daqmx

Jason R. Jones

Nov 03, 2022

CONTENTS

CHAPTER ONE

INSTALLATION

1.1 From PIP

To install from pip:

```
python -m pip install daqmx
```

1.2 From `setup.py`

To install from the github repository:

```
$>git clone https://github.com/slightlynybbled/daqmx & python setup.py install
```

CHAPTER

TWO

GETTING STARTED

2.1 Install the Drivers

Download an install the latest published NI DAQmx software from the National Instruments site. This should install all necessary hardware drivers into your OS.

2.2 Attach Hardware

Attach your device to your PC. Ensure tha the DAQmx software on your PC has detected the hardware and assigned it a valid name (i.e. "Dev3").

2.3 Acquire Hardware

From within your program, acquire the hardware:

```
daq = NIDAQmxInstrument()
```

You may also specify the DAQmx-assigned name in order to acquire a specific instrument:

```
daq = NIDAQmxInstrument(device_name='Dev3')
```

Hardware acquisition from the model number is also supported:

```
daq = NIDAQmxInstrument(model_number='USB-6001')
```

Finally, you may specify the serial number:

```
daq = NIDAQmxInstrument(serial_number='1B5D996')
```

2.4 Sample Analog Input

Read analog input 0, print it to the console:

```
print(f'daq.ai0.value: {daq.ai0.value:.3f}V')
```

2.5 Capture Analog Input

Take multiple analog samples with more control over the hardware:

```
values = daq.ai1.capture(
    sample_count=10, rate=100,
    max_voltage=10.0, min_voltage=-10.0,
    mode='differential', timeout=3.0
)
print(values)
```

Note that the `values` variable contains a numpy array which represents all samples acquired during the capture process.

CHAPTER

THREE

API

3.1 Instrument

The `NIDAQmxInstrument` class is the primary method through which most users will acquire hardware.

class daqmx.**NIDAQmxInstrument**(*device_name: str = None*, *serial_number: (<class 'str'>, <class 'int'>) = None*, *model_number: str = None*, *loglevel=20*)

This class will create the tasks and coordinate with the hardware in order to achieve a particular end on an input or output of the DAQ module.

The methods within this object utilize the concepts found in the NI-DAQmx Help menu, such as channels and tasks.

Parameters

- **device_name** – the device name, often formatted like *Dev3*
- **serial_number** – the serial number as a hexadecimal value (this is usually what is printed on the label)
- **model_number** – the model number as printed on the label

property sn

Returns the device serial number

Returns
the device serial number

property model

Returns the device model number

Returns
the device model number

property outputs

Returns a list of outputs associated with the device.

Returns
a list of outputs associated with the device.

property inputs

Returns a list of inputs associated with the device

Returns
a list of inputs associated with the device

3.2 Port

The `Port` class is the class which implements port writes and reads. It may be used directly or through the instrument.

class daqmx.**Port**(*device: str*, *port: str*)

Represents the port object as defined by DAQmx.

Parameters

- **device** – the device string as defined by DAQmx (i.e. 'Dev3')
- **port** – the port name as defined by DAQmx (i.e. 'port2')

property lines

Lists all of the lines attached to the port

Returns

a list of line names

3.3 Analog Input

The `AnalogInput` class is the class which implements most of the analog input functionality. It may be used directly or through the instrument.

class daqmx.**AnalogInput**(*device: str*, *analog_input: str = None*, *sample_count: int = 1000*, *rate: (<class 'int'>, <class 'float'>) = 1000.0*, *max_voltage: (<class 'int'>, <class 'float'>) = 5.0*, *min_voltage: (<class 'int'>, <class 'float'>) = 0.0*, *mode: str = 'differential'*, *timeout: (<class 'int'>, <class 'float'>) = None*)

Represents an analog input on the DAQmx device.

Parameters

- **device** – the device string assigned by DAQmx (i.e. 'Dev3)
- **analog_input** – the analog input name assigned by DAQmx (i.e. "ao0")
- **sample_count** – the number of samples to take
- **rate** – the frequency at which to sample the input
- **max_voltage** – the maximum expected voltage
- **min_voltage** – the minimum expected voltage
- **mode** – the mode; valid values: differential, pseudo-differential, / singled-ended referenced, singled-ended non-referenced
- **timeout** – the time at which an error will occur if no response / from the instrument is received.

property value

Return a single sample of the analog input

Returns

a floating-point value representing the voltage `sample`(*analog_input: Optional[str] = None*)

Return a single sample of the analog input

Returns
a floating-point value representing the voltage

`capture`(*analog_input: str = None*, *sample_count: int = None*, *rate: (<class 'int'>, <class 'float'>) = None*, *max_voltage: (<class 'int'>, <class 'float'>) = None*, *min_voltage: (<class 'int'>, <class 'float'>) = None*, *mode: str = None*, *timeout: (<class 'int'>, <class 'float'>) = None*)

Will capture <sample_count> samples at <rate>Hz in the <mode> mode.

Parameters

- `analog_input` – the analog input name assigned by DAQmx (i.e. "ao0")
- `sample_count` – the number of samples to take
- `rate` – the frequency at which to sample the input
- `max_voltage` – the maximum expected voltage
- `min_voltage` – the minimum expected voltage
- `mode` – the mode; valid values: differential, pseudo-differential, / singled-ended referenced, singled-ended non-referenced
- `timeout` – the time at which an error will occur if no response / from the instrument is received.

Returns
a numpy array containing all resulting values

`find_dominant_frequency`(*analog_input: str = None*, *sample_count: int = None*, *rate: (<class 'int'>, <class 'float'>) = None*, *max_voltage: (<class 'int'>, <class 'float'>) = None*, *min_voltage: (<class 'int'>, <class 'float'>) = None*, *mode: str = None*, *timeout: (<class 'int'>, <class 'float'>) = None*)

Acquires the fundamental frequency observed within the samples

Parameters

- `analog_input` – the NI analog input designation (i.e. 'ai0')
- `sample_count` – the number of samples to acquired
- `rate` – the sample rate in Hz :param max_voltage: the maximum voltage possible
- `min_voltage` – the minimum voltage range
- `mode` – the voltage mode of operation; choices: 'differential', 'pseudo-differential', 'single-ended referenced', 'single-ended non-referenced'
- `timeout` – the time at which the function should return if this time has elapsed; set to -1 to make infinite (default)

Returns
the frequency found to be at the highest amplitude; this is often the fundamental frequency in many domains

CHAPTER

FOUR

EXAMPLES

4.1 Hardware Acquisition

```
from daqmx import NIDAQmxInstrument

# first, we allocate the hardware using the automatic hardware allocation
# available to the instrument; this is safe when there is only one NIDAQmx
# instrument, but you may wish to specify a serial number or model number
# for a safer experience
daq = NIDAQmxInstrument()
print(daq)  # printing the instrument will result in showing the
            # device name, model number, and serial number

# you may also want to specify a particular device name, as assigned by
# the DAQmx interface; this is usually something like `Dev3`, although
# I believe that it may be renamed through the DAQmx interface
daq = NIDAQmxInstrument(device_name='Dev3')
print(daq)

# you might also simply wish to specify the model number to acquire
daq = NIDAQmxInstrument(model_number='USB-6001')
print(daq)

# further, you may wish to specify a particular serial number
daq = NIDAQmxInstrument(serial_number='1B5D996')  # <-- the serial number, as
print(daq)                                        # read off the back of the
                                                  # device in a hex format, is
                                                  # entered as a string

daq = NIDAQmxInstrument(serial_number=28694934)  # <-- this is the same device,
print(daq)                                       # entering the serial number
                                                 # as an integer instead of as
                                                 # a hex value
```

4.2 Read Digital Input

```
from daqmx import NIDAQmxInstrument

# tested with NI USB-6001
# which has the following digital inputs:
#  - port0/line0 through line7
#  - port1/line0 through line3
#  - port2/line0

# first, we allocate the hardware using the automatic hardware
# allocation available to the instrument; this is safe when there
# is only one NIDAQmx instrument, but you may wish to specify a
# serial number or model number for a safer experience
daq = NIDAQmxInstrument()

print(daq)

# read the True or False state on the digital outputs
# by reading the `portX` and `lineX` attributes; you
# may wish to use the 5V output available to force the
# pin to a state
print(daq.port0.line0)
print(daq.port0.line1)

# !!! IMPORTANT !!!!
# if you set the value of a port/line, the hardware will
# be changed to an output; however if you read the value
# using the similar syntax, the hardware will be changed
# to an input!
```

4.3 Set/Clear Digital Output

```
from daqmx import NIDAQmxInstrument

# tested with NI USB-6001
# which has the following digital outputs:
#  - port0/line0 through line7
#  - port1/line0 through line3
#  - port2/line0

# first, we allocate the hardware using the automatic hardware
# allocation available to the instrument; this is safe when there
# is only one NIDAQmx instrument, but you may wish to specify a
# serial number or model number for a safer experience
daq = NIDAQmxInstrument()

print(daq)

# set the True or False state on the digital outputs by setting the
```

(continues on next page)

(continued from previous page)

```
# `portX` and `lineX` attributes;
# use your multimeter to verify!
daq.port0.line0 = False
daq.port0.line1 = True

# you may wish to acquire the port separately
# and manipulate it directly
port = daq.port1
port.line0 = True

# if you try to set an output that doesn't exist, you
# should see errors (uncomment to see)
#port.line5 = True

# !!! IMPORTANT !!!!
# if you set the value of a port/line, the hardware will
# be changed to an output; however if you read the value
# using the similar syntax, the hardware will be changed
# to an input!
```

4.4 Read Analog Input

```
from daqmx import NIDAQmxInstrument, AnalogInput

# tested with NI USB-6001
# which has the following analog inputs:
#   - ai0
#   - ai1
#   - ai2
#   - ai3

# first, we allocate the hardware using the automatic hardware
# allocation available to the instrument; this is safe when there
# is only one NIDAQmx instrument, but you may wish to specify a
# serial number or model number for a safer experience
daq = NIDAQmxInstrument()

print(daq)

# the easiest way to get a single sample is to select the analog input
# attribute on the daq and interrogate its `value` attribute
print(f'daq.ai0.value: {daq.ai0.value:.3f}V')
print(f'daq.ai1.value: {daq.ai1.value:.3f}V')
print(f'daq.ai2.value: {daq.ai2.value:.3f}V')
print(f'daq.ai3.value: {daq.ai3.value:.3f}V')

# you will start throwing errors if you interrogate
# inputs that don't exist on the device (uncomment to see!)
#print(f'daq.ai4.value: {daq.ai4.value:.3f}V')
```

(continues on next page)

(continued from previous page)

```
# for more nuanced control over the analog
# input, we could use the `capture` method
values = daq.ai1.capture(
    sample_count=10, rate=100,
    max_voltage=10.0, min_voltage=-10.0,
    mode='differential', timeout=3.0
)
print(f'values: {values} V')

# note that the values come back as type `numpy.ndarray`
print(f'type(values): {type(values)}')

# if you already know your device name, you might be
# happier going straight to the `AnalogInput` constructor:
ai0 = AnalogInput(device='Dev3', analog_input='ai0')

# we can do anything that we could have
# done previously with the daq.aiX
print(f'ai0.value: {ai0.value:.3f}V')
```

4.5 Write Analog Output

```
from daqmx import NIDAQmxInstrument

# tested with NI USB-6001
# which has the following analog outputs:
#   - ao0
#   - ao1

# first, we allocate the hardware using the automatic hardware
# allocation available to the instrument; this is safe when there
# is only one NIDAQmx instrument, but you may wish to specify a
# serial number or model number for a safer experience
daq = NIDAQmxInstrument()

print(daq)

# set the voltage on the analog outputs by setting the
# attribute `aoX`; use your multimeter to verify!
daq.ao0 = 1.02
daq.ao1 = 2.04

# once the attribute is set, you should be able to read
# it on the daq; if the attribute hasn't been set, this
# will result in an error (for now)
print(f'ao0: {daq.ao0:.2f}V')

# if you set an attribute on an output that doesn't exist,
# then the attribute will be set on the object, but nothing
# will happen!  be sure that you are setting valid attributes
```

(continues on next page)

(continued from previous page)

```
daq.ao2 = 3.0
print(daq.ao2)  # <-- there is no "ao2"!!!
```

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX