signal to use as the source of the trigger.
- **trigger_edge** (*Optional[nidaqmx.constants.Edge]*) – Specifies on which edge of the digital signal to start acquiring or generating samples.

cfg_dig_pattern_start_trig(*trigger_source*, *trigger_pattern*, *trigger_when=<DigitalPatternCondition.PATTERN_MATCHES: 10254>*)

Configures a task to start acquiring or generating samples when a digital pattern is matched.

Parameters

- **trigger_source** (*str*) – Specifies the physical channels to use for pattern matching. The order of the physical channels determines the order of the pattern. If a port is included, the order of the physical channels within the port is in ascending order.
- **trigger_pattern** (*str*) – Specifies the digital pattern that must be met for the trigger to occur.
- **trigger_when** (*Optional[nidaqmx.constants.DigitalPatternCondition]*) – Specifies the condition under which the trigger occurs.

delay

float – Specifies an amount of time to wait after the Start Trigger is received before acquiring or generating the first sample. This value is in the units you specify with **delay_units**.

delay_units

nidaqmx.constants.DigitalWidthUnits – Specifies the units of **delay**.

dig_edge_dig_fltr_enable

bool – Specifies whether to apply a digital filter to the trigger signal.

dig_edge_dig_fltr_min_pulse_width

float – Specifies in seconds the minimum pulse width the filter recognizes.

dig_edge_dig_fltr_timebase_rate
float – Specifies in hertz the rate of the pulse width filter timebase. NI-DAQmx uses this value to compute settings for the filter.

dig_edge_dig_fltr_timebase_src
str – Specifies the input terminal of the signal to use as the timebase of the pulse width filter.

dig_edge_dig_sync_enable
bool – Specifies whether to synchronize recognition of transitions in the signal to the internal timebase of the device. If you set this property to True, the device does not recognize and act upon the trigger until the next pulse of the internal timebase.

dig_edge_edge
nidaqmx.constants.Edge – Specifies on which edge of a digital pulse to start acquiring or generating samples.

dig_edge_src
str – Specifies the name of a terminal where there is a digital signal to use as the source of the Start Trigger.

dig_pattern_pattern
str – Specifies the digital pattern that must be met for the Start Trigger to occur.

dig_pattern_src
nidaqmx.system.physical_channel.PhysicalChannel – Specifies the physical channels to use for pattern matching. The order of the physical channels determines the order of the pattern. If a port is included, the order of the physical channels within the port is in ascending order.

dig_pattern_trig_when
nidaqmx.constants.DigitalPatternCondition – Specifies whether the Start Trigger occurs when the physical channels specified with **dig_pattern_src** match or differ from the digital pattern specified with **dig_pattern_pattern**.

disable_start_trig()
Configures the task to start acquiring or generating samples immediately upon starting the task.

retriggerable
bool – Specifies whether a finite task resets and waits for another Start Trigger after the task completes. When you set this property to True, the device performs a finite acquisition or generation each time the Start Trigger occurs until the task stops. The device ignores a trigger if it is in the process of acquiring or generating signals.

term
str – Indicates the name of the internal Start Trigger terminal for the task. This property does not return the name of the trigger source terminal.

trig_type
nidaqmx.constants.TriggerType – Specifies the type of trigger to use to start a task.

nidaqmx.types

class nidaqmx.types.**AOExpirationState**(*physical_channel*, *expiration_state*, *output_type*)
Bases: tuple

expiration_state
Alias for field number 1

output_type
Alias for field number 2

physical_channel
Alias for field number 0

class nidaqmx.types.**AOPowerUpState**(*physical_channel*, *power_up_state*, *channel_type*)
Bases: tuple

channel_type
Alias for field number 2

physical_channel
Alias for field number 0

power_up_state
Alias for field number 1

class nidaqmx.types.**CDAQSyncConnection**(*output_port*, *input_port*)
Bases: tuple

input_port
Alias for field number 1

output_port
Alias for field number 0

class nidaqmx.types.**COExpirationState**(*physical_channel*, *expiration_state*)
Bases: tuple

expiration_state
Alias for field number 1

physical_channel
Alias for field number 0

class nidaqmx.types.**CtrFreq**(*freq*, *duty_cycle*)
Bases: tuple

duty_cycle
Alias for field number 1

freq
Alias for field number 0

class nidaqmx.types.**CtrTick**(*high_tick*, *low_tick*)
Bases: tuple

high_tick
Alias for field number 0

low_tick
Alias for field number 1

class nidaqmx.types.**CtrTime**(*high_time*, *low_time*)
Bases: tuple

high_time
Alias for field number 0

low_time
Alias for field number 1

class nidaqmx.types.**DOExpirationState**(*physical_channel*, *expiration_state*)
Bases: tuple

expiration_state
Alias for field number 1

physical_channel
Alias for field number 0

class nidaqmx.types.**DOPowerUpState**(*physical_channel*, *power_up_state*)
Bases: tuple

physical_channel
Alias for field number 0

power_up_state
Alias for field number 1

class nidaqmx.types.**DOResistorPowerUpState**(*physical_channel*, *power_up_state*)
Bases: tuple

physical_channel
Alias for field number 0

power_up_state
Alias for field number 1

nidaqmx.utils

nidaqmx.utils.**flatten_channel_string**(*channel_names*)
Converts a list of channel names to a comma-delimited list of names.

You can use this method to convert a list of physical or virtual channel names to a single string prior to using the DAQmx Create Channel methods or instantiating a DAQmx Task object.

Parameters **channel_names** (*List[str]*) – The list of physical or virtual channel names.

Returns The resulting comma-delimited list of physical or virtual channel names.

Return type str

nidaqmx.utils.**unflatten_channel_string**(*channel_names*)
Converts a comma-delimited list of channel names to a list of names.

You can use this method to convert a comma-delimited list or range of physical or virtual channels into a list of physical or virtual channel names.

Parameters **channel_names** (*str*) – The list or range of physical or virtual channels.

Returns The list of physical or virtual channel names. Each element of the list contains a single channel.

Return type List[str]

CHAPTER 9

Indices and Tables

- genindex
- modindex

Python Module Index

n

Index

Symbols

A

B

C

D

E

F

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z