dwfpy

Release 1.1

Marius Greuel

Jul 20, 2022

INTRODUCTION

CHAPTER

ONE

OVERVIEW

DwfPy is a Python package that allows you to access Digilent WaveForms devices via Python. It provides a low-level API with complete access to the Digilent WaveForms API, and also a simple but powerful high-level API, which allows you to configure WaveForms devices with a single statement.

For instance, to output a 1kHz sine-wave on a **Analog Discovery 2**, you can simply write:

```
import dwfpy as dwf

with dwf.AnalogDiscovery2() as device:
    print('Generating a 1kHz sine wave on WaveGen channel 1...')
    device.analog_output['ch1'].setup('sine', frequency=1e3, amplitude=1, start=True)
    input('Press Enter key to exit.')
```

CHAPTER

TWO

FEATURES

- Pythonic abstraction of Digilent Waveforms API.
- Low-level API with complete access to the Digilent Waveforms API.
- Powerful high-level API that supports one-line configuration statements.
- Supports all sub-modules, such as oscilloscope, arbitrary waveform generator, logic analyzer, pattern generator, digital I/O, and power supplies.
- Works with all WaveForms devices, such as the Analog Discovery 2 or the Digital Discovery.

CHAPTER

THREE

INSTALLING DWFPY

You can install the dwfpy package from PyPI using pip:

```
pip install dwfpy
```

In order to use the **DwfPy** package, you need **Python 3.6** or higher.

As **DwfPy** builds on top of the WaveForms API, you need to install the WaveForms software, which includes the required runtime components to access the WaveForms devices.

The source code for the **DwfPy** package can be found at GitHub at https://github.com/mariusgreuel/dwfpy.

CHAPTER

FOUR

DOCUMENTATION

You can find the **DwfPy** user's guide at https://dwfpy.readthedocs.io/.

Detailed information about the Digilent Waveforms API is available from the WaveForms.

CHAPTER

FIVE

EXAMPLES

You can find Python examples using **DwfPy** in the dwfpy GitHub repository at https://github.com/mariusgreuel/dwfpy/tree/main/examples.

CHAPTER

SIX

GETTING HELP

For issues with **DwfPy**, please visit the dwfpy GitHub issue tracker.

CHAPTER

SEVEN

WORKING WITH DEVICES

7.1 Creating a Device Instance

To find and create an instance of a Digilent Waveforms device, you simply create an instance of the *Device* class. You should use the Python with statement to ensure that your device is closed automatically on exit, or when an exception occurs:

```
import dwfpy as dwf

with dwf.Device() as device:
    print(f'Found device: {device.name} ({device.serial_number})')
```

7.2 Using a Specific Device Instance

If you know the kind of device you have connected to your PC, you can use a specialized class, such as *AnalogDiscovery2* or *DigitalDiscovery*, which provides additional functions that are specific to that device:

```
import dwfpy as dwf

with dwf.AnalogDiscovery2() as device:
    print(f'Found an Analog Discovery 2: {device.user_name} ({device.serial_number})')
```

You can find the available device classes in the *device* module.

7.3 Filtering Devices

If you have multiple devices installed, you can pass additional filter parameters, while creating the device instance. For example, you can pass a **serial number** to pick a specific device:

```
import dwfpy as dwf

with dwf.Device(serial_number='123456ABCDEF') as device:
    print(f'Found a device with matching serial number: {device.user_name} ({device.
↪serial_number})')
```

You can find additional filter parameters in the class constructor method *Device*.

7.4 Enumerating Devices

If you want to enumerate all present devices and get the devices properties, you can use the *Device.enumerate()* function to enumerate devices:

```
import dwfpy as dwf

for device in dwf.Device.enumerate():
    print(f'Found device: {device.name} {device.serial_number}')
```

For a complete example, see examples/device_enumeration.py and examples/device_info.py

CHAPTER

EIGHT

USING THE OSCILLOSCOPE

If your device has an oscilloscope, you can access it via the property *analog_input*. Individual channels can be accessed via a zero-based index, or a label, such as 'ch1'.

There are various high-level functions that you can use to control the oscilloscope:

Channel Setup

- `analog_input[ch].setup()` - Sets up the channel for data acquisition.

Data Acquisition

- `analog_input[ch].get_sample()` - Gets the last ADC conversion sample.
- *analog_input.single()* - Starts a single data acquisition.
- *analog_input.record()* - Starts a data recording.

Triggering

- *analog_input.setup_edge_trigger()* - Trigger upon a certain voltage level in the positive or negative slope of the waveform.
- *analog_input.setup_pulse_trigger()* - Trigger upon a positive or negative pulse width when measured at a certain voltage level.
- *analog_input.setup_transition_trigger()* - Sets up a transition trigger.
- *analog_input.setup_window_trigger()* - Trigger upon a signal entering or exiting a window at certain voltage thresholds.

8.1 Setting Up Channels

You can use the `analog_input[ch].setup()` function on a respective analog input channel to setup a channel. You can specify the channel voltage range, offset voltage, and more.

```
import dwfpy as dwf

with dwf.Device() as device:
    scope = device.analog_input
    scope[0].setup(range=5.0, offset=2.0)
```

8.2 Getting the Current ADC Sample

You can use the `analog_input[ch].get_sample()` function on a respective analog input channel to get the current ADC reading.

Note: You need to call the *analog_input.read_status()* function before calling `get_sample()` to read a sample from the device.

```
import dwfpy as dwf

with dwf.Device() as device:
    scope = device.analog_input
    scope[0].setup(range=50.0)
    scope.configure()
    scope.read_status()
    print(f'CH1: {scope[0].get_sample()}V')
```

For a complete example, see examples/analog_in_sample.py.

8.3 Single Data Acquisition

You can use the *single()* function on the analog input unit to perform a single-shot data acquisition. To configure the oscilloscope, pass the parameter `configure=True`. To start the acquisition immediately and wait for the acquisition to finish, pass the parameter `start=True`.

```
import dwfpy as dwf

with dwf.Device() as device:
    scope = device.analog_input
    scope[0].setup(range=5.0)
    scope.single(sample_rate=1e6, buffer_size=4096, configure=True, start=True)
    samples = scope[0].get_data()
    print(samples)
```

For a complete example, see examples/analog_in_single.py.

8.4 Recording Samples

You can use the *record()* function on the analog input unit to perform data recording. To configure the oscilloscope, pass the parameter `configure=True`. To start the acquisition immediately, pass the parameter `start=True`.

```
import dwfpy as dwf

with dwf.Device() as device:
    scope = device.analog_input
    scope[0].setup(range=5.0)
    recorder = scope.record(sample_rate=1e6, sample_count=1e6, configure=True,␣
↪start=True)
```

(continues on next page)

(continued from previous page)

```
    samples = recorder.channels[0].data_samples
    print(samples)
```

For a complete example, see examples/analog_in_record.py.

8.5 Setting Up an Edge Trigger

You can use the *setup_edge_trigger()* function to trigger the oscilloscope on a positive or negative slope of the waveform.

```
import dwfpy as dwf

with dwf.Device() as device:
    scope[0].setup(range=5.0)
    scope.setup_edge_trigger(mode='normal', channel=0, slope='rising', level=0.5,␣
↪hysteresis=0.01)
    scope.single(sample_rate=1e6, buffer_size=4096, configure=True, start=True)
    samples = scope[0].get_data()
```

For a complete example, see examples/analog_in_single.py.

CHAPTER

NINE

USING THE WAVEFORM GENERATOR

If your device has an arbitrary waveform generator, you can access it via the property *analog_output*. Individual channels can be accessed via a zero-based index, or a label, such as 'ch1'.

There are various high-level functions that you can use to control the waveform generator:

Channel Setup

- `analog_output[ch].setup()` - Sets up a new carrier waveform.
- `analog_output[ch].setup_am()` - Applies an AM modulation to a waveform.
- `analog_output[ch].setup_fm()` - Applies an FM modulation to a waveform (sweep).

9.1 Setting up a new Waveform

You can use the `setup()` function to create a new waveform. You can specify the generator function, frequency, amplitude, offset, symmetry, and phase. By default, the carrier node is enabled via the parameter `enabled=True`. To start the channel immediately, pass the parameter `start=True`.

```
import dwfpy as dwf

with dwf.Device() as device:
    wavegen = device.analog_output
    wavegen[0].setup(function='sine', frequency=1e3, amplitude=1.0, start=True)
```

For a complete example, see examples/analog_out_sine.py.

9.2 Setting up an Amplitude Modulation

You can use the `setup_am()` function to modulate the amplitude. You can specify the generator function, frequency, amplitude, offset, symmetry, and phase, similar to the `setup()` function. By default, the AM node is enabled via the parameter `enabled=True`. To start the channel immediately, pass the parameter `start=True`.

```
import dwfpy as dwf

with dwf.Device() as device:
    wavegen = device.analog_output
    wavegen[0].setup(function='sine', frequency=1e3, amplitude=1.0)
    wavegen[0].setup_am(function='triangle', frequency=10, amplitude=50, start=True)
```

9.3 Setting up a Frequency Modulation

You can use the `setup_fm()` function to modulate the frequency. You can specify the generator function, frequency, amplitude, offset, symmetry, and phase, similar to the `setup()` function. By default, the FM node is enabled via the parameter `enabled=True`. To start the channel immediately, pass the parameter `start=True`.

```
import dwfpy as dwf

with dwf.Device() as device:
    wavegen = device.analog_output
    wavegen[0].setup(function='sine', frequency=1e3, amplitude=1.0)
    wavegen[0].setup_fm(function='sine', frequency=10, amplitude=10, start=True)
```

CHAPTER

TEN

USING THE LOGIC ANALYZER

If your device has a logic analyzer, you can access it via the property *digital_input*. Individual channels can be accessed via a zero-based index, or a label, such as 'ch1'.

There are various high-level functions that you can use to control the logic analyzer:

Acquisition

- *digital_input.single()* - Starts a single data acquisition.
- *digital_input.record()* - Starts a data recording.

Triggering

- *digital_input.setup_trigger()* - Sets up the trigger condition.
- *digital_input.setup_edge_trigger()* - Sets up an edge trigger.
- *digital_input.setup_level_trigger()* - Sets up a level trigger.
- *digital_input.setup_glitch_trigger()* - Sets up a glitch trigger.
- *digital_input.setup_timeout_trigger()* - Sets up a timeout trigger.
- *digital_input.setup_more_trigger()* - Sets up a more trigger.
- *digital_input.setup_length_trigger()* - Sets up a length trigger.
- *digital_input.setup_counter_trigger()* - Sets up a counter trigger.

Channel specific Triggering

- `digital_input[ch].setup_trigger()` - Sets up the trigger condition for this channel.
- `digital_input[ch].setup_reset_trigger()` - Sets up the trigger reset condition for this channel.

10.1 Single Data Acquisition

You can use the *single()* function on the digital input unit to perform a single-shot data acquisition. To configure the logic analyzer, pass the parameter `configure=True`. To start the acquisition immediately and wait for the acquisition to finish, pass the parameter `start=True`.

```
import dwfpy as dwf

with dwf.Device() as device:
    logic = device.digital_input
    samples = logic.single(sample_rate=1e6, buffer_size=4096, configure=True, start=True)
    print(samples)
```

For a complete example, see examples/digital_in_acquisition.py.

10.2 Recording Samples

You can use the *record()* function on the digital input unit to perform data recording. To configure the logic analyzer, pass the parameter `configure=True`. To start the acquisition immediately, pass the parameter `start=True`.

```
import dwfpy as dwf

with dwf.Device() as device:
    logic = device.digital_input
    recorder = logic.record(sample_rate=1e6, sample_count=1e6, configure=True,␣
↪start=True)
    samples = recorder.data_samples
    print(samples)
```

For a complete example, see examples/digital_in_record.py.

For an examples that uses data compression, see examples/digital_in_record_compress.py.

CHAPTER

ELEVEN

USING THE PATTERN GENERATOR

If your device has a pattern generator, you can access it via the property *digital_output*. Individual channels can be accessed via a zero-based index, or a label, such as 'ch1'.

There are various functions that you can use to configure pins:

- `digital_output[ch].setup_constant()` - Sets up the channel as a constant output.
- `digital_output[ch].setup_clock()` - Sets up the channel as a clock output.
- `digital_output[ch].setup_pulse()` - Sets up the channel as a pulse output.
- `digital_output[ch].setup_random()` - Sets up the channel as a random output.
- `digital_output[ch].setup_custom()` - Sets up the channel with a custom output.

All setup function allow you to speficy the output mode ('push-pull', 'open-drain', 'open-source', or 'three-state') and the idle state ('init', 'low', 'high', or 'z').

11.1 Setting up a Constant Output

You can use the `setup_constant()` function to drive a channel with a constant output value. To start the channel immediately, pass the parameter `start=True`.

```
import dwfpy as dwf

with dwf.Device() as device:
    pattern = device.digital_output
    # Output a high level on pin DIO-0.
    pattern[0].setup_constant(value=True, start=True)
```

11.2 Setting up a Clock Signal

You can use the `setup_clock()` function to output a clock or PWM signal. You can specify the frequency, duty_cycle, phase, delay, and repetition count. By default, the channel is enabled via the parameter `enabled=True`. To start the channel immediately, pass the parameter `start=True`.

```
import dwfpy as dwf

with dwf.Device() as device:
    pattern = device.digital_output
```

(continues on next page)

(continued from previous page)

```
    # Output a 1kHz clock on pin DIO-0.
    pattern[0].setup_clock(frequency=1e3, start=True)
```

For a complete example, see examples/digital_out_clock.py.

11.3 Setting up a Random Signal

You can use the `setup_random()` function to output a clock or PWM signal. You can specify the rate, delay, and repetition count. By default, the channel is enabled via the parameter `enabled=True`. To start the channel immediately, pass the parameter `start=True`.

```
import dwfpy as dwf

with dwf.Device() as device:
    pattern = device.digital_output
    # Output a random pattern at a 1kHz rate on pin DIO-0.
    pattern[0].setup_random(rate=1e3, start=True)
```

For a complete example, see examples/digital_out_pins.py.

CHAPTER

TWELVE

USING THE DIGITAL I/O

If your device has Digital I/O, you can access it via the property *digital_io*. Individual channels can be accessed via a zero-based index, or a label, such as 'dio0', 'dio1', etc.

12.1 Reading from Inputs

By default, pins are configured as inputs.

Note: You need to call the *digital_io.read_status()* function before reading from `input_state` to read all input states from the device.

```
import dwfpy as dwf

with dwf.Device() as device:
    io = device.digital_io
    # Configure DIO-0 as input
    io[0].setup(enabled=False, configure=True)
    # Read all I/O pins
    io.read_status()
    dio0 = io[1].input_state
    print(f'DIO-0={dio0}')
```

12.2 Writing to Outputs

You can use the `setup()` function to setup a Digital I/O pin. To configure the Digital I/O pins, pass the parameter `configure=True`.

```
import dwfpy as dwf

with dwf.Device() as device:
    io = device.digital_io
    # Configure DIO-0 as output, and set the state to high
    io[0].setup(enabled=True, state=True)
    # Configure DIO-1 as output, and set the state to low
    io[1].setup(enabled=True, state=False, configure=True)
    # Output a one on DIO-1
    io[1].output_state = True
```

For a complete example, see examples/digital_in_acquisition.py.

CHAPTER

THIRTEEN

USING THE POWER SUPPLIES

Depending on the device you have, there may be one or more power supplies that can be accessed via the channels and nodes of the **Analog IO** module, which are documented in the WaveForms.

Note: In order to output a voltage, most power supplies have to be enabled both individually and via the **master enable switch**.

For instance, to output 3.3V on the positive power supply of an **Analog Discovery 2**, you can write:

```
import dwfpy as dwf

with dwf.Device() as device:
    # Set the voltage of the positive power supply to 3.3V.
    device.analog_io[0][1].value = 3.3

    # Enable the positive power supply.
    device.analog_io[0][0].value = True

    # Enable the master-enable switch.
    device.analog_io.master_enable = True
```

Instead of the generic **Device** class, you can also use the specialized device classes, which simplifies the access to the power supplies.

For instance, to output 3.3V on the positive power supply of an **Analog Discovery 2**, you can write:

```
import dwfpy as dwf

with dwf.AnalogDiscovery2() as device:
    # Set the positive power supply to 3.3V and enable it.
    device.supplies.positive.setup(voltage=3.3)
    device.supplies.master_enable = True
```

CHAPTER

FOURTEEN

LABELLING CHANNELS

Instead of indexing channels via a zero-based integer index, the channels can be accessed via a label. By default, analog channels are labelled *ch1*, *ch2*, etc. Digital channels are labelled *dio0*, *dio1*, etc.

You can rename the channel labels to you needs. For instance, if you have a clock signal on DIO pin 0 that you want to access via the label *clock*, you could write:

```
import dwfpy as dwf

with dwf.Device() as device:
    # Rename pin DIO-0 from 'dio0' to 'clock'
    device.digital_output[0].label = 'clock'

    # DIO-0 can now be referenced via the label 'clock'
    device.digital_output['clock'].setup_clock(frequency=1e3)
```

CHAPTER

FIFTEEN

DWFPY DESIGN

The design of the **dwfpy** package consists of three layers: Python bindings, low-level API, and high-level API.

15.1 Bindings

The **dwfpy** bindings give you raw access to the C API of the DWF DLL. In order to provide a natural look-and-feel while working in Python, the following API changes have been made:

- The naming of the DWF API has been adapted to match Python naming conventions.
- The DWF function API has been declared using the ctypes *CFUNCTYPE* prototypes, which allow you to pass Python types such as *int*, instead of ctypes types such as *c_int*.
- Error handling is performed automatically, i.e. the C functions return value is checked and an exception is thrown when appropriate.
- Function return values are directly returned as a scalar or tuple.

For instance, the DWF C API to get the minimum and maximum offset voltage of an analog-out channel is:

```
int FDwfAnalogOutNodeOffsetInfo(HDWF hdwf, int idxChannel, AnalogOutNode node, double␣
↪*pMin, double *pMax);
```

When working with the **dwfpy** bindings, it can be used as follows:

```
import dwfpy as dwf
import dwfpy.bindings as dwfb

with dwf.Device() as device:
    min_offset, max_offset = dwfb.dwf_analog_out_node_offset_info(device.handle, 0, dwfb.
↪ANALOG_OUT_NODE_CARRIER)
```

Typically, you do not work with the **dwfpy** bindings. Instead, use the low-level or high-level API.

15.2 Low-Level API

The **dwfpy** low-level API is designed to map the bindings to a more structured API.

- Functional units of the device have been grouped into objects.
- Channels and nodes are abstracted as collections, using a dict-like API.

For instance, the DWF C API to set the offset voltage of the analog_out channel 1 (arbitary waveform generator) is:

```
int FDwfAnalogOutNodeOffsetSet(HDWF hdwf, int idxChannel, AnalogOutNode node, double␣
↪vOffset);
```

When working with the **dwfpy** low-level API, it can be used as follows:

```
import dwfpy as dwf

with dwf.Device() as device:
    device.analog_output.channels[0].nodes[dwf.AnalogOutputNode.CARRIER].offset = 1.23
```

15.3 High-Level API

The dwfpy high-level API is designed to perform multiple common actions with a single line of Python code.

For instance, to start channel 1 of the arbitary waveform generator to output a sine-wave, using a frequency of 1kHz and an amplitude of 1Vpp, you would write:

```
import dwfpy as dwf

with dwf.Device() as device:
    device.analog_output['ch1'].setup(function='sine', frequency=1e3, amplitude=1,␣
↪start=True)
```

dwfpy	*dwfpy* is a package for accessing Digilent WaveForms devices.

CHAPTER

SIXTEEN

DWFPY

dwfpy is a package for accessing Digilent WaveForms devices.

16.1 Example

import dwfpy as dwf

with dwf.Device() as device:
    print(f'Found device: {device.name} ({device.serial_number})')

    # Generate sine wave device.analog_output[0].setup(function='sine', frequency=1000, amplitude=1.41, offset=1.41, configure=True)

16.2 Available subpackages

bindings
    Provides access to the raw C bindings of the DWF API.

Modules

dwfpy.analog_input	Analog Input module for Digilent WaveForms devices.
dwfpy.analog_io	Analog IO module for Digilent WaveForms devices.
dwfpy.analog_output	Analog Output module for Digilent WaveForms devices.
dwfpy.analog_recorder	Recorder for Analog Input data.
dwfpy.application	Support for Digilent WaveForms applications.
dwfpy.bindings	Python bindings for Digilent WaveForms API.
dwfpy.configuration	Configuration set for Digilent WaveForms devices.
dwfpy.constants	Constants used by Digilent WaveForms API.
dwfpy.device	Support for Digilent WaveForms devices.
dwfpy.device_info	Device information for Digilent WaveForms devices.
dwfpy.digital_input	Digital Input module for Digilent WaveForms devices.
dwfpy.digital_io	Digital IO module for Digilent WaveForms devices.
dwfpy.digital_output	Digital Output module for Digilent WaveForms devices.
dwfpy.digital_recorder	Recorder for Digital Input data.
dwfpy.exceptions	Exceptions for Digilent WaveForms.
dwfpy.helpers	Internal helper functions.
dwfpy.protocols	Protocols module for Digilent WaveForms devices.

16.3 dwfpy.analog_input

Analog Input module for Digilent WaveForms devices.

Classes

AnalogInput(device)	Analog Input module (Oscilloscope).
AnalogInputChannel(module, channel)	Represents an Analog Input channel.
AnalogInputTrigger(module)	Represents the trigger unit of an Analog Input module.

16.3.1 dwfpy.analog_input.AnalogInput

class AnalogInput(*device*)

Bases: `object`

Analog Input module (Oscilloscope).

Methods

configure	Configures the instrument and optionally starts the acquisition.
force_trigger	Force trigger of instrument.
read_status	Gets the acquisition state and optionally reads the data.
record	Starts a data recording.
reset	Resets and configures all instrument parameters to default values.
scan_screen	Starts a scan-screen data acquisition.
scan_shift	Starts a scan-shift data acquisition.
setup_acquisition	Sets up a new data acquisition.
setup_channel	Sets up a channel for data acquisition.
setup_edge_trigger	Trigger upon a certain voltage level in the positive or negative slope of the waveform.
setup_pulse_trigger	Trigger upon a positive or negative pulse width when measured at a certain voltage level.
setup_transition_trigger	Sets up a transition trigger.
setup_window_trigger	Trigger upon a signal entering or exiting a window at certain voltage thresholds.
single	Starts a single data acquisition.
wait_for_status	Waits for the specified acquisition state.

Attributes

acquisition_mode	Gets or sets the acquisition mode.
acquisition_mode_info	Gets the supported acquisition modes.
auto_triggered	Returns True if the acquisition is auto triggered.
buffer_size	Gets or sets the buffer size.
buffer_size_max	Gets the maximum supported buffer size.
buffer_size_min	Gets the minimum supported buffer size.
channels	Gets a collection of Analog Input channels.
counter	Gets or sets the current timeout in seconds.
counter_max	Gets the maximum count value.
counter_status	Gets the count, frequency and tick values.
device	Gets the device.
frequency	Gets or sets the configured sample frequency in Hz.
frequency_max	Gets the maximum supported sample frequency.
frequency_min	Gets the minimum supported sample frequency.
noise_buffer_size	Gets or sets the noise buffer size.
noise_buffer_size_max	Gets the maximum supported noise buffer size.
record_length	Gets or sets the record length in seconds.
record_status	Gets information about the recording process: Returns (available_samples, lost_samples, corrupted_samples)
remaining_samples	Gets the number of samples left in the acquisition.
time	Gets instrument trigger time information.
timeout_max	Gets the maximum timeout value.
trigger	Gets the trigger unit.
valid_samples	Gets the number of valid/acquired data samples.
write_index	Gets the buffer write pointer, which is needed in scan_screen acquisition mode to display the scan bar.

property acquisition_mode: *AcquisitionMode*

Gets or sets the acquisition mode.

Return type
AcquisitionMode

property acquisition_mode_info: Tuple[*AcquisitionMode*, ...]

Gets the supported acquisition modes.

Return type
Tuple[*AcquisitionMode*, ...]

property auto_triggered: bool

Returns True if the acquisition is auto triggered.

Before calling this function, call the 'read_status()' function to read the data from the device.

Return type
bool

property buffer_size: int

Gets or sets the buffer size.

Return type
int

property buffer_size_max: int

Gets the maximum supported buffer size.

Return type
int

property buffer_size_min: int

Gets the minimum supported buffer size.

Return type
int

property channels: Tuple[*AnalogInputChannel*, ...]

Gets a collection of Analog Input channels.

Return type
Tuple[*AnalogInputChannel*, ...]

configure(*reconfigure=False*, *start=False*)

Configures the instrument and optionally starts the acquisition.

Return type
None

property counter: float

Gets or sets the current timeout in seconds.

Return type
float

property counter_max: float

Gets the maximum count value.

Return type
float

property counter_status: Tuple[float, float, int]

Gets the count, frequency and tick values.

Return type
Tuple[float, float, int]

property device: *Device*

Gets the device.

Return type
Device

force_trigger()

Force trigger of instrument.

Return type
None

property frequency: float

Gets or sets the configured sample frequency in Hz.

Return type
float

property frequency_max: float

Gets the maximum supported sample frequency.

Return type
float

property frequency_min: float

Gets the minimum supported sample frequency.

Return type
float

property noise_buffer_size: int

Gets or sets the noise buffer size.

Return type
int

property noise_buffer_size_max: int

Gets the maximum supported noise buffer size.

Return type
int

read_status(*read_data=False*)

Gets the acquisition state and optionally reads the data.

Return type
Status

record(*sample_rate=None*, *length=None*, *buffer_size=None*, *callback=None*, *configure=False*, *start=False*)

Starts a data recording.

sample_rate
[float, optional] The sampling rate in Hz.

length
[float, optional] The recording length in seconds.

buffer_size
[int or float, optional] The buffer size.

configure
[bool, optional] If True, then the instrument is configured (default False).

start
[bool, optional] If True, then the recording is started (default False).

AnalogRecorder
The recorder instance.

Return type
AnalogRecorder

property record_length: float

Gets or sets the record length in seconds.

Return type
float

property record_status: Tuple[int, int, int]

Gets information about the recording process: Returns (available_samples, lost_samples, corrupted_samples)

Before calling this function, call the 'read_status()' function to read the data from the device.

Return type
Tuple[int, int, int]

property remaining_samples: int

Gets the number of samples left in the acquisition.

Before calling this function, call the 'read_status()' function to read the data from the device.

Return type
int

reset()

Resets and configures all instrument parameters to default values.

Return type
None

scan_screen(*sample_rate=None*, *buffer_size=None*, *configure=False*, *start=False*)

Starts a scan-screen data acquisition.

sample_rate
[float, optional] The sampling rate in Hz.

buffer_size
[int or float, optional] The buffer size.

configure
[bool, optional] If True, then the instrument is configured (default False).

start
[bool, optional] If True, then the acquisition is started (default False).

Return type
None

scan_shift(*sample_rate=None*, *buffer_size=None*, *configure=False*, *start=False*)

Starts a scan-shift data acquisition.

sample_rate
[float, optional] The sampling rate in Hz.

buffer_size
[int or float, optional] The buffer size.

configure
[bool, optional] If True, then the instrument is configured (default False).

start
[bool, optional] If True, then the acquisition is started (default False).

Return type
None

`setup_acquisition`(*mode=None*, *sample_rate=None*, *buffer_size=None*, *record_length=None*, *configure=False*, *start=False*)

Sets up a new data acquisition.

mode
[str or AcquisitionMode, optional] The sampling mode. Can be 'single', 'scan-shift', 'scan-screen', or 'record'.

sample_rate
[float, optional] The sampling rate in Hz.

buffer_size
[int or float, optional] The buffer size.

record_length
[float, optional] The record length in seconds.

configure
[bool, optional] If True, then the instrument is configured (default False).

start
[bool, optional] If True, then the acquisition is started (default False).

Return type
None

`setup_channel`(*channel*, *range=None*, *offset=None*, *coupling=None*, *bandwidth=None*, *attenuation=None*, *impedance=None*, *filter=None*, *enabled=True*)

Sets up a channel for data acquisition.

channel
[int] The channel to setup.

range
[float, optional] The channel range in Volts.

offset
[float, optional] The channel offset in Volts.

coupling
[str or AnalogInputCoupling, optional] The channel coupling. Can be 'dc' or 'ac'.

bandwidth
[float, optional] The channel bandwidth in Hz.

attenuation
[float, optional] The channel attenuation.

impedance
[float, optional] The channel impedance in Ohms.

filter
[str or FilterMode, optional] The channel acquisition filter. Can be 'decimate', 'average', or 'min-max'.

enabled
[bool, optional] If True, then the channel is enabled (default True).

Return type
None

`setup_edge_trigger`(*channel*, *slope=None*, *level=None*, *hysteresis=None*, *position=None*, *hold_off=None*, *mode=None*)

Trigger upon a certain voltage level in the positive or negative slope of the waveform.

channel
[int] The source channel used for triggering.

slope
[str or TriggerSlope, optional] The trigger slope. Can be 'rising', 'falling', or 'either'.

level
[float, optional] The trigger level in Volts.

hysteresis
[float, optional] The trigger hysteresis in Volts.

position
[float, optional] The horizontal trigger position in seconds.

hold_off
[float, optional] The trigger hold-off time in seconds.

mode
[str, optional] The trigger mode. Can be 'normal' or 'auto'.

Return type
None

`setup_pulse_trigger`(*channel*, *condition=None*, *length_condition=None*, *length=None*, *level=None*, *hysteresis=None*, *position=None*, *hold_off=None*, *mode=None*)

Trigger upon a positive or negative pulse width when measured at a certain voltage level.

channel
[int] The source channel used for triggering.

condition
[str, optional] The trigger condition. Can be 'positive' or 'negative'.

length_condition
[str, optional] The trigger length condition. Can be 'less', 'timeout', or 'more'.

length
[float, optional] The pulse length in seconds.

level
[float, optional] The trigger level in Volts.

hysteresis
[float, optional] The trigger hysteresis in Volts.

position
[float, optional] The horizontal trigger position in seconds.

hold_off
[float, optional] The trigger hold-off time in seconds.

mode
[str, optional] The trigger mode. Can be 'normal' or 'auto'.

Return type
None

`setup_transition_trigger`(*channel*, *condition=None*, *length_condition=None*, *length=None*, *level=None*, *hysteresis=None*, *position=None*, *hold_off=None*, *mode=None*)

Sets up a transition trigger.

channel
[int] The source channel used for triggering.

condition
[str, optional] The trigger condition. Can be 'rising', 'falling', or 'either'.

length_condition
[str, optional] The trigger length condition. Can be 'less', 'timeout', or 'more'.

length
[float, optional] The transition length in seconds.

level
[float, optional] The trigger level in Volts.

hysteresis
[float, optional] The trigger hysteresis in Volts.

position
[float, optional] The horizontal trigger position in seconds.

hold_off
[float, optional] The trigger hold-off time in seconds.

mode
[str, optional] The trigger mode. Can be 'normal' or 'auto'.

Return type
None

`setup_window_trigger`(*channel*, *condition=None*, *length=None*, *level=None*, *hysteresis=None*, *position=None*, *hold_off=None*, *mode=None*)

Trigger upon a signal entering or exiting a window at certain voltage thresholds.

channel
[int] The source channel used for triggering.

condition
[str, optional] The trigger condition. Can be 'entering' or 'exiting'.

length
[float, optional] The window length in seconds.

level
[float, optional] The trigger level in Volts.

hysteresis
[float, optional] The trigger hysteresis in Volts.

position
[float, optional] The horizontal trigger position in seconds.

hold_off
[float, optional] The trigger hold-off time in seconds.

mode
[str, optional] The trigger mode. Can be 'normal' or 'auto'.

Return type
None

single(*sample_rate=None*, *buffer_size=None*, *continuous=True*, *configure=False*, *start=False*)

Starts a single data acquisition.

sample_rate
[float, optional] The sampling rate in Hz.

buffer_size
[int or float, optional] The buffer size.

continuous
[bool, optional] If True, then the instrument is rearmed after the data is retrieved. (default True).

configure
[bool, optional] If True, then the instrument is configured (default False).

start
[bool, optional] If True, then the acquisition is started (default False).

Return type
None

property time: Tuple[int, int, int]

Gets instrument trigger time information.

Before calling this function, call the 'read_status()' function to read the data from the device.

Return type
Tuple[int, int, int]

property timeout_max: float

Gets the maximum timeout value.

Return type
float

property trigger: *AnalogInputTrigger*

Gets the trigger unit.

Return type
AnalogInputTrigger

property valid_samples: int

Gets the number of valid/acquired data samples.

Before calling this function, call the 'read_status()' function to read the data from the device.

Return type
int

wait_for_status(*status*, *read_data=False*)

Waits for the specified acquisition state.

Return type
None

property write_index: int

Gets the buffer write pointer, which is needed in scan_screen acquisition mode to display the scan bar.

Before calling this function, call the 'read_status()' function to read the data from the device.

Return type
int

16.3.2 dwfpy.analog_input.AnalogInputChannel

class AnalogInputChannel(*module*, *channel*)

Bases: object

Represents an Analog Input channel.

Methods

get_data	Gets the acquired data samples.
get_noise	Gets the acquired noise samples.
get_sample	Gets the last ADC conversion sample.
setup	Sets up the channel for data acquisition.

Attributes

adc_bits	Gets the number bits used by the ADC.
attenuation	Gets or sets the channel attenuation.
bandwidth	Gets or sets the channel bandwidth in Hz.
coupling	Gets or sets the channel coupling.
coupling_info	Gets the supported channel coupling modes.
device	Gets the device.
enabled	Enables or disables the channel.
filter	Gets or sets the acquisition filter.
filter_info	Gets the supported acquisition filters.
impedance	Gets or sets the channel impedance in Ohms.
index	Gets the channel index.
label	Gets or sets the channel label.
module	Gets the Analog Input module.
offset	Gets or sets the channel offset in volts.
offset_max	Gets the maximum supported offset voltage.
offset_min	Gets the minimum supported offset voltage.
offset_steps	Gets the number of adjustable steps.
range	Gets or sets the channel range.
range_info	Gets the supported channel ranges.
range_max	Gets the maximum channel range.
range_min	Gets the minimum channel range.
range_steps	Gets the number of channel range steps.

property adc_bits: int

Gets the number bits used by the ADC.

Return type
int

property attenuation: float

Gets or sets the channel attenuation.

Return type
float

property bandwidth: float

Gets or sets the channel bandwidth in Hz.

Return type
float

property coupling: *AnalogInputCoupling*

Gets or sets the channel coupling.

Return type
AnalogInputCoupling

property coupling_info: Tuple[*AnalogInputCoupling*, ...]

Gets the supported channel coupling modes.

Return type
Tuple[*AnalogInputCoupling*, ...]

property device: *Device*

Gets the device.

Return type
Device

property enabled: bool

Enables or disables the channel.

Return type
bool

property filter: *FilterMode*

Gets or sets the acquisition filter.

Return type
FilterMode

property filter_info: Tuple[*FilterMode*, ...]

Gets the supported acquisition filters.

Return type
Tuple[*FilterMode*, ...]

get_data(*first_sample=0*, *sample_count=- 1*, *raw=False*)

Gets the acquired data samples.

Before calling this function, call the 'read_status()' function to read the data from the device.

get_noise(*first_sample=0*, *sample_count=- 1*)

Gets the acquired noise samples.

Before calling this function, call the 'read_status()' function to read the data from the device.

get_sample()

Gets the last ADC conversion sample.

Before calling this function, call the 'read_status()' function to read the data from the device.

Return type
float

property impedance: float

Gets or sets the channel impedance in Ohms.

Return type
float

property index: int

Gets the channel index.

Return type
int

property label: str

Gets or sets the channel label.

Return type
str

property module: *AnalogInput*

Gets the Analog Input module.

Return type
AnalogInput

property offset: float

Gets or sets the channel offset in volts.

Return type
float

property offset_max: float

Gets the maximum supported offset voltage.

Return type
float

property offset_min: float

Gets the minimum supported offset voltage.

Return type
float

property offset_steps: int

Gets the number of adjustable steps.

Return type
int

property range: float

Gets or sets the channel range.

Return type
float

property range_info: Tuple[float, ...]

Gets the supported channel ranges.

Return type
Tuple[float, ...]

property range_max: float

Gets the maximum channel range.

Return type
float

property range_min: float

Gets the minimum channel range.

Return type
float

property range_steps: int

Gets the number of channel range steps.

Return type
int

setup(*range=None*, *offset=None*, *coupling=None*, *bandwidth=None*, *attenuation=None*, *impedance=None*, *filter=None*, *enabled=True*)

Sets up the channel for data acquisition.

range
[float, optional] The channel range in Volts.

offset
[float, optional] The channel offset in Volts.

coupling
[str or AnalogInputCoupling, optional] The channel coupling. Can be 'dc' or 'ac'.

bandwidth
[float, optional] The channel bandwidth in Hz.

attenuation
[float, optional] The channel attenuation.

impedance
[float, optional] The channel impedance in Ohms.

filter
[str or FilterMode, optional] The channel acquisition filter. Can be 'decimate', 'average', or 'min-max'.

enabled
[bool, optional] If True, then the channel is enabled (default True).

Return type
None

16.3.3 dwfpy.analog_input.AnalogInputTrigger

class AnalogInputTrigger(*module*)

Bases: `object`

Represents the trigger unit of an Analog Input module.

Methods

Attributes

actual_position	Gets the actual trigger position in seconds.
auto_timeout	Gets or sets the auto trigger timeout in seconds.
auto_timeout_max	Gets the maximum auto trigger timeout in seconds.
auto_timeout_min	Gets the minimum auto trigger timeout in seconds.
auto_timeout_steps	Gets the number of adjustable steps for the auto trigger timeout.
channel	Gets or sets the trigger channel.
channel_max	Gets the maximum channel index that can be triggered on.
channel_min	Gets the minimum channel index that can be triggered on.
condition	Gets or sets the trigger condition.
condition_info	Gets the supported trigger conditions.
filter	Gets or sets the trigger filter.
filter_info	Gets the supported trigger filters.
hold_off	Gets or sets the trigger hold-off time in seconds.
hold_off_max	Gets the maximum trigger hold-off time in seconds.
hold_off_min	Gets the minimum trigger hold-off time in seconds.
hold_off_steps	Gets the number of adjustable steps for the trigger hold-off time.
hysteresis	Gets or sets the trigger hysteresis level in volts.
hysteresis_max	Gets the maximum trigger hysteresis level.
hysteresis_min	Gets the minimum trigger hysteresis level.
hysteresis_steps	Gets the number of trigger hysteresis level steps.
length	Gets or sets the trigger length in seconds.
length_condition	Gets or sets the trigger length condition.
length_condition_info	Gets the supported trigger length conditions.
length_max	Gets the maximum trigger length in seconds.
length_min	Gets the minimum trigger length in seconds.
length_steps	Gets the number of trigger length steps.
level	Gets or sets the trigger voltage level in volts.
level_max	Gets the maximum trigger voltage level.
level_min	Gets the minimum trigger voltage level.
level_steps	Gets the number of trigger voltage level steps.
position	Gets or sets the horizontal trigger position in seconds.
position_max	Gets the maximum supported trigger position in seconds.

continues on next page

Table 1 – continued from previous page

position_min	Gets the minimum supported trigger position in seconds.
position_steps	Gets the number of trigger position steps.
sampling_delay	Gets or sets the sampling delay.
sampling_slope	Gets or sets the sampling slope.
sampling_source	Gets or sets the acquisition data sampling source.
source	Gets or sets the current trigger source setting for the instrument.
type	Gets or sets the output type.
type_info	Gets the supported output types.

property actual_position: float

Gets the actual trigger position in seconds.

Return type
float

property auto_timeout: float

Gets or sets the auto trigger timeout in seconds.

Return type
float

property auto_timeout_max: float

Gets the maximum auto trigger timeout in seconds.

Return type
float

property auto_timeout_min: float

Gets the minimum auto trigger timeout in seconds.

Return type
float

property auto_timeout_steps: int

Gets the number of adjustable steps for the auto trigger timeout.

Return type
int

property channel: int

Gets or sets the trigger channel.

Return type
int

property channel_max: int

Gets the maximum channel index that can be triggered on.

Return type
int

property channel_min: int

Gets the minimum channel index that can be triggered on.

Return type
int

property condition: *TriggerSlope*

Gets or sets the trigger condition.

Return type
TriggerSlope

property condition_info: Tuple[*TriggerSlope*, ...]

Gets the supported trigger conditions.

Return type
Tuple[*TriggerSlope*, ...]

property filter: *FilterMode*

Gets or sets the trigger filter.

Return type
FilterMode

property filter_info: Tuple[*FilterMode*, ...]

Gets the supported trigger filters.

Return type
Tuple[*FilterMode*, ...]

property hold_off: float

Gets or sets the trigger hold-off time in seconds.

Return type
float

property hold_off_max: float

Gets the maximum trigger hold-off time in seconds.

Return type
float

property hold_off_min: float

Gets the minimum trigger hold-off time in seconds.

Return type
float

property hold_off_steps: int

Gets the number of adjustable steps for the trigger hold-off time.

Return type
int

property hysteresis: float

Gets or sets the trigger hysteresis level in volts.

Return type
float

property hysteresis_max: float

Gets the maximum trigger hysteresis level.

Return type
float

property hysteresis_min: float

Gets the minimum trigger hysteresis level.

Return type
float

property hysteresis_steps: int

Gets the number of trigger hysteresis level steps.

Return type
int

property length: float

Gets or sets the trigger length in seconds.

Return type
float

property length_condition: *TriggerLengthCondition*

Gets or sets the trigger length condition.

Return type
TriggerLengthCondition

property length_condition_info: Tuple[*TriggerLengthCondition*, ...]

Gets the supported trigger length conditions.

Return type
Tuple[*TriggerLengthCondition*, ...]

property length_max: float

Gets the maximum trigger length in seconds.

Return type
float

property length_min: float

Gets the minimum trigger length in seconds.

Return type
float

property length_steps: int

Gets the number of trigger length steps.

Return type
int

property level: float

Gets or sets the trigger voltage level in volts.

Return type
float

property level_max: float

Gets the maximum trigger voltage level.

Return type
float

property level_min: float

Gets the minimum trigger voltage level.

Return type
float

property level_steps: int

Gets the number of trigger voltage level steps.

Return type
int

property position: float

Gets or sets the horizontal trigger position in seconds.

Return type
float

property position_max: float

Gets the maximum supported trigger position in seconds.

Return type
float

property position_min: float

Gets the minimum supported trigger position in seconds.

Return type
float

property position_steps: int

Gets the number of trigger position steps.

Return type
int

property sampling_delay: float

Gets or sets the sampling delay.

Return type
float

property sampling_slope: *TriggerSlope*

Gets or sets the sampling slope.

Return type
TriggerSlope

property sampling_source: *TriggerSource*

Gets or sets the acquisition data sampling source.

Return type
TriggerSource

property source: *TriggerSource*

Gets or sets the current trigger source setting for the instrument.

Return type
TriggerSource

property type: *TriggerType*

Gets or sets the output type.

Return type
TriggerType

property type_info: Tuple[*TriggerType*, ...]

Gets the supported output types.

Return type
Tuple[*TriggerType*, ...]

16.4 dwfpy.analog_io

Analog IO module for Digilent WaveForms devices.

Classes

AnalogIo(device)	Analog IO module.
AnalogIoChannel(module, channel)	Represents an Analog IO channel.
AnalogIoChannelNode(channel, node)	Represents an Analog IO channel node.

16.4.1 dwfpy.analog_io.AnalogIo

class AnalogIo(*device*)

Bases: object

Analog IO module.

Methods

configure	Configures the instrument.
read_status	Reads the status of the device.
reset	Resets and configures all instrument parameters to default values.

Attributes

channels	Gets a collection of Analog IO channels.
device	Gets the device.
master_enable	Gets or sets the master enable switch.
master_enable_can_read	Return True when the status of the master enable can be read.
master_enable_can_set	Return True when the status of the master enable can be set.
master_enable_status	Gets the master enable status.

property channels: Tuple[*AnalogIoChannel*, ...]

Gets a collection of Analog IO channels.

Return type
Tuple[*AnalogIoChannel*, ...]

configure()

Configures the instrument.

Return type
None

property device: *Device*

Gets the device.

Return type
Device

property master_enable: bool

Gets or sets the master enable switch.

Return type
bool

property master_enable_can_read: bool

Return True when the status of the master enable can be read.

Return type
bool

property master_enable_can_set: bool

Return True when the status of the master enable can be set.

Return type
bool

property master_enable_status: bool

Gets the master enable status.

Return type
bool

read_status()

Reads the status of the device.

Return type
None

reset()

Resets and configures all instrument parameters to default values.

Return type
None

16.4.2 dwfpy.analog_io.AnalogIoChannel

class AnalogIoChannel(*module*, *channel*)

Bases: `object`

Represents an Analog IO channel.

Methods

Attributes

device	Gets the device.
index	Gets the channel index.
label	Gets the channel label.
module	Gets the Analog IO module.
name	Gets the channel name.
nodes	Gets the channel nodes.

property device: *Device*

Gets the device.

Return type
Device

property index: int

Gets the channel index.

Return type
int

property label: str

Gets the channel label.

Return type
str

property module: *AnalogIo*

Gets the Analog IO module.

Return type
AnalogIo

property name: str

Gets the channel name.

Return type
str

property nodes: Tuple[*AnalogIoChannelNode*, ...]

Gets the channel nodes.

Return type
Tuple[*AnalogIoChannelNode*, ...]

16.4.3 dwfpy.analog_io.AnalogIoChannelNode

class AnalogIoChannelNode(*channel*, *node*)

Bases: `object`

Represents an Analog IO channel node.

Methods

Attributes

index	Gets the node index.
name	Gets the node name.
node_info	Gets the supported channel node types.
status	Gets the actual node value.
status_max	Gets the maximum node value.
status_min	Gets the minimum node value.
status_steps	Gets the number of node value steps.
unit	Gets the unit of the node value.
value	Gets or sets the node value.
value_max	Gets the maximum node value.
value_min	Gets the minimum node value.
value_steps	Gets the number of node value steps.

property index: int

Gets the node index.

Return type
`int`

property name: str

Gets the node name.

Return type
`str`

property node_info: Tuple[*ChannelNodeType*, ...]

Gets the supported channel node types.

Return type
`Tuple[`*ChannelNodeType*`, ...]`

property status: float

Gets the actual node value.

Return type
`float`

property status_max: float

Gets the maximum node value.

Return type
`float`

property status_min: float

Gets the minimum node value.

Return type
float

property status_steps: int

Gets the number of node value steps.

Return type
int

property unit: str

Gets the unit of the node value.

Return type
str

property value: float

Gets or sets the node value.

Return type
float

property value_max: float

Gets the maximum node value.

Return type
float

property value_min: float

Gets the minimum node value.

Return type
float

property value_steps: int

Gets the number of node value steps.

Return type
int

16.5 dwfpy.analog_output

Analog Output module for Digilent WaveForms devices.

Classes

AnalogOutput(device)	Analog Output module (Arbitrary Waveform Generator).
AnalogOutputChannel(module, channel)	Represents an Analog Output channel.
AnalogOutputChannelNode(channel, node)	Represents an Analog Output channel node.
AnalogOutputChannelTrigger(channel)	Represents the trigger unit of an Analog Output channel.

16.5.1 dwfpy.analog_output.AnalogOutput

class AnalogOutput(*device*)

Bases: `object`

Analog Output module (Arbitrary Waveform Generator).

Methods

Attributes

channels	Gets a collection of Analog Output channels.
device	Gets the device.

property channels: Tuple[*AnalogOutputChannel*, ...]

Gets a collection of Analog Output channels.

Return type
Tuple[*AnalogOutputChannel*, ...]

property device: *Device*

Gets the device.

Return type
Device

16.5.2 dwfpy.analog_output.AnalogOutputChannel

class AnalogOutputChannel(*module*, *channel*)

Bases: `object`

Represents an Analog Output channel.

Methods

apply	Applies changes to the instrument.
configure	Configures and optionally starts the instrument.
read_status	Gets the instrument status.
reset	Resets and configures all instrument parameters to default values.
setup	Sets up a new carrier waveform.
setup_am	Applies an AM modulation to a waveform.
setup_fm	Applies an FM modulation to a waveform.

Attributes

device	Gets the device.
enable_repeat_trigger	Enables the repeat count trigger in wait-run repeat cycles.
idle	Gets or sets the generator idle output option.
idle_info	Gets the supported channel nodes.
index	Gets the channel index.
label	Gets or sets the channel label.
limitation	Gets or sets the limitation value.
limitation_max	Gets the maximum limitation value.
limitation_min	Gets the minimum limitation value.
master	Gets or sets the master node index.
mode	Gets or sets the generator mode option.
module	Gets the Analog Output module.
nodes	Gets the channel nodes.
repeat_count	Gets or sets the repeat count.
repeat_count_max	Gets the maximum repeat count.
repeat_count_min	Gets the minimum repeat count.
repeat_count_status	Gets the remaining repeat count.
run_length	Gets or sets the run length in seconds.
run_length_max	Gets the maximum run length in seconds.
run_length_min	Gets the minimum run length in seconds.
run_length_status	Gets the remaining run length in seconds.
trigger	Gets the trigger unit.
wait_length	Gets or sets the wait length in seconds.
wait_length_max	Gets the maximum wait length in seconds.
wait_length_min	Gets the minimum wait length in seconds.

apply()

Applies changes to the instrument.

Return type
None

configure(*start=False*)

Configures and optionally starts the instrument.

Return type
None

property device: *Device*

Gets the device.

Return type
Device

property enable_repeat_trigger: **bool**

Enables the repeat count trigger in wait-run repeat cycles.

Return type
bool

property idle: *AnalogOutputIdle*

Gets or sets the generator idle output option.

Return type
AnalogOutputIdle

property idle_info: Tuple[*AnalogOutputIdle*, ...]

Gets the supported channel nodes.

Return type
Tuple[*AnalogOutputIdle*, ...]

property index: int

Gets the channel index.

Return type
int

property label: str

Gets or sets the channel label.

Return type
str

property limitation: float

Gets or sets the limitation value. Voltage offset in volts or modulation offset percentage.

Return type
float

property limitation_max: float

Gets the maximum limitation value.

Return type
float

property limitation_min: float

Gets the minimum limitation value.

Return type
float

property master: int

Gets or sets the master node index.

Return type
int

property mode: *AnalogOutputMode*

Gets or sets the generator mode option.

Return type
AnalogOutputMode

property module: *AnalogOutput*

Gets the Analog Output module.

Return type
AnalogOutput

property nodes: Tuple[*AnalogOutputChannelNode*, ...]

Gets the channel nodes.

Return type
Tuple[*AnalogOutputChannelNode*, ...]

read_status()

Gets the instrument status.

Return type
Status

property repeat_count: int

Gets or sets the repeat count.

Return type
int

property repeat_count_max: int

Gets the maximum repeat count.

Return type
int

property repeat_count_min: int

Gets the minimum repeat count.

Return type
int

property repeat_count_status: float

Gets the remaining repeat count.

Return type
float

reset()

Resets and configures all instrument parameters to default values.

Return type
None

property run_length: float

Gets or sets the run length in seconds.

Return type
float

property run_length_max: float

Gets the maximum run length in seconds.

Return type
float

property run_length_min: float

Gets the minimum run length in seconds.

Return type
float

property run_length_status: float

Gets the remaining run length in seconds.

Return type
float

setup(*function=None*, *frequency=None*, *amplitude=None*, *offset=None*, *symmetry=None*, *phase=None*, *enabled=True*, *configure=False*, *start=False*)

Sets up a new carrier waveform.

function
[str, optional] The generator function. Can be 'dc', 'sine', 'square', 'triangle', 'ramp-up', 'ramp-down', 'noise', 'pulse', 'trapezium', or 'sine-power'.

frequency
[float, optional] The waveform frequency in Hz.

amplitude
[float, optional] The waveform amplitude in Volts.

offset
[float, optional] The waveform offset in Volts.

symmetry
[float, optional] The waveform symmetry (or duty cycle) in percent.

phase
[float, optional] The waveform phase in degree.

enabled
[bool, optional] If True, then the node is enabled (default True).

configure
[bool, optional] If True, then the instrument is configured (default False).

start
[bool, optional] If True, then the instrument is started (default False).

Return type
None

setup_am(*function=None*, *frequency=None*, *amplitude=None*, *offset=None*, *symmetry=None*, *phase=None*, *enabled=True*, *configure=False*, *start=False*)

Applies an AM modulation to a waveform.

function
[str, optional] The generator function. Can be 'dc', 'sine', 'square', 'triangle', 'ramp-up', 'ramp-down', 'noise', 'pulse', 'trapezium', or 'sine-power'.

frequency
[float, optional] The waveform frequency in Hz.

amplitude
[float, optional] The waveform amplitude in percent.

offset
[float, optional] The waveform offset in percent.

symmetry
[float, optional] The waveform symmetry (or duty cycle) in percent.

phase
[float, optional] The waveform phase in degree.

enabled
[bool, optional] If True, then the node is enabled (default True).

configure
[bool, optional] If True, then the instrument is configured (default False).

start
[bool, optional] If True, then the instrument is started (default False).

Return type
None

`setup_fm`(*function=None*, *frequency=None*, *amplitude=None*, *offset=None*, *symmetry=None*, *phase=None*, *enabled=True*, *configure=False*, *start=False*)

Applies an FM modulation to a waveform.

function
[str, optional] The generator function. Can be 'dc', 'sine', 'square', 'triangle', 'ramp-up', 'ramp-down', 'noise', 'pulse', 'trapezium', or 'sine-power'.

frequency
[float, optional] The waveform frequency in Hz.

amplitude
[float, optional] The waveform amplitude in percent.

offset
[float, optional] The waveform offset in percent.

symmetry
[float, optional] The waveform symmetry (or duty cycle) in percent.

phase
[float, optional] The waveform phase in degree.

enabled
[bool, optional] If True, then the node is enabled (default True).

configure
[bool, optional] If True, then the instrument is configured (default False).

start
[bool, optional] If True, then the instrument is started (default False).

Return type
None

`property trigger: AnalogOutputChannelTrigger`

Gets the trigger unit.

Return type
AnalogOutputChannelTrigger

`property wait_length: float`

Gets or sets the wait length in seconds.

Return type
float

`property wait_length_max: float`

Gets the maximum wait length in seconds.

Return type
float

property wait_length_min: float

Gets the minimum wait length in seconds.

Return type
float

16.5.3 dwfpy.analog_output.AnalogOutputChannelNode

class AnalogOutputChannelNode(*channel*, *node*)

Bases: object

Represents an Analog Output channel node.

Methods

set_data_samples	Sets the custom data or to prefill the buffer with play samples.
set_play_samples	Sets new data samples for play mode.

Attributes

amplitude	Gets or sets the amplitude.
amplitude_max	Gets the maximum amplitude.
amplitude_min	Gets the minimum amplitude.
data_samples_max	Gets the maximum number of samples allowed for custom data generation.
data_samples_min	Gets the minimum number of samples allowed for custom data generation.
enabled	Enables the node.
frequency	Gets or sets the frequency.
frequency_max	Gets the maximum frequency.
frequency_min	Gets the minimum frequency.
function	Gets or sets the generator function.
function_info	Gets the supported channel nodes.
offset	Gets or sets the offset.
offset_max	Gets the maximum offset.
offset_min	Gets the minimum offset.
phase	Gets or sets the phase.
phase_max	Gets the maximum phase.
phase_min	Gets the minimum phase.
play_status	Gets information about the recording process.
symmetry	Gets or sets the symmetry percentage.
symmetry_max	Gets the maximum symmetry percentage.
symmetry_min	Gets the minimum symmetry percentage.
type	Gets the node type.

property amplitude: float

Gets or sets the amplitude.

Return type
float

property amplitude_max: float

Gets the maximum amplitude.

Return type
float

property amplitude_min: float

Gets the minimum amplitude.

Return type
float

property data_samples_max: int

Gets the maximum number of samples allowed for custom data generation.

Return type
int

property data_samples_min: int

Gets the minimum number of samples allowed for custom data generation.

Return type
int

property enabled: bool

Enables the node.

Return type
bool

property frequency: float

Gets or sets the frequency.

Return type
float

property frequency_max: float

Gets the maximum frequency.

Return type
float

property frequency_min: float

Gets the minimum frequency.

Return type
float

property function: *Function*

Gets or sets the generator function.

Return type
Function

property function_info: Tuple[*Function*, ...]

Gets the supported channel nodes.

Return type
Tuple[*Function*, ...]

property offset: float

Gets or sets the offset.

Return type
float

property offset_max: float

Gets the maximum offset.

Return type
float

property offset_min: float

Gets the minimum offset.

Return type
float

property phase: float

Gets or sets the phase.

Return type
float

property phase_max: float

Gets the maximum phase.

Return type
float

property phase_min: float

Gets the minimum phase.

Return type
float

property play_status: Tuple[int, int, int]

Gets information about the recording process. Returns (samples_free, lost_samples, corrupted_samples)

Return type
Tuple[int, int, int]

set_data_samples(*samples*)

Sets the custom data or to prefill the buffer with play samples.

Return type
None

set_play_samples(*samples*)

Sets new data samples for play mode.

Return type
None

property symmetry: float

Gets or sets the symmetry percentage.

Return type
float

property symmetry_max: float

Gets the maximum symmetry percentage.

Return type
float

property symmetry_min: float

Gets the minimum symmetry percentage.

Return type
float

property type: *AnalogOutputNode*

Gets the node type.

Return type
AnalogOutputNode

16.5.4 dwfpy.analog_output.AnalogOutputChannelTrigger

class AnalogOutputChannelTrigger(*channel*)

Bases: object

Represents the trigger unit of an Analog Output channel.

Methods

Attributes

slope	Gets or sets the trigger slope for the instrument.
source	Gets or sets the current trigger source setting for the instrument.

property slope: *TriggerSlope*

Gets or sets the trigger slope for the instrument.

Return type
TriggerSlope

property source: *TriggerSource*

Gets or sets the current trigger source setting for the instrument.

Return type
TriggerSource

16.6 dwfpy.analog_recorder

Recorder for Analog Input data.

Classes

AnalogRecorder(module)	Recorder for Analog Input data

16.6.1 dwfpy.analog_recorder.AnalogRecorder

class AnalogRecorder(*module*)

Bases: object

Recorder for Analog Input data

Methods

process	Checks the instrument status and processes a chunk of data if available.
record	Starts the recording and processes all samples until the recording is complete.

Attributes

channels	Gets a collection of data channels.
corrupted_samples	Gets the number of corrupted samples.
lost_samples	Gets the number of lost samples.
requested_samples	Gets the number of requested samples for recording.
status	Gets the last acquisition status.
total_samples	Gets the total number of acquired and lost samples.

class ChannelData

Bases: object

Represents the acquired data of a channel.

property data_samples: tuple

Gets the acquired data samples.

Return type
tuple

property channels: Tuple[*ChannelData*, ...]

Gets a collection of data channels.

Return type
Tuple[*ChannelData*, ...]

property corrupted_samples: int

Gets the number of corrupted samples.

Return type
int

property lost_samples: int

Gets the number of lost samples.

Return type
int

process()

Checks the instrument status and processes a chunk of data if available.

bool
If True, then if there is more data to process, and the function must be called again. If False, the recording is complete, and you must stop calling this function.

This function must be called repeatedly by the user to process the recording data. Failure to call this function in a timely manner will cause samples to get lost or corrupted.

Return type
bool

record(*callback=None*)

Starts the recording and processes all samples until the recording is complete.

callback
[function] A user-defined function that is called every time a data chunk is processed. Return True to continue recording, False to abort the recording.

This function blocks until the recording is complete.

Return type
None

property requested_samples: int

Gets the number of requested samples for recording.

Return type
int

property status: *Status*

Gets the last acquisition status.

Return type
Status

property total_samples: int

Gets the total number of acquired and lost samples.

Return type
int

16.7 dwfpy.application

Support for Digilent WaveForms applications.

Classes

Application(*args, **kwargs)	WaveForms application.

16.7.1 dwfpy.application.Application

class Application(**args*, ***kwargs*)

Bases: _Singleton

WaveForms application.

Methods

get_last_error	Gets the last DWF API error code.
get_last_error_message	Gets the last DWF API error message.
get_logger	Gets the WaveForms logger.
get_parameter	Gets a global parameter.
get_version	Gets the DWF API version string.
set_parameter	Sets a global parameter.

Attributes

clock_mode	Specifies the clock mode: 0 internal, 1 output, 2 input, 3 IO.
enable_analog_out	Enables or disables the analog audio output.
enable_audio_output	Enables or disables audio output.
external_frequency	Specifies the external frequency in Hz.
frequency	Gets or sets the frequency in Hz.
led_brightness	Gets or sets the Digital Discovery LED brightness.
on_close_behavior	Gets or sets a value indicating the device close behavior.
usb_limit	Gets or sets the USB current limitation in mA.
usb_power_on_aux	Gets or sets a value to keep the USB power enabled even when AUX supply is connected.

property clock_mode: int

Specifies the clock mode: 0 internal, 1 output, 2 input, 3 IO.

Return type
int

property enable_analog_out: bool

Enables or disables the analog audio output.

Return type
bool

property enable_audio_output: bool

Enables or disables audio output.

Return type
bool

property external_frequency: int

Specifies the external frequency in Hz.

Return type
int

property frequency: int

Gets or sets the frequency in Hz.

Return type
int

static get_last_error()

Gets the last DWF API error code.

Return type
Error

static get_last_error_message()

Gets the last DWF API error message.

Return type
str

static get_logger()

Gets the WaveForms logger.

Return type
Logger

static get_parameter(*parameter*)

Gets a global parameter.

Return type
int

static get_version()

Gets the DWF API version string.

Return type
str

property led_brightness: int

Gets or sets the Digital Discovery LED brightness.

Return type
int

property on_close_behavior: int

Gets or sets a value indicating the device close behavior. 0 = Continue, 1 = Stop, 2 = Shutdown.

Return type
int

static set_parameter(*parameter*, *value*)

Sets a global parameter.

Return type
None

property usb_limit: int

Gets or sets the USB current limitation in mA.

Return type
int

property usb_power_on_aux: bool

Gets or sets a value to keep the USB power enabled even when AUX supply is connected. Applies to Analog Discovery 2

Return type
bool

16.8 dwfpy.bindings

Python bindings for Digilent WaveForms API.

Functions

dwf_analog_in_acquisition_mode_get(*_)

dwf_analog_in_acquisition_mode_info(*_)

dwf_analog_in_acquisition_mode_set(*_)

dwf_analog_in_bits_info(*_)

dwf_analog_in_buffer_size_get(*_)

dwf_analog_in_buffer_size_info(*_)

dwf_analog_in_buffer_size_set(*_)

dwf_analog_in_channel_attenuation_get(*_)

dwf_analog_in_channel_attenuation_set(*_)

dwf_analog_in_channel_count(*_)

continues on next page

Table 2 – continued from previous page

dwf_analog_in_channel_enable_get(*_)
dwf_analog_in_channel_enable_set(*_)
dwf_analog_in_channel_filter_get(*_)
dwf_analog_in_channel_filter_info(*_)
dwf_analog_in_channel_filter_set(*_)
dwf_analog_in_channel_offset_get(*_)
dwf_analog_in_channel_offset_info(*_)
dwf_analog_in_channel_offset_set(*_)
dwf_analog_in_channel_range_get(*_)
dwf_analog_in_channel_range_info(*_)
dwf_analog_in_channel_range_set(*_)
dwf_analog_in_channel_range_steps(*_)
dwf_analog_in_configure(*_)
dwf_analog_in_frequency_get(*_)
dwf_analog_in_frequency_info(*_)
dwf_analog_in_frequency_set(*_)
dwf_analog_in_noise_size_get(*_)
dwf_analog_in_noise_size_info(*_)
dwf_analog_in_noise_size_set(*_)
dwf_analog_in_record_length_get(*_)
dwf_analog_in_record_length_set(*_)
dwf_analog_in_reset(*_)
dwf_analog_in_sampling_delay_get(*_)
dwf_analog_in_sampling_delay_set(*_)
dwf_analog_in_sampling_slope_get(*_)

continues on next page

Table 2 – continued from previous page

dwf_analog_in_sampling_slope_set(*_)
dwf_analog_in_sampling_source_get(*_)
dwf_analog_in_sampling_source_set(*_)
dwf_analog_in_status(*_)
dwf_analog_in_status_auto_triggered(*_)
dwf_analog_in_status_data(*_)
dwf_analog_in_status_data16(*_)
dwf_analog_in_status_data2(*_)
dwf_analog_in_status_index_write(*_)
dwf_analog_in_status_noise(*_)
dwf_analog_in_status_noise2(*_)
dwf_analog_in_status_record(*_)
dwf_analog_in_status_sample(*_)
dwf_analog_in_status_samples_left(*_)
dwf_analog_in_status_samples_valid(*_)
dwf_analog_in_trigger_auto_timeout_get(*_)
dwf_analog_in_trigger_auto_timeout_info(*_)
dwf_analog_in_trigger_auto_timeout_set(*_)
dwf_analog_in_trigger_channel_get(*_)
dwf_analog_in_trigger_channel_info(*_)
dwf_analog_in_trigger_channel_set(*_)
dwf_analog_in_trigger_condition_get(*_)
dwf_analog_in_trigger_condition_info(*_)
dwf_analog_in_trigger_condition_set(*_)
dwf_analog_in_trigger_filter_get(*_)

continues on next page

Table 2 – continued from previous page

dwf_analog_in_trigger_filter_info(*_)
dwf_analog_in_trigger_filter_set(*_)
dwf_analog_in_trigger_force(*_)
dwf_analog_in_trigger_hold_off_get(*_)
dwf_analog_in_trigger_hold_off_info(*_)
dwf_analog_in_trigger_hold_off_set(*_)
dwf_analog_in_trigger_hysteresis_get(*_)
dwf_analog_in_trigger_hysteresis_info(*_)
dwf_analog_in_trigger_hysteresis_set(*_)
dwf_analog_in_trigger_length_condition_get(*_)
dwf_analog_in_trigger_length_condition_info(*_)
dwf_analog_in_trigger_length_condition_set(*_)
dwf_analog_in_trigger_length_get(*_)
dwf_analog_in_trigger_length_info(*_)
dwf_analog_in_trigger_length_set(*_)
dwf_analog_in_trigger_level_get(*_)
dwf_analog_in_trigger_level_info(*_)
dwf_analog_in_trigger_level_set(*_)
dwf_analog_in_trigger_position_get(*_)
dwf_analog_in_trigger_position_info(*_)
dwf_analog_in_trigger_position_set(*_)
dwf_analog_in_trigger_position_status(*_)
dwf_analog_in_trigger_source_get(*_)
dwf_analog_in_trigger_source_set(*_)
dwf_analog_in_trigger_type_get(*_)

continues on next page

Table 2 – continued from previous page

dwf_analog_in_trigger_type_info(*_)
dwf_analog_in_trigger_type_set(*_)
dwf_analog_io_channel_count(*_)
dwf_analog_io_channel_info(*_)
dwf_analog_io_channel_name(*_)
dwf_analog_io_channel_node_get(*_)
dwf_analog_io_channel_node_info(*_)
dwf_analog_io_channel_node_name(*_)
dwf_analog_io_channel_node_set(*_)
dwf_analog_io_channel_node_set_info(*_)
dwf_analog_io_channel_node_status(*_)
dwf_analog_io_channel_node_status_info(*_)
dwf_analog_io_configure(*_)
dwf_analog_io_enable_get(*_)
dwf_analog_io_enable_info(*_)
dwf_analog_io_enable_set(*_)
dwf_analog_io_enable_status(*_)
dwf_analog_io_reset(*_)
dwf_analog_io_status(*_)
dwf_analog_out_configure(*_)
dwf_analog_out_count(*_)
dwf_analog_out_custom_am_fm_enable_get(*_)
dwf_analog_out_custom_am_fm_enable_set(*_)
dwf_analog_out_idle_get(*_)
dwf_analog_out_idle_info(*_)

continues on next page

Table 2 – continued from previous page

dwf_analog_out_idle_set(*_)
dwf_analog_out_limitation_get(*_)
dwf_analog_out_limitation_info(*_)
dwf_analog_out_limitation_set(*_)
dwf_analog_out_master_get(*_)
dwf_analog_out_master_set(*_)
dwf_analog_out_mode_get(*_)
dwf_analog_out_mode_set(*_)
dwf_analog_out_node_amplitude_get(*_)
dwf_analog_out_node_amplitude_info(*_)
dwf_analog_out_node_amplitude_set(*_)
dwf_analog_out_node_data_info(*_)
dwf_analog_out_node_data_set(*_)
dwf_analog_out_node_enable_get(*_)
dwf_analog_out_node_enable_set(*_)
dwf_analog_out_node_frequency_get(*_)
dwf_analog_out_node_frequency_info(*_)
dwf_analog_out_node_frequency_set(*_)
dwf_analog_out_node_function_get(*_)
dwf_analog_out_node_function_info(*_)
dwf_analog_out_node_function_set(*_)
dwf_analog_out_node_info(*_)
dwf_analog_out_node_offset_get(*_)
dwf_analog_out_node_offset_info(*_)
dwf_analog_out_node_offset_set(*_)

continues on next page

Table 2 – continued from previous page

dwf_analog_out_node_phase_get(*_)
dwf_analog_out_node_phase_info(*_)
dwf_analog_out_node_phase_set(*_)
dwf_analog_out_node_play_data(*_)
dwf_analog_out_node_play_status(*_)
dwf_analog_out_node_symmetry_get(*_)
dwf_analog_out_node_symmetry_info(*_)
dwf_analog_out_node_symmetry_set(*_)
dwf_analog_out_repeat_get(*_)
dwf_analog_out_repeat_info(*_)
dwf_analog_out_repeat_set(*_)
dwf_analog_out_repeat_status(*_)
dwf_analog_out_repeat_trigger_get(*_)
dwf_analog_out_repeat_trigger_set(*_)
dwf_analog_out_reset(*_)
dwf_analog_out_run_get(*_)
dwf_analog_out_run_info(*_)
dwf_analog_out_run_set(*_)
dwf_analog_out_run_status(*_)
dwf_analog_out_status(*_)
dwf_analog_out_trigger_slope_get(*_)
dwf_analog_out_trigger_slope_set(*_)
dwf_analog_out_trigger_source_get(*_)
dwf_analog_out_trigger_source_set(*_)
dwf_analog_out_wait_get(*_)

continues on next page

Table 2 – continued from previous page

dwf_analog_out_wait_info(*_)
dwf_analog_out_wait_set(*_)
dwf_device_auto_configure_get(*_)
dwf_device_auto_configure_set(*_)
dwf_device_close(*_)
dwf_device_close_all(*_)
dwf_device_config_open(*_)
dwf_device_enable_set(*_)
dwf_device_open(*_)
dwf_device_reset(*_)
dwf_device_trigger_get(*_)
dwf_device_trigger_info(*_)
dwf_device_trigger_pc(*_)
dwf_device_trigger_set(*_)
dwf_device_trigger_slope_info(*_)
dwf_digital_can_polarity_set(*_)
dwf_digital_can_rate_set(*_)
dwf_digital_can_reset(*_)
dwf_digital_can_rx(*_)
dwf_digital_can_rx_set(*_)
dwf_digital_can_tx(*_)
dwf_digital_can_tx_set(*_)
dwf_digital_i2c_clear(*_)
dwf_digital_i2c_rate_set(*_)
dwf_digital_i2c_read(*_)

continues on next page

Table 2 – continued from previous page

dwf_digital_i2c_read_nak_set(*_)
dwf_digital_i2c_reset(*_)
dwf_digital_i2c_scl_set(*_)
dwf_digital_i2c_sda_set(*_)
dwf_digital_i2c_write(*_)
dwf_digital_i2c_write_one(*_)
dwf_digital_i2c_write_read(*_)
dwf_digital_in_acquisition_mode_get(*_)
dwf_digital_in_acquisition_mode_info(*_)
dwf_digital_in_acquisition_mode_set(*_)
dwf_digital_in_bits_info(*_)
dwf_digital_in_buffer_size_get(*_)
dwf_digital_in_buffer_size_info(*_)
dwf_digital_in_buffer_size_set(*_)
dwf_digital_in_clock_source_get(*_)
dwf_digital_in_clock_source_info(*_)
dwf_digital_in_clock_source_set(*_)
dwf_digital_in_configure(*_)
dwf_digital_in_divider_get(*_)
dwf_digital_in_divider_info(*_)
dwf_digital_in_divider_set(*_)
dwf_digital_in_input_order_set(*_)
dwf_digital_in_internal_clock_info(*_)
dwf_digital_in_reset(*_)
dwf_digital_in_sample_format_get(*_)

continues on next page

Table 2 – continued from previous page

dwf_digital_in_sample_format_set(*_)
dwf_digital_in_sample_mode_get(*_)
dwf_digital_in_sample_mode_info(*_)
dwf_digital_in_sample_mode_set(*_)
dwf_digital_in_sample_sensible_get(*_)
dwf_digital_in_sample_sensible_set(*_)
dwf_digital_in_status(*_)
dwf_digital_in_status_auto_triggered(*_)
dwf_digital_in_status_data(*_)
dwf_digital_in_status_data2(*_)
dwf_digital_in_status_index_write(*_)
dwf_digital_in_status_noise2(*_)
dwf_digital_in_status_record(*_)
dwf_digital_in_status_samples_left(*_)
dwf_digital_in_status_samples_valid(*_)
dwf_digital_in_trigger_auto_timeout_get(*_)
dwf_digital_in_trigger_auto_timeout_info(*_)
dwf_digital_in_trigger_auto_timeout_set(*_)
dwf_digital_in_trigger_count_set(*_)
dwf_digital_in_trigger_get(*_)
dwf_digital_in_trigger_info(*_)
dwf_digital_in_trigger_length_set(*_)
dwf_digital_in_trigger_match_set(*_)
dwf_digital_in_trigger_position_get(*_)
dwf_digital_in_trigger_position_info(*_)

continues on next page

Table 2 – continued from previous page

dwf_digital_in_trigger_position_set(*_)
dwf_digital_in_trigger_prefill_get(*_)
dwf_digital_in_trigger_prefill_set(*_)
dwf_digital_in_trigger_reset_set(*_)
dwf_digital_in_trigger_set(*_)
dwf_digital_in_trigger_slope_get(*_)
dwf_digital_in_trigger_slope_set(*_)
dwf_digital_in_trigger_source_get(*_)
dwf_digital_in_trigger_source_set(*_)
dwf_digital_io_configure(*_)
dwf_digital_io_input_info(*_)
dwf_digital_io_input_info64(*_)
dwf_digital_io_input_status(*_)
dwf_digital_io_input_status64(*_)
dwf_digital_io_output_enable_get(*_)
dwf_digital_io_output_enable_get64(*_)
dwf_digital_io_output_enable_info(*_)
dwf_digital_io_output_enable_info64(*_)
dwf_digital_io_output_enable_set(*_)
dwf_digital_io_output_enable_set64(*_)
dwf_digital_io_output_get(*_)
dwf_digital_io_output_get64(*_)
dwf_digital_io_output_info(*_)
dwf_digital_io_output_info64(*_)
dwf_digital_io_output_set(*_)

continues on next page

Table 2 – continued from previous page

dwf_digital_io_output_set64(*_)
dwf_digital_io_reset(*_)
dwf_digital_io_status(*_)
dwf_digital_out_configure(*_)
dwf_digital_out_count(*_)
dwf_digital_out_counter_get(*_)
dwf_digital_out_counter_info(*_)
dwf_digital_out_counter_init_get(*_)
dwf_digital_out_counter_init_set(*_)
dwf_digital_out_counter_set(*_)
dwf_digital_out_data_info(*_)
dwf_digital_out_data_set(*_)
dwf_digital_out_divider_get(*_)
dwf_digital_out_divider_info(*_)
dwf_digital_out_divider_init_get(*_)
dwf_digital_out_divider_init_set(*_)
dwf_digital_out_divider_set(*_)
dwf_digital_out_enable_get(*_)
dwf_digital_out_enable_set(*_)
dwf_digital_out_idle_get(*_)
dwf_digital_out_idle_info(*_)
dwf_digital_out_idle_set(*_)
dwf_digital_out_internal_clock_info(*_)
dwf_digital_out_output_get(*_)
dwf_digital_out_output_info(*_)

continues on next page

Table 2 – continued from previous page

dwf_digital_out_output_set(*_)
dwf_digital_out_repeat_get(*_)
dwf_digital_out_repeat_info(*_)
dwf_digital_out_repeat_set(*_)
dwf_digital_out_repeat_status(*_)
dwf_digital_out_repeat_trigger_get(*_)
dwf_digital_out_repeat_trigger_set(*_)
dwf_digital_out_reset(*_)
dwf_digital_out_run_get(*_)
dwf_digital_out_run_info(*_)
dwf_digital_out_run_set(*_)
dwf_digital_out_run_status(*_)
dwf_digital_out_status(*_)
dwf_digital_out_trigger_slope_get(*_)
dwf_digital_out_trigger_slope_set(*_)
dwf_digital_out_trigger_source_get(*_)
dwf_digital_out_trigger_source_set(*_)
dwf_digital_out_type_get(*_)
dwf_digital_out_type_info(*_)
dwf_digital_out_type_set(*_)
dwf_digital_out_wait_get(*_)
dwf_digital_out_wait_info(*_)
dwf_digital_out_wait_set(*_)
dwf_digital_spi_clock_set(*_)
dwf_digital_spi_data_set(*_)

continues on next page

Table 2 – continued from previous page

dwf_digital_spi_frequency_set(*_)
dwf_digital_spi_mode_set(*_)
dwf_digital_spi_order_set(*_)
dwf_digital_spi_read(*_)
dwf_digital_spi_read16(*_)
dwf_digital_spi_read32(*_)
dwf_digital_spi_read_one(*_)
dwf_digital_spi_reset(*_)
dwf_digital_spi_select(*_)
dwf_digital_spi_write(*_)
dwf_digital_spi_write16(*_)
dwf_digital_spi_write32(*_)
dwf_digital_spi_write_one(*_)
dwf_digital_spi_write_read(*_)
dwf_digital_spi_write_read16(*_)
dwf_digital_spi_write_read32(*_)
dwf_digital_uart_bits_set(*_)
dwf_digital_uart_parity_set(*_)
dwf_digital_uart_rate_set(*_)
dwf_digital_uart_reset(*_)
dwf_digital_uart_rx(*_)
dwf_digital_uart_rx_set(*_)
dwf_digital_uart_stop_set(*_)
dwf_digital_uart_tx(*_)
dwf_digital_uart_tx_set(*_)

continues on next page

Table 2 – continued from previous page

dwf_enum(*_)	
dwf_enum_config(*_)	
dwf_enum_config_info(*_)	
dwf_enum_config_info_str(*_)	
dwf_enum_device_is_opened(*_)	
dwf_enum_device_name(*_)	
dwf_enum_device_type(*_)	
dwf_enum_sn(*_)	
dwf_enum_user_name(*_)	
dwf_get_last_error(*_)	
dwf_get_last_error_msg(*_)	
dwf_get_version(*_)	
set_error_handler(handler)	Set the error handle for checking the return code of the DWF API.

16.8.1 dwfpy.bindings.dwf_analog_in_acquisition_mode_get

dwf_analog_in_acquisition_mode_get(*_)

16.8.2 dwfpy.bindings.dwf_analog_in_acquisition_mode_info

dwf_analog_in_acquisition_mode_info(*_)

16.8.3 dwfpy.bindings.dwf_analog_in_acquisition_mode_set

dwf_analog_in_acquisition_mode_set(*_)

16.8.4 dwfpy.bindings.dwf_analog_in_bits_info

dwf_analog_in_bits_info(*_)

16.8.5 dwfpy.bindings.dwf_analog_in_buffer_size_get

dwf_analog_in_buffer_size_get(*_)

16.8.6 dwfpy.bindings.dwf_analog_in_buffer_size_info

dwf_analog_in_buffer_size_info(*_)

16.8.7 dwfpy.bindings.dwf_analog_in_buffer_size_set

dwf_analog_in_buffer_size_set(*_)

16.8.8 dwfpy.bindings.dwf_analog_in_channel_attenuation_get

dwf_analog_in_channel_attenuation_get(*_)

16.8.9 dwfpy.bindings.dwf_analog_in_channel_attenuation_set

dwf_analog_in_channel_attenuation_set(*_)

16.8.10 dwfpy.bindings.dwf_analog_in_channel_count

dwf_analog_in_channel_count(*_)

16.8.11 dwfpy.bindings.dwf_analog_in_channel_enable_get

dwf_analog_in_channel_enable_get(*_)

16.8.12 dwfpy.bindings.dwf_analog_in_channel_enable_set

dwf_analog_in_channel_enable_set(*_)

16.8.13 dwfpy.bindings.dwf_analog_in_channel_filter_get

dwf_analog_in_channel_filter_get(*_)

16.8.14 dwfpy.bindings.dwf_analog_in_channel_filter_info

dwf_analog_in_channel_filter_info(*_)

16.8.15 dwfpy.bindings.dwf_analog_in_channel_filter_set

dwf_analog_in_channel_filter_set(*_)

16.8.16 dwfpy.bindings.dwf_analog_in_channel_offset_get

dwf_analog_in_channel_offset_get(*_)

16.8.17 dwfpy.bindings.dwf_analog_in_channel_offset_info

dwf_analog_in_channel_offset_info(*_)

16.8.18 dwfpy.bindings.dwf_analog_in_channel_offset_set

dwf_analog_in_channel_offset_set(*_)

16.8.19 dwfpy.bindings.dwf_analog_in_channel_range_get

dwf_analog_in_channel_range_get(*_)

16.8.20 dwfpy.bindings.dwf_analog_in_channel_range_info

dwf_analog_in_channel_range_info(*_)

16.8.21 dwfpy.bindings.dwf_analog_in_channel_range_set

dwf_analog_in_channel_range_set(*_)

16.8.22 dwfpy.bindings.dwf_analog_in_channel_range_steps

dwf_analog_in_channel_range_steps(*_)

16.8.23 dwfpy.bindings.dwf_analog_in_configure

dwf_analog_in_configure(*_)

16.8.24 dwfpy.bindings.dwf_analog_in_frequency_get

dwf_analog_in_frequency_get(*_)

16.8.25 dwfpy.bindings.dwf_analog_in_frequency_info

dwf_analog_in_frequency_info(*_)

16.8.26 dwfpy.bindings.dwf_analog_in_frequency_set

dwf_analog_in_frequency_set(*_)

16.8.27 dwfpy.bindings.dwf_analog_in_noise_size_get

dwf_analog_in_noise_size_get(*_)

16.8.28 dwfpy.bindings.dwf_analog_in_noise_size_info

dwf_analog_in_noise_size_info(*_)

16.8.29 dwfpy.bindings.dwf_analog_in_noise_size_set

dwf_analog_in_noise_size_set(*_)

16.8.30 dwfpy.bindings.dwf_analog_in_record_length_get

dwf_analog_in_record_length_get(*_)

16.8.31 dwfpy.bindings.dwf_analog_in_record_length_set

dwf_analog_in_record_length_set(*_)

16.8.32 dwfpy.bindings.dwf_analog_in_reset

dwf_analog_in_reset(*_)

16.8.33 dwfpy.bindings.dwf_analog_in_sampling_delay_get

dwf_analog_in_sampling_delay_get(*_)

16.8.34 dwfpy.bindings.dwf_analog_in_sampling_delay_set

dwf_analog_in_sampling_delay_set(*_)

16.8.35 dwfpy.bindings.dwf_analog_in_sampling_slope_get

dwf_analog_in_sampling_slope_get(*_)

16.8.36 dwfpy.bindings.dwf_analog_in_sampling_slope_set

dwf_analog_in_sampling_slope_set(*_)

16.8.37 dwfpy.bindings.dwf_analog_in_sampling_source_get

dwf_analog_in_sampling_source_get(*_)

16.8.38 dwfpy.bindings.dwf_analog_in_sampling_source_set

dwf_analog_in_sampling_source_set(*_)

16.8.39 dwfpy.bindings.dwf_analog_in_status

dwf_analog_in_status(*_)

16.8.40 dwfpy.bindings.dwf_analog_in_status_auto_triggered

dwf_analog_in_status_auto_triggered(*_)

16.8.41 dwfpy.bindings.dwf_analog_in_status_data

dwf_analog_in_status_data(*_)

16.8.42 dwfpy.bindings.dwf_analog_in_status_data16

dwf_analog_in_status_data16(*_)

16.8.43 dwfpy.bindings.dwf_analog_in_status_data2

dwf_analog_in_status_data2(*_)

16.8.44 dwfpy.bindings.dwf_analog_in_status_index_write

dwf_analog_in_status_index_write(*_)

16.8.45 dwfpy.bindings.dwf_analog_in_status_noise

dwf_analog_in_status_noise(*_)

16.8.46 dwfpy.bindings.dwf_analog_in_status_noise2

dwf_analog_in_status_noise2(*_)

16.8.47 dwfpy.bindings.dwf_analog_in_status_record

dwf_analog_in_status_record(*_)

16.8.48 dwfpy.bindings.dwf_analog_in_status_sample

dwf_analog_in_status_sample(*_)

16.8.49 dwfpy.bindings.dwf_analog_in_status_samples_left

dwf_analog_in_status_samples_left(*_)

16.8.50 dwfpy.bindings.dwf_analog_in_status_samples_valid

dwf_analog_in_status_samples_valid(*_)

16.8.51 dwfpy.bindings.dwf_analog_in_trigger_auto_timeout_get

dwf_analog_in_trigger_auto_timeout_get(*_)

16.8.52 dwfpy.bindings.dwf_analog_in_trigger_auto_timeout_info

dwf_analog_in_trigger_auto_timeout_info(*_)

16.8.53 dwfpy.bindings.dwf_analog_in_trigger_auto_timeout_set

dwf_analog_in_trigger_auto_timeout_set(*_)

16.8.54 dwfpy.bindings.dwf_analog_in_trigger_channel_get

dwf_analog_in_trigger_channel_get(*_)

16.8.55 dwfpy.bindings.dwf_analog_in_trigger_channel_info

dwf_analog_in_trigger_channel_info(*_)

16.8.56 dwfpy.bindings.dwf_analog_in_trigger_channel_set

dwf_analog_in_trigger_channel_set(*_)

16.8.57 dwfpy.bindings.dwf_analog_in_trigger_condition_get

dwf_analog_in_trigger_condition_get(*_)

16.8.58 dwfpy.bindings.dwf_analog_in_trigger_condition_info

dwf_analog_in_trigger_condition_info(*_)

16.8.59 dwfpy.bindings.dwf_analog_in_trigger_condition_set

dwf_analog_in_trigger_condition_set(*_)

16.8.60 dwfpy.bindings.dwf_analog_in_trigger_filter_get

dwf_analog_in_trigger_filter_get(*_)

16.8.61 dwfpy.bindings.dwf_analog_in_trigger_filter_info

dwf_analog_in_trigger_filter_info(*_)

16.8.62 dwfpy.bindings.dwf_analog_in_trigger_filter_set

dwf_analog_in_trigger_filter_set(*_)

16.8.63 dwfpy.bindings.dwf_analog_in_trigger_force

dwf_analog_in_trigger_force(*_)

16.8.64 dwfpy.bindings.dwf_analog_in_trigger_hold_off_get

dwf_analog_in_trigger_hold_off_get(*_)

16.8.65 dwfpy.bindings.dwf_analog_in_trigger_hold_off_info

dwf_analog_in_trigger_hold_off_info(*_)

16.8.66 dwfpy.bindings.dwf_analog_in_trigger_hold_off_set

dwf_analog_in_trigger_hold_off_set(*_)

16.8.67 dwfpy.bindings.dwf_analog_in_trigger_hysteresis_get

dwf_analog_in_trigger_hysteresis_get(*_)

16.8.68 dwfpy.bindings.dwf_analog_in_trigger_hysteresis_info

dwf_analog_in_trigger_hysteresis_info(*_)

16.8.69 dwfpy.bindings.dwf_analog_in_trigger_hysteresis_set

dwf_analog_in_trigger_hysteresis_set(*_)

16.8.70 dwfpy.bindings.dwf_analog_in_trigger_length_condition_get

dwf_analog_in_trigger_length_condition_get(*_)

16.8.71 dwfpy.bindings.dwf_analog_in_trigger_length_condition_info

dwf_analog_in_trigger_length_condition_info(*_)

16.8.72 dwfpy.bindings.dwf_analog_in_trigger_length_condition_set

dwf_analog_in_trigger_length_condition_set(*_)

16.8.73 dwfpy.bindings.dwf_analog_in_trigger_length_get

dwf_analog_in_trigger_length_get(*_)

16.8.74 dwfpy.bindings.dwf_analog_in_trigger_length_info

dwf_analog_in_trigger_length_info(*_)

16.8.75 dwfpy.bindings.dwf_analog_in_trigger_length_set

dwf_analog_in_trigger_length_set(*_)

16.8.76 dwfpy.bindings.dwf_analog_in_trigger_level_get

dwf_analog_in_trigger_level_get(*_)

16.8.77 dwfpy.bindings.dwf_analog_in_trigger_level_info

dwf_analog_in_trigger_level_info(*_)

16.8.78 dwfpy.bindings.dwf_analog_in_trigger_level_set

dwf_analog_in_trigger_level_set(*_)

16.8.79 dwfpy.bindings.dwf_analog_in_trigger_position_get

dwf_analog_in_trigger_position_get(*_)

16.8.80 dwfpy.bindings.dwf_analog_in_trigger_position_info

dwf_analog_in_trigger_position_info(*_)

16.8.81 dwfpy.bindings.dwf_analog_in_trigger_position_set

dwf_analog_in_trigger_position_set(*_)

16.8.82 dwfpy.bindings.dwf_analog_in_trigger_position_status

dwf_analog_in_trigger_position_status(*_)

16.8.83 dwfpy.bindings.dwf_analog_in_trigger_source_get

dwf_analog_in_trigger_source_get(*_)

16.8.84 dwfpy.bindings.dwf_analog_in_trigger_source_set

dwf_analog_in_trigger_source_set(*_)

16.8.85 dwfpy.bindings.dwf_analog_in_trigger_type_get

dwf_analog_in_trigger_type_get(*_)

16.8.86 dwfpy.bindings.dwf_analog_in_trigger_type_info

dwf_analog_in_trigger_type_info(*_)

16.8.87 dwfpy.bindings.dwf_analog_in_trigger_type_set

dwf_analog_in_trigger_type_set(*_)

16.8.88 dwfpy.bindings.dwf_analog_io_channel_count

dwf_analog_io_channel_count(*_)

16.8.89 dwfpy.bindings.dwf_analog_io_channel_info

dwf_analog_io_channel_info(*_)

16.8.90 dwfpy.bindings.dwf_analog_io_channel_name

dwf_analog_io_channel_name(*_)

16.8.91 dwfpy.bindings.dwf_analog_io_channel_node_get

dwf_analog_io_channel_node_get(*_)

16.8.92 dwfpy.bindings.dwf_analog_io_channel_node_info

dwf_analog_io_channel_node_info(*_)

16.8.93 dwfpy.bindings.dwf_analog_io_channel_node_name

dwf_analog_io_channel_node_name(*_)

16.8.94 dwfpy.bindings.dwf_analog_io_channel_node_set

dwf_analog_io_channel_node_set(*_)

16.8.95 dwfpy.bindings.dwf_analog_io_channel_node_set_info

dwf_analog_io_channel_node_set_info(*_)

16.8.96 dwfpy.bindings.dwf_analog_io_channel_node_status

dwf_analog_io_channel_node_status(*_)

16.8.97 dwfpy.bindings.dwf_analog_io_channel_node_status_info

dwf_analog_io_channel_node_status_info(*_)

16.8.98 dwfpy.bindings.dwf_analog_io_configure

dwf_analog_io_configure(*_)

16.8.99 dwfpy.bindings.dwf_analog_io_enable_get

dwf_analog_io_enable_get(*_)

16.8.100 dwfpy.bindings.dwf_analog_io_enable_info

dwf_analog_io_enable_info(*_)

16.8.101 dwfpy.bindings.dwf_analog_io_enable_set

dwf_analog_io_enable_set(*_)

16.8.102 dwfpy.bindings.dwf_analog_io_enable_status

dwf_analog_io_enable_status(*_)

16.8.103 dwfpy.bindings.dwf_analog_io_reset

dwf_analog_io_reset(*_)

16.8.104 dwfpy.bindings.dwf_analog_io_status

dwf_analog_io_status(*_)

16.8.105 dwfpy.bindings.dwf_analog_out_configure

dwf_analog_out_configure(*_)

16.8.106 dwfpy.bindings.dwf_analog_out_count

dwf_analog_out_count(*_)

16.8.107 dwfpy.bindings.dwf_analog_out_custom_am_fm_enable_get

dwf_analog_out_custom_am_fm_enable_get(*_)

16.8.108 dwfpy.bindings.dwf_analog_out_custom_am_fm_enable_set

dwf_analog_out_custom_am_fm_enable_set(*_)

16.8.109 dwfpy.bindings.dwf_analog_out_idle_get

dwf_analog_out_idle_get(*_)

16.8.110 dwfpy.bindings.dwf_analog_out_idle_info

dwf_analog_out_idle_info(*_)

16.8.111 dwfpy.bindings.dwf_analog_out_idle_set

dwf_analog_out_idle_set(*_)

16.8.112 dwfpy.bindings.dwf_analog_out_limitation_get

dwf_analog_out_limitation_get(*_)

16.8.113 dwfpy.bindings.dwf_analog_out_limitation_info

dwf_analog_out_limitation_info(*_)

16.8.114 dwfpy.bindings.dwf_analog_out_limitation_set

dwf_analog_out_limitation_set(*_)

16.8.115 dwfpy.bindings.dwf_analog_out_master_get

dwf_analog_out_master_get(*_)

16.8.116 dwfpy.bindings.dwf_analog_out_master_set

dwf_analog_out_master_set(*_)

16.8.117 dwfpy.bindings.dwf_analog_out_mode_get

dwf_analog_out_mode_get(*_)

16.8.118 dwfpy.bindings.dwf_analog_out_mode_set

dwf_analog_out_mode_set(*_)

16.8.119 dwfpy.bindings.dwf_analog_out_node_amplitude_get

dwf_analog_out_node_amplitude_get(*_)

16.8.120 dwfpy.bindings.dwf_analog_out_node_amplitude_info

dwf_analog_out_node_amplitude_info(*_)

16.8.121 dwfpy.bindings.dwf_analog_out_node_amplitude_set

dwf_analog_out_node_amplitude_set(*_)

16.8.122 dwfpy.bindings.dwf_analog_out_node_data_info

dwf_analog_out_node_data_info(*_)

16.8.123 dwfpy.bindings.dwf_analog_out_node_data_set

dwf_analog_out_node_data_set(*_)

16.8.124 dwfpy.bindings.dwf_analog_out_node_enable_get

dwf_analog_out_node_enable_get(*_)

16.8.125 dwfpy.bindings.dwf_analog_out_node_enable_set

dwf_analog_out_node_enable_set(*_)

16.8.126 dwfpy.bindings.dwf_analog_out_node_frequency_get

dwf_analog_out_node_frequency_get(*_)

16.8.127 dwfpy.bindings.dwf_analog_out_node_frequency_info

dwf_analog_out_node_frequency_info(*_)

16.8.128 dwfpy.bindings.dwf_analog_out_node_frequency_set

dwf_analog_out_node_frequency_set(*_)

16.8.129 dwfpy.bindings.dwf_analog_out_node_function_get

dwf_analog_out_node_function_get(*_)

16.8.130 dwfpy.bindings.dwf_analog_out_node_function_info

dwf_analog_out_node_function_info(*_)

16.8.131 dwfpy.bindings.dwf_analog_out_node_function_set

dwf_analog_out_node_function_set(*_)

16.8.132 dwfpy.bindings.dwf_analog_out_node_info

dwf_analog_out_node_info(*_)

16.8.133 dwfpy.bindings.dwf_analog_out_node_offset_get

dwf_analog_out_node_offset_get(*_)

16.8.134 dwfpy.bindings.dwf_analog_out_node_offset_info

dwf_analog_out_node_offset_info(*_)

16.8.135 dwfpy.bindings.dwf_analog_out_node_offset_set

dwf_analog_out_node_offset_set(*_)

16.8.136 dwfpy.bindings.dwf_analog_out_node_phase_get

dwf_analog_out_node_phase_get(*_)

16.8.137 dwfpy.bindings.dwf_analog_out_node_phase_info

dwf_analog_out_node_phase_info(*_)

16.8.138 dwfpy.bindings.dwf_analog_out_node_phase_set

dwf_analog_out_node_phase_set(*_)

16.8.139 dwfpy.bindings.dwf_analog_out_node_play_data

dwf_analog_out_node_play_data(*_)

16.8.140 dwfpy.bindings.dwf_analog_out_node_play_status

dwf_analog_out_node_play_status(*_)

16.8.141 dwfpy.bindings.dwf_analog_out_node_symmetry_get

dwf_analog_out_node_symmetry_get(*_)

16.8.142 dwfpy.bindings.dwf_analog_out_node_symmetry_info

dwf_analog_out_node_symmetry_info(*_)

16.8.143 dwfpy.bindings.dwf_analog_out_node_symmetry_set

dwf_analog_out_node_symmetry_set(*_)

16.8.144 dwfpy.bindings.dwf_analog_out_repeat_get

dwf_analog_out_repeat_get(*_)

16.8.145 dwfpy.bindings.dwf_analog_out_repeat_info

dwf_analog_out_repeat_info(*_)

16.8.146 dwfpy.bindings.dwf_analog_out_repeat_set

dwf_analog_out_repeat_set(*_)

16.8.147 dwfpy.bindings.dwf_analog_out_repeat_status

dwf_analog_out_repeat_status(*_)

16.8.148 dwfpy.bindings.dwf_analog_out_repeat_trigger_get

dwf_analog_out_repeat_trigger_get(*_)

16.8.149 dwfpy.bindings.dwf_analog_out_repeat_trigger_set

dwf_analog_out_repeat_trigger_set(*_)

16.8.150 dwfpy.bindings.dwf_analog_out_reset

dwf_analog_out_reset(*_)

16.8.151 dwfpy.bindings.dwf_analog_out_run_get

dwf_analog_out_run_get(*_)

16.8.152 dwfpy.bindings.dwf_analog_out_run_info

dwf_analog_out_run_info(*_)

16.8.153 dwfpy.bindings.dwf_analog_out_run_set

dwf_analog_out_run_set(*_)

16.8.154 dwfpy.bindings.dwf_analog_out_run_status

dwf_analog_out_run_status(*_)

16.8.155 dwfpy.bindings.dwf_analog_out_status

dwf_analog_out_status(*_)

16.8.156 dwfpy.bindings.dwf_analog_out_trigger_slope_get

dwf_analog_out_trigger_slope_get(*_)

16.8.157 dwfpy.bindings.dwf_analog_out_trigger_slope_set

dwf_analog_out_trigger_slope_set(*_)

16.8.158 dwfpy.bindings.dwf_analog_out_trigger_source_get

dwf_analog_out_trigger_source_get(*_)

16.8.159 dwfpy.bindings.dwf_analog_out_trigger_source_set

dwf_analog_out_trigger_source_set(*_)

16.8.160 dwfpy.bindings.dwf_analog_out_wait_get

dwf_analog_out_wait_get(*_)

16.8.161 dwfpy.bindings.dwf_analog_out_wait_info

dwf_analog_out_wait_info(*_)

16.8.162 dwfpy.bindings.dwf_analog_out_wait_set

dwf_analog_out_wait_set(*_)

16.8.163 dwfpy.bindings.dwf_device_auto_configure_get

dwf_device_auto_configure_get(*_)

16.8.164 dwfpy.bindings.dwf_device_auto_configure_set

dwf_device_auto_configure_set(*_)

16.8.165 dwfpy.bindings.dwf_device_close

dwf_device_close(*_)

16.8.166 dwfpy.bindings.dwf_device_close_all

dwf_device_close_all(*_)

16.8.167 dwfpy.bindings.dwf_device_config_open

dwf_device_config_open(*_)

16.8.168 dwfpy.bindings.dwf_device_enable_set

dwf_device_enable_set(*_)

16.8.169 dwfpy.bindings.dwf_device_open

dwf_device_open(*_)

16.8.170 dwfpy.bindings.dwf_device_reset

dwf_device_reset(*_)

16.8.171 dwfpy.bindings.dwf_device_trigger_get

dwf_device_trigger_get(*_)

16.8.172 dwfpy.bindings.dwf_device_trigger_info

dwf_device_trigger_info(*_)

16.8.173 dwfpy.bindings.dwf_device_trigger_pc

dwf_device_trigger_pc(*_)

16.8.174 dwfpy.bindings.dwf_device_trigger_set

dwf_device_trigger_set(*_)

16.8.175 dwfpy.bindings.dwf_device_trigger_slope_info

dwf_device_trigger_slope_info(*_)

16.8.176 dwfpy.bindings.dwf_digital_can_polarity_set

dwf_digital_can_polarity_set(*_)

16.8.177 dwfpy.bindings.dwf_digital_can_rate_set

dwf_digital_can_rate_set(*_)

16.8.178 dwfpy.bindings.dwf_digital_can_reset

dwf_digital_can_reset(*_)

16.8.179 dwfpy.bindings.dwf_digital_can_rx

dwf_digital_can_rx(*_)

16.8.180 dwfpy.bindings.dwf_digital_can_rx_set

dwf_digital_can_rx_set(*_)

16.8.181 dwfpy.bindings.dwf_digital_can_tx

dwf_digital_can_tx(*_)

16.8.182 dwfpy.bindings.dwf_digital_can_tx_set

dwf_digital_can_tx_set(*_)

16.8.183 dwfpy.bindings.dwf_digital_i2c_clear

dwf_digital_i2c_clear(*_)

16.8.184 dwfpy.bindings.dwf_digital_i2c_rate_set

dwf_digital_i2c_rate_set(*_)

16.8.185 dwfpy.bindings.dwf_digital_i2c_read

dwf_digital_i2c_read(*_)

16.8.186 dwfpy.bindings.dwf_digital_i2c_read_nak_set

dwf_digital_i2c_read_nak_set(*_)

16.8.187 dwfpy.bindings.dwf_digital_i2c_reset

dwf_digital_i2c_reset(*_)

16.8.188 dwfpy.bindings.dwf_digital_i2c_scl_set

dwf_digital_i2c_scl_set(*_)

16.8.189 dwfpy.bindings.dwf_digital_i2c_sda_set

dwf_digital_i2c_sda_set(*_)

16.8.190 dwfpy.bindings.dwf_digital_i2c_write

dwf_digital_i2c_write(*_)

16.8.191 dwfpy.bindings.dwf_digital_i2c_write_one

dwf_digital_i2c_write_one(*_)

16.8.192 dwfpy.bindings.dwf_digital_i2c_write_read

dwf_digital_i2c_write_read(*_)

16.8.193 dwfpy.bindings.dwf_digital_in_acquisition_mode_get

dwf_digital_in_acquisition_mode_get(*_)

16.8.194 dwfpy.bindings.dwf_digital_in_acquisition_mode_info

dwf_digital_in_acquisition_mode_info(*_)

16.8.195 dwfpy.bindings.dwf_digital_in_acquisition_mode_set

dwf_digital_in_acquisition_mode_set(*_)

16.8.196 dwfpy.bindings.dwf_digital_in_bits_info

dwf_digital_in_bits_info(*_)

16.8.197 dwfpy.bindings.dwf_digital_in_buffer_size_get

dwf_digital_in_buffer_size_get(*_)

16.8.198 dwfpy.bindings.dwf_digital_in_buffer_size_info

dwf_digital_in_buffer_size_info(*_)

16.8.199 dwfpy.bindings.dwf_digital_in_buffer_size_set

dwf_digital_in_buffer_size_set(*_)

16.8.200 dwfpy.bindings.dwf_digital_in_clock_source_get

dwf_digital_in_clock_source_get(*_)

16.8.201 dwfpy.bindings.dwf_digital_in_clock_source_info

dwf_digital_in_clock_source_info(*_)

16.8.202 dwfpy.bindings.dwf_digital_in_clock_source_set

dwf_digital_in_clock_source_set(*_)

16.8.203 dwfpy.bindings.dwf_digital_in_configure

dwf_digital_in_configure(*_)

16.8.204 dwfpy.bindings.dwf_digital_in_divider_get

dwf_digital_in_divider_get(*_)

16.8.205 dwfpy.bindings.dwf_digital_in_divider_info

dwf_digital_in_divider_info(*_)

16.8.206 dwfpy.bindings.dwf_digital_in_divider_set

dwf_digital_in_divider_set(*_)

16.8.207 dwfpy.bindings.dwf_digital_in_input_order_set

dwf_digital_in_input_order_set(*_)

16.8.208 dwfpy.bindings.dwf_digital_in_internal_clock_info

dwf_digital_in_internal_clock_info(*_)

16.8.209 dwfpy.bindings.dwf_digital_in_reset

dwf_digital_in_reset(*_)

16.8.210 dwfpy.bindings.dwf_digital_in_sample_format_get

dwf_digital_in_sample_format_get(*_)

16.8.211 dwfpy.bindings.dwf_digital_in_sample_format_set

dwf_digital_in_sample_format_set(*_)

16.8.212 dwfpy.bindings.dwf_digital_in_sample_mode_get

dwf_digital_in_sample_mode_get(*_)

16.8.213 dwfpy.bindings.dwf_digital_in_sample_mode_info

dwf_digital_in_sample_mode_info(*_)

16.8.214 dwfpy.bindings.dwf_digital_in_sample_mode_set

dwf_digital_in_sample_mode_set(*_)

16.8.215 dwfpy.bindings.dwf_digital_in_sample_sensible_get

dwf_digital_in_sample_sensible_get(*_)

16.8.216 dwfpy.bindings.dwf_digital_in_sample_sensible_set

dwf_digital_in_sample_sensible_set(*_)

16.8.217 dwfpy.bindings.dwf_digital_in_status

dwf_digital_in_status(*_)

16.8.218 dwfpy.bindings.dwf_digital_in_status_auto_triggered

dwf_digital_in_status_auto_triggered(*_)

16.8.219 dwfpy.bindings.dwf_digital_in_status_data

dwf_digital_in_status_data(*_)

16.8.220 dwfpy.bindings.dwf_digital_in_status_data2

dwf_digital_in_status_data2(*_)

16.8.221 dwfpy.bindings.dwf_digital_in_status_index_write

dwf_digital_in_status_index_write(*_)

16.8.222 dwfpy.bindings.dwf_digital_in_status_noise2

dwf_digital_in_status_noise2(*_)

16.8.223 dwfpy.bindings.dwf_digital_in_status_record

dwf_digital_in_status_record(*_)

16.8.224 dwfpy.bindings.dwf_digital_in_status_samples_left

dwf_digital_in_status_samples_left(*_)

16.8.225 dwfpy.bindings.dwf_digital_in_status_samples_valid

dwf_digital_in_status_samples_valid(*_)

16.8.226 dwfpy.bindings.dwf_digital_in_trigger_auto_timeout_get

dwf_digital_in_trigger_auto_timeout_get(*_)

16.8.227 dwfpy.bindings.dwf_digital_in_trigger_auto_timeout_info

dwf_digital_in_trigger_auto_timeout_info(*_)

16.8.228 dwfpy.bindings.dwf_digital_in_trigger_auto_timeout_set

dwf_digital_in_trigger_auto_timeout_set(*_)

16.8.229 dwfpy.bindings.dwf_digital_in_trigger_count_set

dwf_digital_in_trigger_count_set(*_)

16.8.230 dwfpy.bindings.dwf_digital_in_trigger_get

dwf_digital_in_trigger_get(*_)

16.8.231 dwfpy.bindings.dwf_digital_in_trigger_info

dwf_digital_in_trigger_info(*_)

16.8.232 dwfpy.bindings.dwf_digital_in_trigger_length_set

dwf_digital_in_trigger_length_set(*_)

16.8.233 dwfpy.bindings.dwf_digital_in_trigger_match_set

dwf_digital_in_trigger_match_set(*_)

16.8.234 dwfpy.bindings.dwf_digital_in_trigger_position_get

dwf_digital_in_trigger_position_get(*_)

16.8.235 dwfpy.bindings.dwf_digital_in_trigger_position_info

dwf_digital_in_trigger_position_info(*_)

16.8.236 dwfpy.bindings.dwf_digital_in_trigger_position_set

dwf_digital_in_trigger_position_set(*_)

16.8.237 dwfpy.bindings.dwf_digital_in_trigger_prefill_get

dwf_digital_in_trigger_prefill_get(*_)

16.8.238 dwfpy.bindings.dwf_digital_in_trigger_prefill_set

dwf_digital_in_trigger_prefill_set(*_)

16.8.239 dwfpy.bindings.dwf_digital_in_trigger_reset_set

dwf_digital_in_trigger_reset_set(*_)

16.8.240 dwfpy.bindings.dwf_digital_in_trigger_set

dwf_digital_in_trigger_set(*_)

16.8.241 dwfpy.bindings.dwf_digital_in_trigger_slope_get

dwf_digital_in_trigger_slope_get(*_)

16.8.242 dwfpy.bindings.dwf_digital_in_trigger_slope_set

dwf_digital_in_trigger_slope_set(*_)

16.8.243 dwfpy.bindings.dwf_digital_in_trigger_source_get

dwf_digital_in_trigger_source_get(*_)

16.8.244 dwfpy.bindings.dwf_digital_in_trigger_source_set

dwf_digital_in_trigger_source_set(*_)

16.8.245 dwfpy.bindings.dwf_digital_io_configure

dwf_digital_io_configure(*_)

16.8.246 dwfpy.bindings.dwf_digital_io_input_info

dwf_digital_io_input_info(*_)

16.8.247 dwfpy.bindings.dwf_digital_io_input_info64

dwf_digital_io_input_info64(*_)

16.8.248 dwfpy.bindings.dwf_digital_io_input_status

dwf_digital_io_input_status(*_)

16.8.249 dwfpy.bindings.dwf_digital_io_input_status64

dwf_digital_io_input_status64(*_)

16.8.250 dwfpy.bindings.dwf_digital_io_output_enable_get

dwf_digital_io_output_enable_get(*_)

16.8.251 dwfpy.bindings.dwf_digital_io_output_enable_get64

dwf_digital_io_output_enable_get64(*_)

16.8.252 dwfpy.bindings.dwf_digital_io_output_enable_info

dwf_digital_io_output_enable_info(*_)

16.8.253 dwfpy.bindings.dwf_digital_io_output_enable_info64

dwf_digital_io_output_enable_info64(*_)

16.8.254 dwfpy.bindings.dwf_digital_io_output_enable_set

dwf_digital_io_output_enable_set(*_)

16.8.255 dwfpy.bindings.dwf_digital_io_output_enable_set64

dwf_digital_io_output_enable_set64(*_)

16.8.256 dwfpy.bindings.dwf_digital_io_output_get

dwf_digital_io_output_get(*_)

16.8.257 dwfpy.bindings.dwf_digital_io_output_get64

dwf_digital_io_output_get64(*_)

16.8.258 dwfpy.bindings.dwf_digital_io_output_info

dwf_digital_io_output_info(*_)

16.8.259 dwfpy.bindings.dwf_digital_io_output_info64

dwf_digital_io_output_info64(*_)

16.8.260 dwfpy.bindings.dwf_digital_io_output_set

dwf_digital_io_output_set(*_)

16.8.261 dwfpy.bindings.dwf_digital_io_output_set64

dwf_digital_io_output_set64(*_)

16.8.262 dwfpy.bindings.dwf_digital_io_reset

dwf_digital_io_reset(*_)

16.8.263 dwfpy.bindings.dwf_digital_io_status

dwf_digital_io_status(*_)

16.8.264 dwfpy.bindings.dwf_digital_out_configure

dwf_digital_out_configure(*_)

16.8.265 dwfpy.bindings.dwf_digital_out_count

dwf_digital_out_count(*_)

16.8.266 dwfpy.bindings.dwf_digital_out_counter_get

dwf_digital_out_counter_get(*_)

16.8.267 dwfpy.bindings.dwf_digital_out_counter_info

dwf_digital_out_counter_info(*_)

16.8.268 dwfpy.bindings.dwf_digital_out_counter_init_get

dwf_digital_out_counter_init_get(*_)

16.8.269 dwfpy.bindings.dwf_digital_out_counter_init_set

dwf_digital_out_counter_init_set(*_)

16.8.270 dwfpy.bindings.dwf_digital_out_counter_set

dwf_digital_out_counter_set(*_)

16.8.271 dwfpy.bindings.dwf_digital_out_data_info

dwf_digital_out_data_info(*_)

16.8.272 dwfpy.bindings.dwf_digital_out_data_set

dwf_digital_out_data_set(*_)

16.8.273 dwfpy.bindings.dwf_digital_out_divider_get

dwf_digital_out_divider_get(*_)

16.8.274 dwfpy.bindings.dwf_digital_out_divider_info

dwf_digital_out_divider_info(*_)

16.8.275 dwfpy.bindings.dwf_digital_out_divider_init_get

dwf_digital_out_divider_init_get(*_)

16.8.276 dwfpy.bindings.dwf_digital_out_divider_init_set

dwf_digital_out_divider_init_set(*_)

16.8.277 dwfpy.bindings.dwf_digital_out_divider_set

dwf_digital_out_divider_set(*_)

16.8.278 dwfpy.bindings.dwf_digital_out_enable_get

dwf_digital_out_enable_get(*_)

16.8.279 dwfpy.bindings.dwf_digital_out_enable_set

dwf_digital_out_enable_set(*_)

16.8.280 dwfpy.bindings.dwf_digital_out_idle_get

dwf_digital_out_idle_get(*_)

16.8.281 dwfpy.bindings.dwf_digital_out_idle_info

dwf_digital_out_idle_info(*_)

16.8.282 dwfpy.bindings.dwf_digital_out_idle_set

dwf_digital_out_idle_set(*_)

16.8.283 dwfpy.bindings.dwf_digital_out_internal_clock_info

dwf_digital_out_internal_clock_info(*_)

16.8.284 dwfpy.bindings.dwf_digital_out_output_get

dwf_digital_out_output_get(*_)

16.8.285 dwfpy.bindings.dwf_digital_out_output_info

dwf_digital_out_output_info(*_)

16.8.286 dwfpy.bindings.dwf_digital_out_output_set

dwf_digital_out_output_set(*_)

16.8.287 dwfpy.bindings.dwf_digital_out_repeat_get

dwf_digital_out_repeat_get(*_)

16.8.288 dwfpy.bindings.dwf_digital_out_repeat_info

dwf_digital_out_repeat_info(*_)

16.8.289 dwfpy.bindings.dwf_digital_out_repeat_set

dwf_digital_out_repeat_set(*_)

16.8.290 dwfpy.bindings.dwf_digital_out_repeat_status

dwf_digital_out_repeat_status(*_)

16.8.291 dwfpy.bindings.dwf_digital_out_repeat_trigger_get

dwf_digital_out_repeat_trigger_get(*_)

16.8.292 dwfpy.bindings.dwf_digital_out_repeat_trigger_set

dwf_digital_out_repeat_trigger_set(*_)

16.8.293 dwfpy.bindings.dwf_digital_out_reset

dwf_digital_out_reset(*_)

16.8.294 dwfpy.bindings.dwf_digital_out_run_get

dwf_digital_out_run_get(*_)

16.8.295 dwfpy.bindings.dwf_digital_out_run_info

dwf_digital_out_run_info(*_)

16.8.296 dwfpy.bindings.dwf_digital_out_run_set

dwf_digital_out_run_set(*_)

16.8.297 dwfpy.bindings.dwf_digital_out_run_status

dwf_digital_out_run_status(*_)

16.8.298 dwfpy.bindings.dwf_digital_out_status

dwf_digital_out_status(*_)

16.8.299 dwfpy.bindings.dwf_digital_out_trigger_slope_get

dwf_digital_out_trigger_slope_get(*_)

16.8.300 dwfpy.bindings.dwf_digital_out_trigger_slope_set

dwf_digital_out_trigger_slope_set(*_)

16.8.301 dwfpy.bindings.dwf_digital_out_trigger_source_get

dwf_digital_out_trigger_source_get(*_)

16.8.302 dwfpy.bindings.dwf_digital_out_trigger_source_set

dwf_digital_out_trigger_source_set(*_)

16.8.303 dwfpy.bindings.dwf_digital_out_type_get

dwf_digital_out_type_get(*_)

16.8.304 dwfpy.bindings.dwf_digital_out_type_info

dwf_digital_out_type_info(*_)

16.8.305 dwfpy.bindings.dwf_digital_out_type_set

dwf_digital_out_type_set(*_)

16.8.306 dwfpy.bindings.dwf_digital_out_wait_get

dwf_digital_out_wait_get(*_)

16.8.307 dwfpy.bindings.dwf_digital_out_wait_info

dwf_digital_out_wait_info(*_)

16.8.308 dwfpy.bindings.dwf_digital_out_wait_set

dwf_digital_out_wait_set(*_)

16.8.309 dwfpy.bindings.dwf_digital_spi_clock_set

dwf_digital_spi_clock_set(*_)

16.8.310 dwfpy.bindings.dwf_digital_spi_data_set

dwf_digital_spi_data_set(*_)

16.8.311 dwfpy.bindings.dwf_digital_spi_frequency_set

dwf_digital_spi_frequency_set(*_)

16.8.312 dwfpy.bindings.dwf_digital_spi_mode_set

dwf_digital_spi_mode_set(*_)

16.8.313 dwfpy.bindings.dwf_digital_spi_order_set

dwf_digital_spi_order_set(*_)

16.8.314 dwfpy.bindings.dwf_digital_spi_read

dwf_digital_spi_read(*_)

16.8.315 dwfpy.bindings.dwf_digital_spi_read16

dwf_digital_spi_read16(*_)

16.8.316 dwfpy.bindings.dwf_digital_spi_read32

dwf_digital_spi_read32(*_)

16.8.317 dwfpy.bindings.dwf_digital_spi_read_one

dwf_digital_spi_read_one(*_)

16.8.318 dwfpy.bindings.dwf_digital_spi_reset

dwf_digital_spi_reset(*_)

16.8.319 dwfpy.bindings.dwf_digital_spi_select

dwf_digital_spi_select(*_)

16.8.320 dwfpy.bindings.dwf_digital_spi_write

dwf_digital_spi_write(*_)

16.8.321 dwfpy.bindings.dwf_digital_spi_write16

dwf_digital_spi_write16(*_)

16.8.322 dwfpy.bindings.dwf_digital_spi_write32

dwf_digital_spi_write32(*_)

16.8.323 dwfpy.bindings.dwf_digital_spi_write_one

dwf_digital_spi_write_one(*_)

16.8.324 dwfpy.bindings.dwf_digital_spi_write_read

dwf_digital_spi_write_read(*_)

16.8.325 dwfpy.bindings.dwf_digital_spi_write_read16

dwf_digital_spi_write_read16(*_)

16.8.326 dwfpy.bindings.dwf_digital_spi_write_read32

dwf_digital_spi_write_read32(*_)

16.8.327 dwfpy.bindings.dwf_digital_uart_bits_set

dwf_digital_uart_bits_set(*_)

16.8.328 dwfpy.bindings.dwf_digital_uart_parity_set

dwf_digital_uart_parity_set(*_)

16.8.329 dwfpy.bindings.dwf_digital_uart_rate_set

dwf_digital_uart_rate_set(*_)

16.8.330 dwfpy.bindings.dwf_digital_uart_reset

dwf_digital_uart_reset(*_)

16.8.331 dwfpy.bindings.dwf_digital_uart_rx

dwf_digital_uart_rx(*_)

16.8.332 dwfpy.bindings.dwf_digital_uart_rx_set

dwf_digital_uart_rx_set(*_)

16.8.333 dwfpy.bindings.dwf_digital_uart_stop_set

dwf_digital_uart_stop_set(*_)

16.8.334 dwfpy.bindings.dwf_digital_uart_tx

dwf_digital_uart_tx(*_)

16.8.335 dwfpy.bindings.dwf_digital_uart_tx_set

dwf_digital_uart_tx_set(*_)

16.8.336 dwfpy.bindings.dwf_enum

dwf_enum(*_)

16.8.337 dwfpy.bindings.dwf_enum_config

dwf_enum_config(*_)

16.8.338 dwfpy.bindings.dwf_enum_config_info

dwf_enum_config_info(*_)

16.8.339 dwfpy.bindings.dwf_enum_config_info_str

dwf_enum_config_info_str(*_)

16.8.340 dwfpy.bindings.dwf_enum_device_is_opened

dwf_enum_device_is_opened(*_)

16.8.341 dwfpy.bindings.dwf_enum_device_name

dwf_enum_device_name(*_)

16.8.342 dwfpy.bindings.dwf_enum_device_type

dwf_enum_device_type(*_)

16.8.343 dwfpy.bindings.dwf_enum_sn

dwf_enum_sn(*_)

16.8.344 dwfpy.bindings.dwf_enum_user_name

dwf_enum_user_name(*_)

16.8.345 dwfpy.bindings.dwf_get_last_error

dwf_get_last_error(*_)

16.8.346 dwfpy.bindings.dwf_get_last_error_msg

dwf_get_last_error_msg(*_)

16.8.347 dwfpy.bindings.dwf_get_version

dwf_get_version(*_)

16.8.348 dwfpy.bindings.set_error_handler

set_error_handler(*handler*)

Set the error handle for checking the return code of the DWF API.

16.9 dwfpy.configuration

Configuration set for Digilent WaveForms devices.

Classes

Configuration(index)	Configuration set for Digilent WaveForms devices.

16.9.1 dwfpy.configuration.Configuration

class Configuration(*index*)

Bases: object

Configuration set for Digilent WaveForms devices.

Methods

Attributes

analog_in_buffer_size	Gets the Analog Input buffer size.
analog_in_channel_count	Gets the total number of Analog Input channels.
analog_io_channel_count	Gets the total number of Analog IO channels.
analog_out_buffer_size	Gets the Analog Output buffer size.
analog_out_channel_count	Gets the total number of Analog Output channels.
digital_in_buffer_size	Gets the Digital Input buffer size.
digital_in_channel_count	Gets the total number of Digital Input channels.
digital_io_channel_count	Gets the total number of Digital IO channels.
digital_out_buffer_size	Gets the Digital Output buffer size.
digital_out_channel_count	Gets the total number of Digital Output channels.
text_info	Gets an extra configuration information string.

property analog_in_buffer_size: int

Gets the Analog Input buffer size.

Return type
int

property analog_in_channel_count: int

Gets the total number of Analog Input channels.

Return type
int

property analog_io_channel_count: int

Gets the total number of Analog IO channels.

Return type
int

property analog_out_buffer_size: int

Gets the Analog Output buffer size.

Return type
int

property analog_out_channel_count: int

Gets the total number of Analog Output channels.

Return type
int

property digital_in_buffer_size: int

Gets the Digital Input buffer size.

Return type
int

property digital_in_channel_count: int

Gets the total number of Digital Input channels.

Return type
int

property digital_io_channel_count: int

Gets the total number of Digital IO channels.

Return type
int

property digital_out_buffer_size: int

Gets the Digital Output buffer size.

Return type
int

property digital_out_channel_count: int

Gets the total number of Digital Output channels.

Return type
int

property text_info: str

Gets an extra configuration information string.

Return type
str

16.10 dwfpy.constants

Constants used by Digilent WaveForms API.

Classes

AcquisitionMode(value)	Acquisition mode for analog and digital instruments.
AnalogImpedance(value)	Analog impedance measurement index.
AnalogInputCoupling(value)	Input coupling for analog input instruments.
AnalogOutputIdle(value)	Idle output options of analog output instruments.
AnalogOutputMode(value)	Generator mode of analog output instruments.
AnalogOutputNode(value)	Analog node type of analog output instruments.
ChannelNodeType(value)	Channel node type.
DeviceId(value)	Device identifier.
DeviceType(value)	Device filter type.
DigitalInputClockSource(value)	Clock source for digital input instruments.
DigitalInputSampleMode(value)	Sample mode for digital input instruments.
DigitalOutputIdle(value)	Idle state of a digital output channel.
DigitalOutputMode(value)	Output pin mode of a digital output channel.
DigitalOutputType(value)	Output type of a digital output channel.
DmmMode(value)	DMM mode.
Error(value)	DWF API error codes.
FilterMode(value)	Acquisition filter for analog input channels.
Function(value)	Function type for analog output instruments.
GlobalParameter(value)	Global parameter.
Status(value)	Status of instrument state machine.
TriggerLengthCondition(value)	Trigger length condition.
TriggerSlope(value)	Trigger slope.
TriggerSource(value)	Trigger input source.
TriggerType(value)	Trigger type.
Window(value)	FFT window function.

16.10.1 dwfpy.constants.AcquisitionMode

class AcquisitionMode(*value*)

Bases: IntEnum

Acquisition mode for analog and digital instruments.

Attributes

SINGLE
SCAN_SHIFT
SCAN_SCREEN
RECORD
OVERS
SINGLE1
RECORD2

16.10.2 dwfpy.constants.AnalogImpedance

class AnalogImpedance(*value*)

Bases: IntEnum

Analog impedance measurement index.

Attributes

IMPEDANCE
IMPEDANCE_PHASE
RESISTANCE
REACTANCE
ADMITTANCE
ADMITTANCE_PHASE
CONDUCTANCE
SUSCEPTANCE
SERIES_CAPACITANCE
PARALLEL_CAPACITANCE
SERIES_INDUCTANCE
PARALLEL_INDUCTANCE
DISSIPATION
QUALITY
VRMS
VREAL
VIMAG
IRMS
IREAL
IIMAG

16.10.3 dwfpy.constants.AnalogInputCoupling

class AnalogInputCoupling(*value*)

Bases: IntEnum

Input coupling for analog input instruments.

Attributes

DC
AC

16.10.4 dwfpy.constants.AnalogOutputIdle

class AnalogOutputIdle(*value*)

Bases: IntEnum

Idle output options of analog output instruments.

Attributes

DISABLE
OFFSET
INITIAL

16.10.5 dwfpy.constants.AnalogOutputMode

class AnalogOutputMode(*value*)

Bases: IntEnum

Generator mode of analog output instruments.

Attributes

VOLTAGE
CURRENT

16.10.6 dwfpy.constants.AnalogOutputNode

class AnalogOutputNode(*value*)

Bases: IntEnum

Analog node type of analog output instruments.

Attributes

CARRIER
FM
AM

16.10.7 dwfpy.constants.ChannelNodeType

class ChannelNodeType(*value*)

Bases: IntEnum

Channel node type.

Attributes

ENABLE
VOLTAGE
CURRENT
POWER
TEMPERATURE
DMM
RANGE
MEASURE
TIME
FREQUENCY
RESISTANCE

16.10.8 dwfpy.constants.DeviceId

class DeviceId(*value*)

Bases: IntEnum

Device identifier.

Attributes

ELECTRONICS_EXPLORER
ANALOG_DISCOVERY
ANALOG_DISCOVERY2
DIGITAL_DISCOVERY
ADP3X50
ADP5250

16.10.9 dwfpy.constants.DeviceType

class DeviceType(*value*)

Bases: IntEnum

Device filter type.

Attributes

USB
NETWORK
AXI
REMOTE
SOUND_CARD
DEMO

16.10.10 dwfpy.constants.DigitalInputClockSource

class DigitalInputClockSource(*value*)

Bases: IntEnum

Clock source for digital input instruments.

Attributes

INTERNAL
EXTERNAL
EXTERNAL2

16.10.11 dwfpy.constants.DigitalInputSampleMode

class DigitalInputSampleMode(*value*)

Bases: IntEnum

Sample mode for digital input instruments.

Attributes

SIMPLE
NOISE

16.10.12 dwfpy.constants.DigitalOutputIdle

class DigitalOutputIdle(*value*)

Bases: IntEnum

Idle state of a digital output channel.

Attributes

INIT
LOW
HIGH
ZET

16.10.13 dwfpy.constants.DigitalOutputMode

class DigitalOutputMode(*value*)

Bases: IntEnum

Output pin mode of a digital output channel.

Attributes

PUSH_PULL
OPEN_DRAIN
OPEN_SOURCE
THREE_STATE

16.10.14 dwfpy.constants.DigitalOutputType

class DigitalOutputType(*value*)

Bases: IntEnum

Output type of a digital output channel.

Attributes

PULSE
CUSTOM
RANDOM
ROM
STATE
PLAY

16.10.15 dwfpy.constants.DmmMode

class DmmMode(*value*)

Bases: IntEnum

DMM mode.

Attributes

RESISTANCE
CONTINUITY
DIODE
DC_VOLTAGE
AC_VOLTAGE
DC_CURRENT
AC_CURRENT
DC_LOW_CURRENT
AC_LOW_CURRENT
TEMPERATURE

16.10.16 dwfpy.constants.Error

class Error(*value*)

Bases: IntEnum

DWF API error codes.

Attributes

NO_ERROR
UNKNOWN_ERROR
API_LOCK_TIMEOUT
ALREADY_OPENED
NOT_SUPPORTED
INVALID_PARAMETER0
INVALID_PARAMETER1
INVALID_PARAMETER2
INVALID_PARAMETER3
INVALID_PARAMETER4

16.10.17 dwfpy.constants.FilterMode

class FilterMode(*value*)

Bases: IntEnum

Acquisition filter for analog input channels.

Attributes

DECIMATE
AVERAGE
MIN_MAX

16.10.18 dwfpy.constants.Function

class Function(*value*)

Bases: IntEnum

Function type for analog output instruments.

Attributes

DC
SINE
SQUARE
TRIANGLE
RAMP_UP
RAMP_DOWN
NOISE
PULSE
TRAPEZIUM
SINE_POWER
CUSTOM_PATTERN
PLAY_PATTERN
CUSTOM
PLAY

16.10.19 dwfpy.constants.GlobalParameter

class GlobalParameter(*value*)

Bases: IntEnum

Global parameter.

Attributes

USB_POWER
LED_BRIGHTNESS
ON_CLOSE
AUDIO_OUT
USB_LIMIT
ANALOG_OUT
FREQUENCY
EXT_FREQ
CLOCK_MODE

16.10.20 dwfpy.constants.Status

class Status(*value*)

Bases: IntEnum

Status of instrument state machine.

Attributes

READY
ARMED
DONE
RUNNING
TRIGGERED
CONFIG
PREFILL
WAIT

16.10.21 dwfpy.constants.TriggerLengthCondition

class TriggerLengthCondition(*value*)

Bases: IntEnum

Trigger length condition.

Attributes

LESS
TIMEOUT
MORE

16.10.22 dwfpy.constants.TriggerSlope

class TriggerSlope(*value*)

Bases: IntEnum

Trigger slope.

Attributes

RISE
FALL
EITHER

16.10.23 dwfpy.constants.TriggerSource

class TriggerSource(*value*)

Bases: IntEnum

Trigger input source.

Attributes

NONE
PC
DETECTOR_ANALOG_IN
DETECTOR_DIGITAL_IN
ANALOG_IN
DIGITAL_IN
DIGITAL_OUT
ANALOG_OUT1
ANALOG_OUT2
ANALOG_OUT3
ANALOG_OUT4
EXTERNAL1
EXTERNAL2
EXTERNAL3
EXTERNAL4
HIGH
LOW
CLOCK

16.10.24 dwfpy.constants.TriggerType

class TriggerType(*value*)

Bases: IntEnum

Trigger type.

Attributes

EDGE
PULSE
TRANSITION
WINDOW

16.10.25 dwfpy.constants.Window

class Window(*value*)

Bases: IntEnum

FFT window function.

Attributes

RECTANGULAR
TRIANGULAR
HAMMING
HANN
COSINE
BLACKMAN_HARRIS
FLATTOP
KAISER

16.11 dwfpy.device

Support for Digilent WaveForms devices.

Classes

AnalogDiscovery([configuration, ...])	Digilent Analog Discovery device.
AnalogDiscovery2([configuration, ...])	Digilent Analog Discovery 2 device.
Device([configuration, serial_number, ...])	Generic Digilent WaveForms device.
DeviceBase([configuration, serial_number, ...])	Base class for Digilent WaveForms devices.
DigitalDiscovery([configuration, ...])	Digilent Digital Discovery device.
ElectronicsExplorer([configuration, ...])	Digilent Electronics Explorer device.

16.11.1 dwfpy.device.AnalogDiscovery

class AnalogDiscovery(*configuration=None*, *serial_number=None*, *device_type=None*, *device_index=None*)

Bases: *DeviceBase*

Digilent Analog Discovery device.

Parameters

- **configuration** – Select the active configuration.
- **serial_number** – Filter devices by serial number.
- **device_id** – Filter devices by device ID.
- **device_type** – Filter devices by device type.
- **device_index** – Filter devices by device index.

Methods

close	Closes the device.
get_parameter	Gets a device parameter.
get_trigger	Gets the configured trigger setting for a trigger I/O pin.
open	Opens the device.
reset	Resets and configures all device and instrument parameters to default values.
set_parameter	Sets a device parameter.
set_trigger	Sets the trigger I/O pin with a specific TriggerSource option.
trigger_pc	Generates one pulse on the PC trigger line.

Attributes

analog_input	Gets the Analog Input module (Oscilloscope).
analog_io	Gets the Analog IO module.
analog_output	Gets the Analog Output module (Arbitrary Waveform Generator).
application	Gets the WaveForms application.
auto_configure	Gets or sets a value indicating to automatically configure the device when parameters are changed.
auto_reset	Gets or sets a value indicating to reset the device automatically on exit.
configuration	Gets the selected configuration index.
configurations	Returns a list of device configurations.
digital_input	Gets the Digital Input module (Logic Analyzer).
digital_io	Gets the Digital IO module.
digital_output	Gets the Digital Output module (Pattern Generator).
handle	Gets a handle to the device.
id	Gets the device ID.
is_open	Returns true if the device has been opened.
name	Gets the device name.
protocols	Gets the Digital Protocols module.
revision	Gets the device revision.
serial_number	Gets the 12-digit, unique device serial number.
supplies	Gets the power supplies.
temperature	Gets the temperature of the device.
trigger_info	Gets the supported trigger source options for the global trigger bus.
trigger_slope_info	Gets the supported trigger slopes.
usb_current	Gets the USB line current.
usb_voltage	Gets the USB line voltage.
user_name	Gets the user-defined device name.

class Supplies(*device*)

Bases: `object`

The power supplies.

class Negative(*device*)

Bases: `object`

The negative power supply.

property enabled: bool

Enables the negative power supply.

Return type
`bool`

setup(*enabled=True*)

Sets up the negative power supply.

enabled
[bool, optional] If True, then the power supply is enabled (default True).

Return type
`None`

class Positive(*device*)

Bases: object

The positive power supply.

property enabled: bool

Enables or disables the positive power supply.

Return type
bool

setup(*enabled=True*)

Sets up the positive power supply.

enabled
[bool, optional] If True, then the power supply is enabled (default True).

Return type
None

property master_enable: bool

Gets or sets the master enable switch.

Return type
bool

property master_enable_status: bool

Gets the master enable status.

Return type
bool

property negative: *Negative*

Gets the negative power supply.

Return type
Negative

property positive: *Positive*

Gets the positive power supply.

Return type
Positive

property regulator_current: float

Gets the current taken by the supply regulators.

Return type
float

property regulator_voltage: float

Gets the voltage input for the supply regulators.

Return type
float

property analog_input: *AnalogInput*

Gets the Analog Input module (Oscilloscope).

Return type
AnalogInput

property analog_io: *AnalogIo*

Gets the Analog IO module.

Return type
AnalogIo

property analog_output: *AnalogOutput*

Gets the Analog Output module (Arbitrary Waveform Generator).

Return type
AnalogOutput

property application: *Application*

Gets the WaveForms application.

Return type
Application

property auto_configure: int

Gets or sets a value indicating to automatically configure the device when parameters are changed.

Return type
int

property auto_reset: bool

Gets or sets a value indicating to reset the device automatically on exit.

Return type
bool

close()

Closes the device.

Return type
None

property configuration: Optional[Union[int, str]]

Gets the selected configuration index.

Return type
Union[int, str, None]

property configurations: Tuple[*Configuration*, ...]

Returns a list of device configurations.

Return type
Tuple[*Configuration*, ...]

property digital_input: *DigitalInput*

Gets the Digital Input module (Logic Analyzer).

Return type
DigitalInput

property digital_io: *DigitalIo*

Gets the Digital IO module.

Return type
DigitalIo

property digital_output: *DigitalOutput*

Gets the Digital Output module (Pattern Generator).

Return type
DigitalOutput

get_parameter(*parameter*)

Gets a device parameter.

Return type
int

get_trigger(*pin*)

Gets the configured trigger setting for a trigger I/O pin.

Return type
TriggerSource

property handle: object

Gets a handle to the device.

Return type
object

property id: *DeviceId*

Gets the device ID.

Return type
DeviceId

property is_open: bool

Returns true if the device has been opened.

Return type
bool

property name: str

Gets the device name.

Return type
str

open()

Opens the device.

Return type
None

property protocols: *Protocols*

Gets the Digital Protocols module.

Return type
Protocols

reset()

Resets and configures all device and instrument parameters to default values.

Return type
None

property revision: str

Gets the device revision.

Return type
str

property serial_number: str

Gets the 12-digit, unique device serial number.

Return type
str

set_parameter(*parameter*, *value*)

Sets a device parameter.

Return type
None

set_trigger(*pin*, *trigger_source*)

Sets the trigger I/O pin with a specific TriggerSource option.

Return type
None

property supplies: *Supplies*

Gets the power supplies.

Return type
Supplies

property temperature: float

Gets the temperature of the device.

Return type
float

property trigger_info: Tuple[*TriggerSource*, ...]

Gets the supported trigger source options for the global trigger bus.

Return type
Tuple[*TriggerSource*, ...]

trigger_pc()

Generates one pulse on the PC trigger line.

Return type
None

property trigger_slope_info: Tuple[*TriggerSlope*, ...]

Gets the supported trigger slopes.

Return type
Tuple[*TriggerSlope*, ...]

property usb_current: float

Gets the USB line current.

Return type
float

property usb_voltage: float

Gets the USB line voltage.

Return type
float

property user_name: str

Gets the user-defined device name.

Return type
str

16.11.2 dwfpy.device.AnalogDiscovery2

class AnalogDiscovery2(*configuration=None*, *serial_number=None*, *device_type=None*, *device_index=None*)

Bases: *DeviceBase*

Digilent Analog Discovery 2 device.

Parameters

- **configuration** – Select the active configuration.
- **serial_number** – Filter devices by serial number.
- **device_id** – Filter devices by device ID.
- **device_type** – Filter devices by device type.
- **device_index** – Filter devices by device index.

Methods

close	Closes the device.
get_parameter	Gets a device parameter.
get_trigger	Gets the configured trigger setting for a trigger I/O pin.
open	Opens the device.
reset	Resets and configures all device and instrument parameters to default values.
set_parameter	Sets a device parameter.
set_trigger	Sets the trigger I/O pin with a specific TriggerSource option.
trigger_pc	Generates one pulse on the PC trigger line.

Attributes

analog_input	Gets the Analog Input module (Oscilloscope).
analog_io	Gets the Analog IO module.
analog_output	Gets the Analog Output module (Arbitrary Waveform Generator).
application	Gets the WaveForms application.
auto_configure	Gets or sets a value indicating to automatically configure the device when parameters are changed.
auto_reset	Gets or sets a value indicating to reset the device automatically on exit.
aux_current	Gets the AUX line current.
aux_voltage	Gets the AUX line voltage.
configuration	Gets the selected configuration index.
configurations	Returns a list of device configurations.
digital_input	Gets the Digital Input module (Logic Analyzer).
digital_io	Gets the Digital IO module.
digital_output	Gets the Digital Output module (Pattern Generator).
handle	Gets a handle to the device.
id	Gets the device ID.
is_open	Returns true if the device has been opened.
name	Gets the device name.
protocols	Gets the Digital Protocols module.
revision	Gets the device revision.
serial_number	Gets the 12-digit, unique device serial number.
supplies	Gets the power supplies.
temperature	Gets the temperature of the device.
trigger_info	Gets the supported trigger source options for the global trigger bus.
trigger_slope_info	Gets the supported trigger slopes.
usb_current	Gets the USB line current.
usb_voltage	Gets the USB line voltage.
user_name	Gets the user-defined device name.

class Supplies(*device*)

Bases: `object`

The power supplies.

class Negative(*device*)

Bases: `object`

The negative power supply.

property enabled: bool

Enables the negative power supply.

Return type
`bool`

setup(*voltage*, *enabled=True*)

Sets up the negative power supply.

voltage
[float] The output voltage (must be a negative value).

enabled
[bool, optional] If True, then the power supply is enabled (default True).

Return type
None

property voltage: float

Gets or sets the voltage of the negative power supply.

Return type
float

class Positive(*device*)

Bases: object

The positive power supply.

property enabled: bool

Enables or disables the positive power supply.

Return type
bool

setup(*voltage*, *enabled=True*)

Sets up the positive power supply.

voltage
[float] The output voltage.

enabled
[bool, optional] If True, then the power supply is enabled (default True).

Return type
None

property voltage: float

Gets or sets the voltage of the positive power supply.

Return type
float

property master_enable: bool

Gets or sets the master enable switch.

Return type
bool

property master_enable_status: bool

Gets the master enable status.

Return type
bool

property negative: *Negative*

Gets the negative power supply.

Return type
Negative

property positive: *Positive*

Gets the positive power supply.

Return type
Positive

property analog_input: *AnalogInput*

Gets the Analog Input module (Oscilloscope).

Return type
AnalogInput

property analog_io: *AnalogIo*

Gets the Analog IO module.

Return type
AnalogIo

property analog_output: *AnalogOutput*

Gets the Analog Output module (Arbitrary Waveform Generator).

Return type
AnalogOutput

property application: *Application*

Gets the WaveForms application.

Return type
Application

property auto_configure: int

Gets or sets a value indicating to automatically configure the device when parameters are changed.

Return type
int

property auto_reset: bool

Gets or sets a value indicating to reset the device automatically on exit.

Return type
bool

property aux_current: float

Gets the AUX line current.

Return type
float

property aux_voltage: float

Gets the AUX line voltage.

Return type
float

close()

Closes the device.

Return type
None

property configuration: Optional[Union[int, str]]

Gets the selected configuration index.

Return type
Union[int, str, None]

property configurations: Tuple[*Configuration*, ...]

Returns a list of device configurations.

Return type
Tuple[*Configuration*, ...]

property digital_input: *DigitalInput*

Gets the Digital Input module (Logic Analyzer).

Return type
DigitalInput

property digital_io: *DigitalIo*

Gets the Digital IO module.

Return type
DigitalIo

property digital_output: *DigitalOutput*

Gets the Digital Output module (Pattern Generator).

Return type
DigitalOutput

get_parameter(*parameter*)

Gets a device parameter.

Return type
int

get_trigger(*pin*)

Gets the configured trigger setting for a trigger I/O pin.

Return type
TriggerSource

property handle: object

Gets a handle to the device.

Return type
object

property id: *DeviceId*

Gets the device ID.

Return type
DeviceId

property is_open: bool

Returns true if the device has been opened.

Return type
bool

property name: str

Gets the device name.

Return type
str

open()

Opens the device.

Return type
None

property protocols: *Protocols*

Gets the Digital Protocols module.

Return type
Protocols

reset()

Resets and configures all device and instrument parameters to default values.

Return type
None

property revision: str

Gets the device revision.

Return type
str

property serial_number: str

Gets the 12-digit, unique device serial number.

Return type
str

set_parameter(*parameter*, *value*)

Sets a device parameter.

Return type
None

set_trigger(*pin*, *trigger_source*)

Sets the trigger I/O pin with a specific TriggerSource option.

Return type
None

property supplies: *Supplies*

Gets the power supplies.

Return type
Supplies

property temperature: float

Gets the temperature of the device.

Return type
float

property trigger_info: Tuple[*TriggerSource*, ...]

Gets the supported trigger source options for the global trigger bus.

Return type
Tuple[*TriggerSource*, ...]

trigger_pc()

Generates one pulse on the PC trigger line.

Return type
None

property trigger_slope_info: Tuple[*TriggerSlope*, ...]

Gets the supported trigger slopes.

Return type
Tuple[*TriggerSlope*, ...]

property usb_current: float

Gets the USB line current.

Return type
float

property usb_voltage: float

Gets the USB line voltage.

Return type
float

property user_name: str

Gets the user-defined device name.

Return type
str

16.11.3 dwfpy.device.Device

class Device(*configuration=None*, *serial_number=None*, *device_id=None*, *device_type=None*, *device_index=None*)

Bases: *DeviceBase*

Generic Digilent WaveForms device.

Parameters

- **configuration** (Union[int, str, None]) – Select the active configuration.
- **serial_number** (Optional[str]) – Filter devices by serial number.
- **device_id** (Optional[int]) – Filter devices by device ID.
- **device_type** (Optional[int]) – Filter devices by device type.
- **device_index** (Union[int, _EnumeratedIndex, None]) – Filter devices by device index.

Methods

close	Closes the device.
close_all	Closes all opened devices by the calling process.
enumerate	Enumerates all devices.
get_parameter	Gets a device parameter.
get_trigger	Gets the configured trigger setting for a trigger I/O pin.
open	Opens the device.
reset	Resets and configures all device and instrument parameters to default values.
set_parameter	Sets a device parameter.
set_trigger	Sets the trigger I/O pin with a specific TriggerSource option.
trigger_pc	Generates one pulse on the PC trigger line.

Attributes

analog_input	Gets the Analog Input module (Oscilloscope).
analog_io	Gets the Analog IO module.
analog_output	Gets the Analog Output module (Arbitrary Waveform Generator).
application	Gets the WaveForms application.
auto_configure	Gets or sets a value indicating to automatically configure the device when parameters are changed.
auto_reset	Gets or sets a value indicating to reset the device automatically on exit.
configuration	Gets the selected configuration index.
configurations	Returns a list of device configurations.
digital_input	Gets the Digital Input module (Logic Analyzer).
digital_io	Gets the Digital IO module.
digital_output	Gets the Digital Output module (Pattern Generator).
handle	Gets a handle to the device.
id	Gets the device ID.
is_open	Returns true if the device has been opened.
name	Gets the device name.
protocols	Gets the Digital Protocols module.
revision	Gets the device revision.
serial_number	Gets the 12-digit, unique device serial number.
trigger_info	Gets the supported trigger source options for the global trigger bus.
trigger_slope_info	Gets the supported trigger slopes.
user_name	Gets the user-defined device name.

property analog_input: *AnalogInput*

Gets the Analog Input module (Oscilloscope).

Return type
AnalogInput

property analog_io: *AnalogIo*

Gets the Analog IO module.

Return type
AnalogIo

property analog_output: *AnalogOutput*

Gets the Analog Output module (Arbitrary Waveform Generator).

Return type
AnalogOutput

property application: *Application*

Gets the WaveForms application.

Return type
Application

property auto_configure: int

Gets or sets a value indicating to automatically configure the device when parameters are changed.

Return type
int

property auto_reset: bool

Gets or sets a value indicating to reset the device automatically on exit.

Return type
bool

close()

Closes the device.

Return type
None

static close_all()

Closes all opened devices by the calling process.

Return type
None

property configuration: Optional[Union[int, str]]

Gets the selected configuration index.

Return type
Union[int, str, None]

property configurations: Tuple[*Configuration*, ...]

Returns a list of device configurations.

Return type
Tuple[*Configuration*, ...]

property digital_input: *DigitalInput*

Gets the Digital Input module (Logic Analyzer).

Return type
DigitalInput

property digital_io: *DigitalIo*

Gets the Digital IO module.

Return type
DigitalIo

property digital_output: *DigitalOutput*

Gets the Digital Output module (Pattern Generator).

Return type
DigitalOutput

static enumerate(*enum_filter=0*)

Enumerates all devices.

Return type
tuple

get_parameter(*parameter*)

Gets a device parameter.

Return type
int

get_trigger(*pin*)

Gets the configured trigger setting for a trigger I/O pin.

Return type
TriggerSource

property handle: object

Gets a handle to the device.

Return type
object

property id: *DeviceId*

Gets the device ID.

Return type
DeviceId

property is_open: bool

Returns true if the device has been opened.

Return type
bool

property name: str

Gets the device name.

Return type
str

open()

Opens the device.

Return type
None

property protocols: *Protocols*

Gets the Digital Protocols module.

Return type
Protocols

reset()

Resets and configures all device and instrument parameters to default values.

Return type
None

property revision: str

Gets the device revision.

Return type
str

property serial_number: str

Gets the 12-digit, unique device serial number.

Return type
str

set_parameter(*parameter*, *value*)

Sets a device parameter.

Return type
None

set_trigger(*pin*, *trigger_source*)

Sets the trigger I/O pin with a specific TriggerSource option.

Return type
None

property trigger_info: Tuple[*TriggerSource*, ...]

Gets the supported trigger source options for the global trigger bus.

Return type
Tuple[*TriggerSource*, ...]

trigger_pc()

Generates one pulse on the PC trigger line.

Return type
None

property trigger_slope_info: Tuple[*TriggerSlope*, ...]

Gets the supported trigger slopes.

Return type
Tuple[*TriggerSlope*, ...]

property user_name: str

Gets the user-defined device name.

Return type
str

16.11.4 dwfpy.device.DeviceBase

class DeviceBase(*configuration=None*, *serial_number=None*, *device_id=None*, *device_type=None*, *device_index=None*)

Bases: `object`

Base class for Digilent WaveForms devices.

Parameters

- **configuration** (`Union[int, str, None]`) – Select the active configuration.
- **serial_number** (`Optional[str]`) – Filter devices by serial number.
- **device_id** (`Optional[int]`) – Filter devices by device ID.
- **device_type** (`Optional[int]`) – Filter devices by device type.
- **device_index** (`Union[int, _EnumeratedIndex, None]`) – Filter devices by device index.

Methods

close	Closes the device.
get_parameter	Gets a device parameter.
get_trigger	Gets the configured trigger setting for a trigger I/O pin.
open	Opens the device.
reset	Resets and configures all device and instrument parameters to default values.
set_parameter	Sets a device parameter.
set_trigger	Sets the trigger I/O pin with a specific TriggerSource option.
trigger_pc	Generates one pulse on the PC trigger line.

Attributes

analog_input	Gets the Analog Input module (Oscilloscope).
analog_io	Gets the Analog IO module.
analog_output	Gets the Analog Output module (Arbitrary Waveform Generator).
application	Gets the WaveForms application.
auto_configure	Gets or sets a value indicating to automatically configure the device when parameters are changed.
auto_reset	Gets or sets a value indicating to reset the device automatically on exit.
configuration	Gets the selected configuration index.
configurations	Returns a list of device configurations.
digital_input	Gets the Digital Input module (Logic Analyzer).
digital_io	Gets the Digital IO module.
digital_output	Gets the Digital Output module (Pattern Generator).
handle	Gets a handle to the device.
id	Gets the device ID.
is_open	Returns true if the device has been opened.
name	Gets the device name.
protocols	Gets the Digital Protocols module.
revision	Gets the device revision.
serial_number	Gets the 12-digit, unique device serial number.
trigger_info	Gets the supported trigger source options for the global trigger bus.
trigger_slope_info	Gets the supported trigger slopes.
user_name	Gets the user-defined device name.

property analog_input: *AnalogInput*

Gets the Analog Input module (Oscilloscope).

Return type
AnalogInput

property analog_io: *AnalogIo*

Gets the Analog IO module.

Return type
AnalogIo

property analog_output: *AnalogOutput*

Gets the Analog Output module (Arbitrary Waveform Generator).

Return type
AnalogOutput

property application: *Application*

Gets the WaveForms application.

Return type
Application

property auto_configure: int

Gets or sets a value indicating to automatically configure the device when parameters are changed.

Return type
int

property auto_reset: bool

Gets or sets a value indicating to reset the device automatically on exit.

Return type
bool

close()

Closes the device.

Return type
None

property configuration: Optional[Union[int, str]]

Gets the selected configuration index.

Return type
Union[int, str, None]

property configurations: Tuple[*Configuration*, ...]

Returns a list of device configurations.

Return type
Tuple[*Configuration*, ...]

property digital_input: *DigitalInput*

Gets the Digital Input module (Logic Analyzer).

Return type
DigitalInput

property digital_io: *DigitalIo*

Gets the Digital IO module.

Return type
DigitalIo

property digital_output: *DigitalOutput*

Gets the Digital Output module (Pattern Generator).

Return type
DigitalOutput

get_parameter(*parameter***)**

Gets a device parameter.

Return type
int

get_trigger(*pin***)**

Gets the configured trigger setting for a trigger I/O pin.

Return type
TriggerSource

property handle: object

Gets a handle to the device.

Return type
object

property id: *DeviceId*

Gets the device ID.

Return type
DeviceId

property is_open: bool

Returns true if the device has been opened.

Return type
bool

property name: str

Gets the device name.

Return type
str

open()

Opens the device.

Return type
None

property protocols: *Protocols*

Gets the Digital Protocols module.

Return type
Protocols

reset()

Resets and configures all device and instrument parameters to default values.

Return type
None

property revision: str

Gets the device revision.

Return type
str

property serial_number: str

Gets the 12-digit, unique device serial number.

Return type
str

set_parameter(*parameter*, *value*)

Sets a device parameter.

Return type
None

set_trigger(*pin*, *trigger_source*)

Sets the trigger I/O pin with a specific TriggerSource option.

Return type
None

property trigger_info: Tuple[*TriggerSource*, ...]

Gets the supported trigger source options for the global trigger bus.

Return type
Tuple[*TriggerSource*, ...]

trigger_pc()

Generates one pulse on the PC trigger line.

Return type
None

property trigger_slope_info: Tuple[*TriggerSlope*, ...]

Gets the supported trigger slopes.

Return type
Tuple[*TriggerSlope*, ...]

property user_name: str

Gets the user-defined device name.

Return type
str

16.11.5 dwfpy.device.DigitalDiscovery

class DigitalDiscovery(*configuration=None*, *serial_number=None*, *device_type=None*, *device_index=None*)

Bases: *DeviceBase*

Digilent Digital Discovery device.

Parameters

- **configuration** – Select the active configuration.
- **serial_number** – Filter devices by serial number.
- **device_id** – Filter devices by device ID.
- **device_type** – Filter devices by device type.
- **device_index** – Filter devices by device index.

Methods

close	Closes the device.
get_parameter	Gets a device parameter.
get_trigger	Gets the configured trigger setting for a trigger I/O pin.
open	Opens the device.
reset	Resets and configures all device and instrument parameters to default values.
set_parameter	Sets a device parameter.
set_trigger	Sets the trigger I/O pin with a specific TriggerSource option.
trigger_pc	Generates one pulse on the PC trigger line.

Attributes

analog_input	Gets the Analog Input module (Oscilloscope).
analog_io	Gets the Analog IO module.
analog_output	Gets the Analog Output module (Arbitrary Waveform Generator).
application	Gets the WaveForms application.
auto_configure	Gets or sets a value indicating to automatically configure the device when parameters are changed.
auto_reset	Gets or sets a value indicating to reset the device automatically on exit.
configuration	Gets the selected configuration index.
configurations	Returns a list of device configurations.
digital_frequency	Gets or sets the frequency for DIO lines.
digital_input	Gets the Digital Input module (Logic Analyzer).
digital_io	Gets the Digital IO module.
digital_output	Gets the Digital Output module (Pattern Generator).
din_pull_up_down	Gets or sets the weak pull for DIN lines.
dio_drive_strength	Gets or sets the drive strength for DIO lines.
dio_pull_enable	Gets or sets the pull enable for DIO 39-24 as bit-field set.
dio_pull_up_down	Gets or sets the pull up/down for DIO 39-24 as bit-field set.
dio_slew	Gets or sets the slew rate for DIO lines.
handle	Gets a handle to the device.
id	Gets the device ID.
is_open	Returns true if the device has been opened.
name	Gets the device name.
protocols	Gets the Digital Protocols module.
revision	Gets the device revision.
serial_number	Gets the 12-digit, unique device serial number.
supplies	Gets the power supplies.
trigger_info	Gets the supported trigger source options for the global trigger bus.
trigger_slope_info	Gets the supported trigger slopes.
usb_current	Gets the USB line current.
usb_voltage	Gets the USB line voltage.
user_name	Gets the user-defined device name.
vio_current	Gets the VIO current reading.
vio_voltage	Gets the VIO voltage reading.

class Supplies(*device*)

Bases: `object`

The power supplies.

class Digital(*device*)

Bases: `object`

The digital power supply.

setup(*voltage*)

Sets up the power supply.

voltage
[float] The output voltage.

Return type
None

property voltage: float

Gets or sets the voltage of the digital power supply.

Return type
float

property digital: *Digital*

Gets the digital power supply.

Return type
Digital

property master_enable: bool

Gets or sets the master enable switch.

Return type
bool

property master_enable_status: bool

Gets the master enable status.

Return type
bool

property analog_input: *AnalogInput*

Gets the Analog Input module (Oscilloscope).

Return type
AnalogInput

property analog_io: *AnalogIo*

Gets the Analog IO module.

Return type
AnalogIo

property analog_output: *AnalogOutput*

Gets the Analog Output module (Arbitrary Waveform Generator).

Return type
AnalogOutput

property application: *Application*

Gets the WaveForms application.

Return type
Application

property auto_configure: int

Gets or sets a value indicating to automatically configure the device when parameters are changed.

Return type
int

property auto_reset: bool

Gets or sets a value indicating to reset the device automatically on exit.

Return type
bool

close()

Closes the device.

Return type
None

property configuration: Optional[Union[int, str]]

Gets the selected configuration index.

Return type
Union[int, str, None]

property configurations: Tuple[*Configuration*, ...]

Returns a list of device configurations.

Return type
Tuple[*Configuration*, ...]

property digital_frequency: float

Gets or sets the frequency for DIO lines.

Return type
float

property digital_input: *DigitalInput*

Gets the Digital Input module (Logic Analyzer).

Return type
DigitalInput

property digital_io: *DigitalIo*

Gets the Digital IO module.

Return type
DigitalIo

property digital_output: *DigitalOutput*

Gets the Digital Output module (Pattern Generator).

Return type
DigitalOutput

property din_pull_up_down: float

Gets or sets the weak pull for DIN lines. Can be 0.0=low, 0.5=middle, 1.0=high

Return type
float

property dio_drive_strength: float

Gets or sets the drive strength for DIO lines. Can be 0 (auto based on digital voltage), 2, 4, 6, 8, 12, or 16mA

Return type
float

property dio_pull_enable: int

Gets or sets the pull enable for DIO 39-24 as bit-field set.

Return type
int

property dio_pull_up_down: int
Gets or sets the pull up/down for DIO 39-24 as bit-field set.

Return type
int

property dio_slew: int
Gets or sets the slew rate for DIO lines. 0=QuietIO, 1=Slow, 2=Fast

Return type
int

get_parameter(*parameter*)
Gets a device parameter.

Return type
int

get_trigger(*pin*)
Gets the configured trigger setting for a trigger I/O pin.

Return type
TriggerSource

property handle: object
Gets a handle to the device.

Return type
object

property id: *DeviceId*
Gets the device ID.

Return type
DeviceId

property is_open: bool
Returns true if the device has been opened.

Return type
bool

property name: str
Gets the device name.

Return type
str

open()
Opens the device.

Return type
None

property protocols: *Protocols*
Gets the Digital Protocols module.

Return type
Protocols

reset()

Resets and configures all device and instrument parameters to default values.

Return type
None

property revision: str

Gets the device revision.

Return type
str

property serial_number: str

Gets the 12-digit, unique device serial number.

Return type
str

set_parameter(*parameter*, *value*)

Sets a device parameter.

Return type
None

set_trigger(*pin*, *trigger_source*)

Sets the trigger I/O pin with a specific TriggerSource option.

Return type
None

property supplies: *Supplies*

Gets the power supplies.

Return type
Supplies

property trigger_info: Tuple[*TriggerSource*, ...]

Gets the supported trigger source options for the global trigger bus.

Return type
Tuple[*TriggerSource*, ...]

trigger_pc()

Generates one pulse on the PC trigger line.

Return type
None

property trigger_slope_info: Tuple[*TriggerSlope*, ...]

Gets the supported trigger slopes.

Return type
Tuple[*TriggerSlope*, ...]

property usb_current: float

Gets the USB line current.

Return type
float

property usb_voltage: float

Gets the USB line voltage.

Return type
float

property user_name: str

Gets the user-defined device name.

Return type
str

property vio_current: float

Gets the VIO current reading.

Return type
float

property vio_voltage: float

Gets the VIO voltage reading.

Return type
float

16.11.6 dwfpy.device.ElectronicsExplorer

class ElectronicsExplorer(*configuration=None*, *serial_number=None*, *device_type=None*, *device_index=None*)

Bases: *DeviceBase*

Digilent Electronics Explorer device.

Parameters

- **configuration** – Select the active configuration.
- **serial_number** – Filter devices by serial number.
- **device_id** – Filter devices by device ID.
- **device_type** – Filter devices by device type.
- **device_index** – Filter devices by device index.

Methods

close	Closes the device.
get_parameter	Gets a device parameter.
get_trigger	Gets the configured trigger setting for a trigger I/O pin.
open	Opens the device.
reset	Resets and configures all device and instrument parameters to default values.
set_parameter	Sets a device parameter.
set_trigger	Sets the trigger I/O pin with a specific TriggerSource option.
trigger_pc	Generates one pulse on the PC trigger line.

Attributes

analog_input	Gets the Analog Input module (Oscilloscope).
analog_io	Gets the Analog IO module.
analog_output	Gets the Analog Output module (Arbitrary Waveform Generator).
application	Gets the WaveForms application.
auto_configure	Gets or sets a value indicating to automatically configure the device when parameters are changed.
auto_reset	Gets or sets a value indicating to reset the device automatically on exit.
configuration	Gets the selected configuration index.
configurations	Returns a list of device configurations.
digital_input	Gets the Digital Input module (Logic Analyzer).
digital_io	Gets the Digital IO module.
digital_output	Gets the Digital Output module (Pattern Generator).
handle	Gets a handle to the device.
id	Gets the device ID.
is_open	Returns true if the device has been opened.
name	Gets the device name.
protocols	Gets the Digital Protocols module.
revision	Gets the device revision.
serial_number	Gets the 12-digit, unique device serial number.
supplies	Gets the power supplies.
trigger_info	Gets the supported trigger source options for the global trigger bus.
trigger_slope_info	Gets the supported trigger slopes.
user_name	Gets the user-defined device name.
voltmeters	Gets the voltmeters.

class Supplies(*device*)

Bases: `object`

The power supplies.

class Fixed(*device*)

Bases: `object`

The fixed power supply.

property current: float

Gets the current of the fixed power supply.

Return type
`float`

property enabled: bool

Enables or disables the fixed power supply.

Return type
`bool`

setup(*voltage*, *enabled=True*)

Sets up the power supply.

voltage
[float] The output voltage.

enabled
[bool, optional] If True, then the power supply is enabled (default True).

Return type
None

property voltage: float

Gets or sets the voltage of the fixed power supply.

Return type
float

class Negative(*device*)

Bases: object

The negative power supply.

property current: float

Gets the current of the negative power supply.

Return type
float

property current_limit: float

Gets or sets the current limit of the negative power supply.

Return type
float

property enabled: bool

Enables the negative power supply.

Return type
bool

setup(*voltage*, *current_limit=None*, *enabled=True*)

Sets up the negative power supply.

voltage
[float] The output voltage (must be a negative value).

current_limit
[float, optional] The power supply current limit in A.

enabled
[bool, optional] If True, then the power supply is enabled (default True).

Return type
None

property voltage: float

Gets or sets the voltage of the negative power supply.

Return type
float

class Positive(*device*)

Bases: object

The positive power supply.

property current: float

Gets the current of the positive power supply.

Return type
float

property current_limit: float

Gets or sets the current limit of the positive power supply.

Return type
float

property enabled: bool

Enables or disables the positive power supply.

Return type
bool

setup(*voltage*, *current_limit=None*, *enabled=True*)

Sets up the positive power supply.

voltage
[float] The output voltage.

current_limit
[float, optional] The power supply current limit in A.

enabled
[bool, optional] If True, then the power supply is enabled (default True).

Return type
None

property voltage: float

Gets or sets the voltage of the positive power supply.

Return type
float

class Reference1(*device*)

Bases: object

The voltage reference 1.

property enabled: bool

Enables the voltage reference 1.

Return type
bool

setup(*voltage*, *enabled=True*)

Sets up the voltage reference.

voltage
[float] The output voltage.

enabled
[bool, optional] If True, then the voltage reference is enabled (default True).

Return type
None

property voltage: float

Gets or sets the voltage of the voltage reference 1.

Return type
float

class Reference2(*device*)

Bases: object

The voltage reference 2.

property enabled: bool

Enables the voltage reference 2.

Return type
bool

setup(*voltage*, *enabled=True*)

Sets up the voltage reference.

voltage
[float] The output voltage.

enabled
[bool, optional] If True, then the voltage reference is enabled (default True).

Return type
None

property voltage: float

Gets or sets the voltage of the voltage reference 2.

Return type
float

property fixed: *Fixed*

Gets the fixed power supply.

Return type
Fixed

property master_enable: bool

Gets or sets the master enable switch.

Return type
bool

property master_enable_status: bool

Gets the master enable status.

Return type
bool

property negative: *Negative*

Gets the negative power supply.

Return type
Negative

property positive: *Positive*

Gets the positive power supply.

Return type
Positive

property reference1: *Reference1*

Gets the voltage reference 1.

Return type
Reference1

property reference2: *Reference2*

Gets the voltage reference 2.

Return type
Reference2

class Voltmeters(*device*)

Bases: object

The devices voltmeters.

property voltmeter1: float

Reads the voltage of voltmeter 1.

Return type
float

property voltmeter2: float

Reads the voltage of voltmeter 2.

Return type
float

property voltmeter3: float

Reads the voltage of voltmeter 3.

Return type
float

property voltmeter4: float

Reads the voltage of voltmeter 4.

Return type
float

property analog_input: *AnalogInput*

Gets the Analog Input module (Oscilloscope).

Return type
AnalogInput

property analog_io: *AnalogIo*

Gets the Analog IO module.

Return type
AnalogIo

property analog_output: *AnalogOutput*

Gets the Analog Output module (Arbitrary Waveform Generator).

Return type
AnalogOutput

property application: *Application*

Gets the WaveForms application.

Return type
Application

property auto_configure: int

Gets or sets a value indicating to automatically configure the device when parameters are changed.

Return type
int

property auto_reset: bool

Gets or sets a value indicating to reset the device automatically on exit.

Return type
bool

close()

Closes the device.

Return type
None

property configuration: Optional[Union[int, str]]

Gets the selected configuration index.

Return type
Union[int, str, None]

property configurations: Tuple[*Configuration*, ...]

Returns a list of device configurations.

Return type
Tuple[*Configuration*, ...]

property digital_input: *DigitalInput*

Gets the Digital Input module (Logic Analyzer).

Return type
DigitalInput

property digital_io: *DigitalIo*

Gets the Digital IO module.

Return type
DigitalIo

property digital_output: *DigitalOutput*

Gets the Digital Output module (Pattern Generator).

Return type
DigitalOutput

get_parameter(*parameter*)

Gets a device parameter.

Return type
int

get_trigger(*pin*)

Gets the configured trigger setting for a trigger I/O pin.

Return type
TriggerSource

property handle: object

Gets a handle to the device.

Return type
object

property id: *DeviceId*

Gets the device ID.

Return type
DeviceId

property is_open: bool

Returns true if the device has been opened.

Return type
bool

property name: str

Gets the device name.

Return type
str

open()

Opens the device.

Return type
None

property protocols: *Protocols*

Gets the Digital Protocols module.

Return type
Protocols

reset()

Resets and configures all device and instrument parameters to default values.

Return type
None

property revision: str

Gets the device revision.

Return type
str

property serial_number: str

Gets the 12-digit, unique device serial number.

Return type
str

set_parameter(*parameter*, *value*)

Sets a device parameter.

Return type
None

set_trigger(*pin*, *trigger_source*)

Sets the trigger I/O pin with a specific TriggerSource option.

Return type
None

property supplies: *Supplies*

Gets the power supplies.

Return type
Supplies

property trigger_info: Tuple[*TriggerSource*, ...]

Gets the supported trigger source options for the global trigger bus.

Return type
Tuple[*TriggerSource*, ...]

trigger_pc()

Generates one pulse on the PC trigger line.

Return type
None

property trigger_slope_info: Tuple[*TriggerSlope*, ...]

Gets the supported trigger slopes.

Return type
Tuple[*TriggerSlope*, ...]

property user_name: str

Gets the user-defined device name.

Return type
str

property voltmeters: *Voltmeters*

Gets the voltmeters.

Return type
Voltmeters

16.12 dwfpy.device_info

Device information for Digilent WaveForms devices.

Classes

DeviceInfo([device_index])	Device information gathered during device enumeration.

16.12.1 dwfpy.device_info.DeviceInfo

class DeviceInfo(*device_index=None*)

Bases: object

Device information gathered during device enumeration.

Methods

get_device_name	Gets the device name.
get_properties	Read all device properties.
get_serial_number	Gets the device serial number.
get_user_name	Gets the user-defined device name.
normalize_serial_number	Normalizes the serial number, i.e. strip the leading 'SN:' and convert to uppercase.

Attributes

configurations	Returns a list of device configurations.
has_properties	Returns True, if the device properties have been read.
id	Gets the device ID.
is_open	Returns true if the device has been opened.
name	Gets the device name.
revision	Gets the device revision.
serial_number	Gets the 12-digit, unique device serial number.
user_name	Gets the user-defined device name.

property configurations: Tuple[*Configuration*, ...]

Returns a list of device configurations.

Return type
Tuple[*Configuration*, ...]

static get_device_name(*device_index*)

Gets the device name.

Return type
str

get_properties(*device_index*)

Read all device properties.

Return type
None

static get_serial_number(*device_index*)

Gets the device serial number.

Return type
str

static get_user_name(*device_index*)

Gets the user-defined device name.

Return type
str

property has_properties: bool

Returns True, if the device properties have been read.

Return type
bool

property id: int

Gets the device ID.

Return type
int

property is_open: bool

Returns true if the device has been opened.

Return type
bool

property name: str

Gets the device name.

Return type
str

static normalize_serial_number(*serial_number*)

Normalizes the serial number, i.e. strip the leading 'SN:' and convert to uppercase.

Return type
str

property revision: int

Gets the device revision.

Return type
int

property serial_number: str

Gets the 12-digit, unique device serial number.

Return type
str

property user_name: str

Gets the user-defined device name.

Return type
str

16.13 dwfpy.digital_input

Digital Input module for Digilent WaveForms devices.

Classes

DigitalInput(device)	Digital Input module (Logic Analyzer).
DigitalInputChannel(module, channel)	Represents a Digital Input channel.
DigitalInputChannelTrigger(channel)	Represents the trigger unit of a Digital Input channel.
DigitalInputClock(module)	Represents the clock unit of a Digital Input device.
DigitalInputTrigger(module)	Represents the trigger unit of a Digital Input device.

16.13.1 dwfpy.digital_input.DigitalInput

class DigitalInput(*device*)

Bases: object

Digital Input module (Logic Analyzer).

Methods

configure	Configures the instrument and starts the acquisition..
get_data	Gets the acquired data samples.
get_noise	Gets the acquired noise samples.
read_status	Gets the acquisition state and optionally reads the data.
record	Starts a data recording.
reset	Resets and configures all instrument parameters to default values.
setup_acquisition	Sets up a new data acquisition.
setup_counter_trigger	Sets up a counter trigger.
setup_edge_trigger	Sets up an edge trigger.
setup_glitch_trigger	Sets up a glitch trigger.
setup_length_trigger	Sets up a length trigger.
setup_level_trigger	Sets up a level trigger.
setup_more_trigger	Sets up a more trigger.
setup_timeout_trigger	Sets up a timeout trigger.
setup_trigger	Sets up the trigger condition.
single	Starts a single data acquisition.
stream	Streams data to a callback function.
wait_for_status	Waits for the specified acquisition state.

Attributes

acquisition_mode	Gets or sets the acquisition mode.
acquisition_mode_info	Gets the supported acquisition modes.
auto_triggered	Returns True if the acquisition is auto triggered.
buffer_size	Gets or sets the buffer size.
buffer_size_max	Gets the maximum supported buffer size.
channels	Gets a collection of Digital Input channels.
clock	Gets the clock unit.
device	Gets the device.
dio_first	Gets or sets a value indicating if DIO pins will be placed in front of the DIN pins.
record_status	Gets information about the recording process.
remaining_samples	Gets the number of samples left in the acquisition.
sample_format	Gets or sets the number of bits to be sampled.
sample_mode	Gets or sets the sample mode.
sample_mode_info	Gets the supported sample modes.
sample_rate	Gets or sets the sample rate.
sample_sensible	Gets or sets the signals to be used for data compression in record acquisition mode.
time	Returns instrument trigger time information.
trigger	Gets the trigger unit.
valid_samples	Gets the number of valid/acquired data samples.
write_index	Gets the buffer write pointer which is needed in scan_screen acquisition mode to display the scan bar.

property acquisition_mode: *AcquisitionMode*

Gets or sets the acquisition mode.

Return type
AcquisitionMode

property acquisition_mode_info: Tuple[*AcquisitionMode*, ...]

Gets the supported acquisition modes.

Return type
Tuple[*AcquisitionMode*, ...]

property auto_triggered: bool

Returns True if the acquisition is auto triggered.

Return type
bool

property buffer_size: int

Gets or sets the buffer size.

Return type
int

property buffer_size_max: int

Gets the maximum supported buffer size.

Return type
int

property channels

Gets a collection of Digital Input channels.

property clock: *DigitalInputClock*

Gets the clock unit.

Return type
DigitalInputClock

configure(*reconfigure=False*, *start=False*)

Configures the instrument and starts the acquisition..

Return type
None

property device: *Device*

Gets the device.

Return type
Device

property dio_first: bool

Gets or sets a value indicating if DIO pins will be placed in front of the DIN pins.

Return type
bool

get_data(*first_sample=0*, *sample_count=- 1*)

Gets the acquired data samples.

Before calling this function, call the 'read_status()' function to read the data from the device.

get_noise(*first_sample=0*, *sample_count=- 1*)

Gets the acquired noise samples.

Before calling this function, call the 'read_status()' function to read the data from the device.

read_status(*read_data=False*)

Gets the acquisition state and optionally reads the data.

Return type
Status

record(*sample_rate=None*, *sample_format=None*, *sample_sensible=None*, *sample_count=None*, *prefill=None*, *callback=None*, *configure=False*, *start=False*)

Starts a data recording.

sample_rate
[float, optional] The sampling frequency in Hz.

sample_format
[int, optional] The number of bits to be sampled. Can be 8, 16, or 32.

sample_sensible
[int, optional] The signals to be used for data compression, as a bit-mask.

sample_count
[int or float, optional] The number of samples to be acquired after the trigger.

prefill
[int or float, optional] The number of samples to be acquired before the trigger.

callback
[function, optional] A custom function to monitor the recording process.

configure
[bool, optional] If True, then the instrument is configured (default False).

start
[bool, optional] If True, then the recording is started (default False).

DigitalRecorder
The recorder instance.

Return type
DigitalRecorder

property record_status: Tuple[int, int, int]
Gets information about the recording process. Returns (available_samples, lost_samples, corrupted_samples)

Return type
Tuple[int, int, int]

property remaining_samples: int
Gets the number of samples left in the acquisition.

Return type
int

reset()
Resets and configures all instrument parameters to default values.

Return type
None

property sample_format: int
Gets or sets the number of bits to be sampled. Can be 8, 16, or 32.

Return type
int

property sample_mode: *DigitalInputSampleMode*
Gets or sets the sample mode.

Return type
DigitalInputSampleMode

property sample_mode_info: Tuple[*DigitalInputSampleMode*, ...]
Gets the supported sample modes.

Return type
Tuple[*DigitalInputSampleMode*, ...]

property sample_rate: float
Gets or sets the sample rate.

Return type
float

property sample_sensible: int

Gets or sets the signals to be used for data compression in record acquisition mode.

Return type
int

setup_acquisition(*mode=None*, *sample_rate=None*, *sample_format=None*, *buffer_size=None*, *position=None*, *prefill=None*, *configure=False*, *start=False*)

Sets up a new data acquisition.

mode
[str, optional] The sampling mode. Can be 'single', 'scan-shift', 'scan-screen', or 'record'.

sample_rate
[float, optional] The sampling frequency in Hz.

sample_format
[int, optional] The number of bits to be sampled. Can be 8, 16, or 32.

buffer_size
[int or float, optional] The buffer size.

position
[int or float, optional] The number of samples to be acquired after the trigger.

prefill
[int or float, optional] The number of samples to be acquired before the trigger (record mode only).

configure
[bool, optional] If True, then the instrument is configured (default False).

start
[bool, optional] If True, then the acquisition is started (default False).

Return type
None

setup_counter_trigger(*channel*, *condition*, *reset_channel*, *reset_condition*, *max_counter*)

Sets up a counter trigger.

channel
[int] The trigger channel.

condition
[str] The trigger condition. Can be 'ignore', 'low', 'high', 'rise', 'fall', or 'edge'.

reset_channel
[int] The counter reset channel.

reset_condition
[str] The reset condition. Can be 'ignore', 'low', 'high', 'rise', 'fall', or 'edge'.

max_counter
[int] The maximum counter value.

Return type
None

`setup_edge_trigger`(*channel*, *edge*)

Sets up an edge trigger.

channel
[int] The trigger channel.

edge
[str] The trigger edge. Can be 'rising' or 'falling'.

Return type
None

`setup_glitch_trigger`(*channel*, *polarity*, *less_than*)

Sets up a glitch trigger.

channel
[int] The trigger channel.

polarity
[str] The trigger polarity. Can be 'positive' or 'negative'.

less_than
[float] The maximum pulse width in seconds.

Return type
None

`setup_length_trigger`(*channel*, *polarity*, *length*, *hysteresis=0.0*)

Sets up a length trigger.

channel
[int] The trigger channel.

polarity
[str] The trigger polarity. Can be 'positive' or 'negative'.

length
[float] The minimum pulse width in seconds.

hysteresis
[float, optional] The pulse width hysteresis in seconds.

Return type
None

`setup_level_trigger`(*channel*, *level*)

Sets up a level trigger.

channel
[int] The trigger channel.

level
[str] The trigger level. Can be 'low' or 'high'.

Return type
None

`setup_more_trigger`(*channel*, *polarity*, *more_than*)

Sets up a more trigger.

channel
[int] The trigger channel.

polarity
[str] The trigger polarity. Can be 'positive' or 'negative'.

more_than
[float] The minimum pulse width in seconds.

Return type
None

`setup_timeout_trigger`(*channel*, *polarity*, *more_than*)

Sets up a timeout trigger.

channel
[int] The trigger channel.

polarity
[str] The trigger polarity. Can be 'positive' or 'negative'.

more_than
[float] The minimum pulse width in seconds.

Return type
None

`setup_trigger`(*source=None*, *slope=None*, *position=None*, *prefill=None*, *auto_timeout=None*)

Sets up the trigger condition.

source
[str, optional] The trigger source. Can be 'none', 'pc', 'detector-analog-in', 'detector-digital-in', 'analog-in', 'digital-in', 'digital-out', 'analog-out1', 'analog-out2', 'analog-out3', 'analog-out3', 'external1', 'external2', 'external3', 'external4', 'low', 'high', or 'clock'.

slope
[str, optional] The trigger slope. Can be 'rising', 'falling', or 'either'.

position
[int, optional] The number of samples to be acquired after the trigger.

prefill
[int, optional] The number of samples to be acquired before the trigger.

auto_timeout
[float, optional] The auto trigger timeout in seconds.

Return type
None

`single`(*sample_rate=None*, *sample_format=None*, *buffer_size=None*, *position=None*, *continuous=True*, *configure=False*, *start=False*)

Starts a single data acquisition.

sample_rate
[float, optional] The sampling frequency in Hz.

sample_format
[int, optional] The number of bits to be sampled. Can be 8, 16, or 32.

buffer_size
[int or float, optional] The buffer size.

position
[int or float, optional] The number of samples to be acquired after the trigger.

continuous
[bool, optional] If True, then the instrument is rearmed after the data is retrieved. (default True).

configure
[bool, optional] If True, then the instrument is configured (default False).

start
[bool, optional] If True, then the acquisition is started (default False).

tuple
If `start` is True, a tuple of integers containing the data samples. None otherwise.

Return type
`Optional[Tuple]`

stream(*callback*, *sample_rate=None*, *sample_format=None*, *sample_sensible=None*, *configure=False*, *start=False*)

Streams data to a callback function.

callback
[function] A user-defined function to receive the streaming data.

sample_rate
[float, optional] The sampling frequency in Hz.

sample_format
[int, optional] The number of bits to be sampled. Can be 8, 16, or 32.

sample_sensible
[int, optional] The signals to be used for data compression.

configure
[bool, optional] If True, then the instrument is configured (default False).

start
[bool, optional] If True, then the streaming is started (default False).

DigitalRecorder
The recorder instance.

Return type
DigitalRecorder

property time: None

Returns instrument trigger time information.

Return type
`None`

property trigger: *DigitalInputTrigger*

Gets the trigger unit.

Return type
DigitalInputTrigger

property valid_samples: **int**

Gets the number of valid/acquired data samples.

Return type
int

wait_for_status(*status*, *read_data=False*)

Waits for the specified acquisition state.

Return type
None

property write_index: **int**

Gets the buffer write pointer which is needed in scan_screen acquisition mode to display the scan bar.

Return type
int

16.13.2 dwfpy.digital_input.DigitalInputChannel

class DigitalInputChannel(*module*, *channel*)

Bases: object

Represents a Digital Input channel.

Methods

setup_reset_trigger	Sets up the trigger reset condition for this channel.
setup_trigger	Sets up the trigger condition for this channel.

Attributes

device	Gets the device.
index	Gets the channel index.
label	Gets or sets the channel label.
module	Gets the Digital Input module.
trigger	Gets the trigger unit.

property device: *Device*

Gets the device.

Return type
Device

property index: int

Gets the channel index.

Return type
int

property label: str

Gets or sets the channel label.

Return type
str

property module: *DigitalInput*

Gets the Digital Input module.

Return type
DigitalInput

setup_reset_trigger(*condition*)

Sets up the trigger reset condition for this channel.

condition
[str] The trigger condition. Can be 'ignore', 'low', 'high', 'rise', 'fall', or 'edge'.

Return type
None

setup_trigger(*condition*)

Sets up the trigger condition for this channel.

condition
[str] The trigger condition. Can be 'ignore', 'low', 'high', 'rise', 'fall', or 'edge'.

Return type
None

property trigger: *DigitalInputChannelTrigger*

Gets the trigger unit.

Return type
DigitalInputChannelTrigger

16.13.3 dwfpy.digital_input.DigitalInputChannelTrigger

class DigitalInputChannelTrigger(*channel*)

Bases: object

Represents the trigger unit of a Digital Input channel.

Methods

Attributes

falling_edge	Gets or sets the edge-fall trigger for this channel.
high_level	Gets or sets the high-level trigger for this channel.
low_level	Gets or sets the low-level trigger for this channel.
rising_edge	Gets or sets the edge-rise trigger for this channel.
supports_falling_edge	Returns True if the trigger detector supports a edge-fall trigger for this channel.
supports_high_level	Returns True if the trigger detector supports a high-level trigger for this channel.
supports_low_level	Returns True if the trigger detector supports a low-level trigger for this channel.
supports_rising_edge	Returns True if the trigger detector supports a edge-rise trigger for this channel.

property falling_edge: bool

Gets or sets the edge-fall trigger for this channel.

Return type
bool

property high_level: bool

Gets or sets the high-level trigger for this channel.

Return type
bool

property low_level: bool

Gets or sets the low-level trigger for this channel.

Return type
bool

property rising_edge: bool

Gets or sets the edge-rise trigger for this channel.

Return type
bool

property supports_falling_edge: bool

Returns True if the trigger detector supports a edge-fall trigger for this channel.

Return type
bool

property supports_high_level: bool

Returns True if the trigger detector supports a high-level trigger for this channel.

Return type
bool

property supports_low_level: bool

Returns True if the trigger detector supports a low-level trigger for this channel.

Return type
bool

property supports_rising_edge: bool

Returns True if the trigger detector supports a edge-rise trigger for this channel.

Return type
bool

16.13.4 dwfpy.digital_input.DigitalInputClock

class DigitalInputClock(*module*)

Bases: object

Represents the clock unit of a Digital Input device.

Methods

Attributes

divider	Gets or sets the configured clock divider value.
divider_max	Gets the maximum supported clock divider value.
frequency	Gets the internal clock frequency in Hz.
source	Gets or sets the clock source.
source_info	Gets the supported clock sources.

property divider: int

Gets or sets the configured clock divider value.

Return type
int

property divider_max: int

Gets the maximum supported clock divider value.

Return type
int

property frequency: float

Gets the internal clock frequency in Hz.

Return type
float

property source: *DigitalInputClockSource*

Gets or sets the clock source.

Return type
DigitalInputClockSource

property source_info: Tuple[*DigitalInputClockSource*, ...]

Gets the supported clock sources.

Return type
Tuple[*DigitalInputClockSource*, ...]

16.13.5 dwfpy.digital_input.DigitalInputTrigger

class DigitalInputTrigger(*module*)

Bases: object

Represents the trigger unit of a Digital Input device.

Methods

get_trigger_mask	Gets state and edge trigger condition as a bit-mask.
get_trigger_mask_info	Gets the supported triggers as a bit-mask.
set_counter	Configures the trigger counter.
set_length	Configures the trigger timing.
set_match	Configures the trigger deserializer.
set_reset_mask	Configures the trigger reset condition as a bit-mask.
set_trigger_mask	Sets state and edge trigger conditions as a bit-mask.

Attributes

auto_timeout	Gets or sets the auto trigger timeout in seconds.
auto_timeout_max	Gets the maximum auto trigger timeout in seconds.
auto_timeout_min	Gets the minimum auto trigger timeout in seconds.
auto_timeout_steps	Gets the number of adjustable steps of the timeout.
position	Gets or sets the number of samples to acquire after trigger.
position_max	Gets the maximum values of the trigger position in samples.
prefill	Gets or sets the number of samples to acquire before arming in 'record' acquisition mode.
slope	Gets or sets the trigger slope for the instrument.
source	Gets or sets the current trigger source setting for the instrument.

property auto_timeout: float

Gets or sets the auto trigger timeout in seconds.

Return type
float

property auto_timeout_max: float

Gets the maximum auto trigger timeout in seconds.

Return type
float

property auto_timeout_min: float

Gets the minimum auto trigger timeout in seconds.

Return type
float

property auto_timeout_steps: int

Gets the number of adjustable steps of the timeout.

Return type
int

get_trigger_mask()

Gets state and edge trigger condition as a bit-mask. Returs (low_level, high_level, rising_edge, falling_edge)

Return type
Tuple[int, int, int, int]

get_trigger_mask_info()

Gets the supported triggers as a bit-mask. Returs (low_level, high_level, rising_edge, falling_edge)

Return type
int

property position: int

Gets or sets the number of samples to acquire after trigger.

Return type
int

property position_max: int

Gets the maximum values of the trigger position in samples.

Return type
int

property prefill: int

Gets or sets the number of samples to acquire before arming in 'record' acquisition mode.

Return type
int

set_counter(*count*, *restart=False*)

Configures the trigger counter.

Return type
None

set_length(*min_length*, *max_length*, *sync_mode*)

Configures the trigger timing.

Return type
None

set_match(*pin*, *mask*, *value*, *bit_stuffing*)

Configures the trigger deserializer.

Return type
None

set_reset_mask(*low_level=0*, *high_level=0*, *rising_edge=0*, *falling_edge=0*)

Configures the trigger reset condition as a bit-mask.

Return type
None

set_trigger_mask(*low_level=0*, *high_level=0*, *rising_edge=0*, *falling_edge=0*)

Sets state and edge trigger conditions as a bit-mask.

Return type
None

property slope: *TriggerSlope*

Gets or sets the trigger slope for the instrument.

Return type
TriggerSlope

property source: *TriggerSource*

Gets or sets the current trigger source setting for the instrument.

Return type
TriggerSource

16.14 dwfpy.digital_io

Digital IO module for Digilent WaveForms devices.

Classes

DigitalIo(device)	Digital IO module.
DigitalIoChannel(module, channel)	Represents an Digital IO channel.

16.14.1 dwfpy.digital_io.DigitalIo

class DigitalIo(*device*)

Bases: object

Digital IO module.

Methods

configure	Configures the instrument.
read_status	Reads the status and input values of the device.
reset	Resets and configures all instrument parameters to default values.

Attributes

channels	Gets a collection of Digital IO channels.
device	Gets the device.
input_state	Gets the input state of all pins as a bit-mask.
input_state_mask	Gets the pins that can be read from as a bit-mask.
output_enable	Enables or disables the specified pins as an output via a bit-mask.
output_enable_mask	Gets the pins that can be enabled as an output as a bit-mask.
output_state	Gets or sets the ouput state of all pins via a bit-mask.
output_state_mask	Gets the pins that can be written to as a bit-mask.

property channels: Tuple[*DigitalIoChannel*, ...]

Gets a collection of Digital IO channels.

Return type
Tuple[*DigitalIoChannel*, ...]

configure()

Configures the instrument.

Return type
None

property device: *Device*

Gets the device.

Return type
Device

property input_state: int

Gets the input state of all pins as a bit-mask. Before calling this function, call the 'read_status()' function to read the Digital I/O states from the device.

Return type
int

property input_state_mask: int

Gets the pins that can be read from as a bit-mask.

Return type
int

property output_enable: int

Enables or disables the specified pins as an output via a bit-mask.

Return type
int

property output_enable_mask: int

Gets the pins that can be enabled as an output as a bit-mask.

Return type
int

property output_state: int

Gets or sets the ouput state of all pins via a bit-mask.

Return type
int

property output_state_mask: int

Gets the pins that can be written to as a bit-mask.

Return type
int

read_status()

Reads the status and input values of the device.

Return type
None

reset()

Resets and configures all instrument parameters to default values.

Return type
None

16.14.2 dwfpy.digital_io.DigitalIoChannel

class DigitalIoChannel(*module*, *channel*)

Bases: object

Represents an Digital IO channel.

Methods

setup	Sets up the channel.

Attributes

can_read	Returns True of the pin can be read.
can_write	Returns True of the pin can be written.
device	Gets the device.
enabled	Enables or disables the pin as an output.
index	Gets the channel index.
input_state	Gets the input state of the pin.
label	Gets or sets the channel label.
module	Gets the Digital IO module.
output_state	Gets or sets the ouput state of the pin.

property can_read: bool

Returns True of the pin can be read.

Return type
bool

property can_write: bool

Returns True of the pin can be written.

Return type
bool

property device: *Device*

Gets the device.

Return type
Device

property enabled: bool

Enables or disables the pin as an output.

Return type
bool

property index: int

Gets the channel index.

Return type
int

property input_state: bool

Gets the input state of the pin. Before calling this function, call the 'read_status()' function to read the Digital I/O states from the device.

Return type
bool

property label: str

Gets or sets the channel label.

Return type
str

property module: *DigitalIo*

Gets the Digital IO module.

Return type
DigitalIo

property output_state: bool

Gets or sets the ouput state of the pin.

Return type
bool

setup(*enabled=None*, *state=None*, *configure=None*)

Sets up the channel.

enabled
[bool, optional] If True, then the channel is configured as an output.

state
[bool, optional] The output state.

configure
[bool, optional] If True, then the instrument is configured.

Return type
None

16.15 dwfpy.digital_output

Digital Output module for Digilent WaveForms devices.

Classes

DigitalOutput(device)	Digital Output module (Pattern Generator).
DigitalOutputChannel(module, channel)	Represents a Digital Output channel.
DigitalOutputTrigger(module)	Represents the trigger unit of a digital output device.

16.15.1 dwfpy.digital_output.DigitalOutput

class DigitalOutput(*device*)

Bases: object

Digital Output module (Pattern Generator).

Methods

configure	Configures and starts the instrument.
read_status	Gets the instrument status.
reset	Resets and configures all instrument parameters to default values.
setup	Sets up the pattern generator.
setup_trigger	Sets up the trigger condition.

Attributes

channels	Gets a collection of Digital Output channels.
clock_frequency	Gets the internal clock frequency in Hz.
device	Gets the device.
repeat_count	Gets or sets the repeat count.
repeat_count_max	Gets the maximum repeat count.
repeat_count_min	Gets the minimum repeat count.
repeat_count_status	Gets the remaining repeat count.
run_length	Gets or sets the run length in seconds.
run_length_max	Gets the maximum run length in seconds.
run_length_min	Gets the minimum run length in seconds.
run_length_status	Gets the remaining run length in seconds.
trigger	Gets the trigger unit.
wait_length	Gets or sets the wait length in seconds.
wait_length_max	Gets the maximum wait length in seconds.
wait_length_min	Gets the minimum wait length in seconds.

property channels: Tuple[*DigitalOutputChannel*, ...]

Gets a collection of Digital Output channels.

Return type
Tuple[*DigitalOutputChannel*, ...]

property clock_frequency: float

Gets the internal clock frequency in Hz.

Return type
float

configure(*start=False*)

Configures and starts the instrument.

Return type
None

property device: *Device*

Gets the device.

Return type
Device

read_status()

Gets the instrument status.

Return type
Status

property repeat_count: int

Gets or sets the repeat count.

Return type
int

property repeat_count_max: int

Gets the maximum repeat count.

Return type
int

property repeat_count_min: int

Gets the minimum repeat count.

Return type
int

property repeat_count_status: float

Gets the remaining repeat count.

Return type
float

reset()

Resets and configures all instrument parameters to default values.

Return type
None

property run_length: float

Gets or sets the run length in seconds.

Return type
float

property run_length_max: float

Gets the maximum run length in seconds.

Return type
float

property run_length_min: float

Gets the minimum run length in seconds.

Return type
float

property run_length_status: float

Gets the remaining run length in seconds.

Return type
float

setup(*run_length=None*, *wait_length=None*, *repeat_count=None*, *configure=False*, *start=False*)

Sets up the pattern generator.

run_length
[float, optional] The run length in seconds.

wait_length
[float, optional] The wait length in seconds.

repeat_count
[int, optional] The repeat count.

configure
[bool, optional] If True, then the instrument is configured (default False).

start
[bool, optional] If True, then the instrument is started (default False).

Return type
None

setup_trigger(*source=None*, *slope=None*)

Sets up the trigger condition.

source
[str, optional] The trigger source. Can be 'none', 'pc', 'detector-analog-in', 'detector-digital-in', 'analog-in', 'digital-in', 'digital-out', 'analog-out1', 'analog-out2', 'analog-out3', 'analog-out3', 'external1', 'external2', 'external3', 'external4', 'low', 'high', or 'clock'.

slope
[str, optional] The trigger slope. Can be 'rising', 'falling', or 'either'.

Return type
None

property trigger: *DigitalOutputTrigger*

Gets the trigger unit.

Return type
DigitalOutputTrigger

property wait_length: float

Gets or sets the wait length in seconds.

Return type
float

property wait_length_max: float

Gets the maximum wait length in seconds.

Return type
float

property wait_length_min: float

Gets the minimum wait length in seconds.

Return type
float

16.15.2 dwfpy.digital_output.DigitalOutputChannel

class DigitalOutputChannel(*module*, *channel*)

Bases: object

Represents a Digital Output channel.

Methods

set_counter	Sets the low and high counter values.
set_custom_bits	Sets the custom data bits.
set_initial_state_and_counter	Sets the initial state and the initial counter.
setup	Sets up the channel.
setup_clock	Sets up the channel as a clock output.
setup_constant	Sets up the channel as a constant output.
setup_custom	Sets up the channel with a custom output.
setup_pulse	Sets up the channel as a pulse output.
setup_random	Sets up the channel as a random output.

Attributes

counter_max	Gets the maximum supported clock counter value.
counter_min	Gets the minimum supported clock counter value.
device	Gets the device.
divider	Gets or sets the divider value.
divider_max	Gets the maximum supported clock divider value.
divider_min	Gets the minimum supported clock divider value.
enabled	Enables or disables the channel.
high_counter	Gets the high counter value.
idle_state	Gets or sets the idle output state.
idle_state_info	Gets the supported idle output states.
index	Gets the channel index.
initial_counter	Gets the initial counter value.
initial_divider	Gets or sets the initial divider value.
initial_state	Gets the initial state.
label	Gets or sets the channel label.
low_counter	Gets the low counter value.
max_bits	Gets the maximum number of bits.
module	Gets the Digital Output module.
output_mode	Gets or sets the output mode.
output_mode_info	Gets the supported output modes.
output_type	Gets or sets the output type.
output_type_info	Gets the supported output types.
repetition	Gets or sets the repetition value.
repetition_max	Gets the maximum repetition value.

property counter_max: int

Gets the maximum supported clock counter value.

Return type
int

property counter_min: int

Gets the minimum supported clock counter value.

Return type
int

property device: *Device*

Gets the device.

Return type
Device

property divider: int

Gets or sets the divider value.

Return type
int

property divider_max: int

Gets the maximum supported clock divider value.

Return type
int

property divider_min: int

Gets the minimum supported clock divider value.

Return type
int

property enabled: bool

Enables or disables the channel.

Return type
bool

property high_counter: int

Gets the high counter value.

Return type
int

property idle_state: *DigitalOutputIdle*

Gets or sets the idle output state.

Return type
DigitalOutputIdle

property idle_state_info: Tuple[*DigitalOutputIdle*, ...]

Gets the supported idle output states.

Return type
Tuple[*DigitalOutputIdle*, ...]

property index: int

Gets the channel index.

Return type
int

property initial_counter: int

Gets the initial counter value.

Return type
int

property initial_divider: int

Gets or sets the initial divider value.

Return type
int

property initial_state: bool

Gets the initial state.

Return type
bool

property label: str

Gets or sets the channel label.

Return type
str

property low_counter: int

Gets the low counter value.

Return type
int

property max_bits: int

Gets the maximum number of bits.

Return type
int

property module: *DigitalOutput*

Gets the Digital Output module.

Return type
DigitalOutput

property output_mode: *DigitalOutputMode*

Gets or sets the output mode.

Return type
DigitalOutputMode

property output_mode_info: Tuple[*DigitalOutputMode*, ...]

Gets the supported output modes.

Return type
Tuple[*DigitalOutputMode*, ...]

property output_type: *DigitalOutputType*

Gets or sets the output type.

Return type
DigitalOutputType

property output_type_info: Tuple[*DigitalOutputType*, ...]

Gets the supported output types.

Return type
Tuple[*DigitalOutputType*, ...]

property repetition: int

Gets or sets the repetition value.

Return type
int

property repetition_max: int

Gets the maximum repetition value.

Return type
int

set_counter(*low*, *high*)

Sets the low and high counter values.

Return type
None

`set_custom_bits`(*data*)

Sets the custom data bits.

Return type
None

`set_initial_state_and_counter`(*state*, *counter*)

Sets the initial state and the initial counter.

Return type
None

`setup`(*output_type=None*, *output_mode=None*, *divider=None*, *low_counter=None*, *high_counter=None*, *initial_divider=None*, *initial_counter=None*, *initial_state=None*, *idle_state=None*, *repetition=None*, *enabled=True*, *configure=False*, *start=False*)

Sets up the channel.

output_type
[str, optional] The output type. Can be 'pulse', 'custom', 'random', 'rom', 'state', or 'play'.

output_mode
[str, optional] The output mode. Can be 'push-pull', 'open-drain', 'open-source', or 'three-state'.

divider
[int, optional] The divider value.

low_counter
[int, optional] The low counter value.

high_counter
[int, optional] The high counter value.

initial_divider
[int, optional] The initial divider value.

initial_counter
[int, optional] The initial counter value.

initial_state
[str or bool, optional] The initial output state. Can be 'low' or 'high'.

idle_state
[str or DigitalOutputIdle, optional] The output idle state. Can be 'initial', 'low', 'high', or 'z'.

repetition
[int, optional] The repetition value. Set to 0 for unlimited repetitions.

enabled
[bool, optional] If True, then the channel is enabled (default True).

configure
[bool, optional] If True, then the instrument is configured (default False).

start
[bool, optional] If True, then the instrument is started (default False).

Return type
None

`setup_clock`(*frequency*, *duty_cycle=50*, *phase=0*, *delay=0*, *repetition=0*, *output_mode=None*, *idle_state=None*, *enabled=True*, *configure=False*, *start=False*)

Sets up the channel as a clock output.

frequency
[float] The frequency in Hz.

duty_cycle
[float, optional] The duty-cycle in percent (default 50).

phase
[float, optional] The phase in degrees (default 0).

delay
[float, optional] The delay in seconds (default 0).

repetition
[int, optional] The repetition count. Set to 0 for unlimited repetitions (default).

output_mode
[str, optional] The output mode. Can be 'push-pull', 'open-drain', 'open-source', or 'three-state'.

idle_state
[str or DigitalOutputIdle, optional] The output idle state. Can be 'initial', 'low', 'high', or 'z'.

enabled
[bool, optional] If True, then the channel is enabled (default True).

configure
[bool, optional] If True, then the instrument is configured (default False).

start
[bool, optional] If True, then the instrument is started (default False).

Return type
`None`

`setup_constant`(*value*, *output_mode=None*, *idle_state=None*, *enabled=True*, *configure=False*, *start=False*)

Sets up the channel as a constant output.

value
[str or bool] The constant output value. Can be 'low' or 'high'.

output_mode
[str, optional] The output mode. Can be 'push-pull', 'open-drain', 'open-source', or 'three-state'.

idle_state
[str or DigitalOutputIdle, optional] The output idle state. Can be 'initial', 'low', 'high', or 'z'.

enabled
[bool, optional] If True, then the channel is enabled (default True).

configure
[bool, optional] If True, then the instrument is configured (default False).

start
[bool, optional] If True, then the instrument is started (default False).

Return type
`None`

`setup_custom`(*frequency*, *data*, *delay=0*, *repetition=0*, *output_mode=None*, *idle_state=None*, *enabled=True*, *configure=False*, *start=False*)

Sets up the channel with a custom output.

frequency
[float] The bit rate in bits/second.

data
[[int]] An array containing the pattern of ones and zeros.

delay
[float, optional] The delay in seconds (default 0).

repetition
[int, optional] The repetition count. Set to 0 for unlimited repetitions (default).

output_mode
[str, optional] The output mode. Can be 'push-pull', 'open-drain', 'open-source', or 'three-state'.

idle_state
[str or DigitalOutputIdle, optional] The output idle state. Can be 'initial', 'low', 'high', or 'z'.

enabled
[bool, optional] If True, then the channel is enabled (default True).

configure
[bool, optional] If True, then the instrument is configured (default False).

start
[bool, optional] If True, then the instrument is started (default False).

Return type
`None`

`setup_pulse`(*low*, *high*, *delay=0*, *repetition=0*, *output_mode=None*, *initial_state=None*, *idle_state=None*, *enabled=True*, *configure=False*, *start=False*)

Sets up the channel as a pulse output.

low
[float] The duration of the low state in seconds.

high
[float] The duration of the high state in seconds.

delay
[float, optional] The initial delay in seconds (default 0).

repetition
[int, optional] The repetition count. Set to 0 for unlimited repetitions (default).

output_mode
[str, optional] The output mode. Can be 'push-pull', 'open-drain', 'open-source', or 'three-state'.

initial_state
[str or bool, optional] The initial output state. Can be 'low' or 'high'.

idle_state
[str or DigitalOutputIdle, optional] The output idle state. Can be 'initial', 'low', 'high', or 'z'.

enabled
[bool, optional] If True, then the channel is enabled (default True).

configure
[bool, optional] If True, then the instrument is configured (default False).

start
[bool, optional] If True, then the instrument is started (default False).

Return type
None

`setup_random`(*frequency*, *delay=0*, *repetition=0*, *output_mode=None*, *idle_state=None*, *enabled=True*, *configure=False*, *start=False*)

Sets up the channel as a random output.

frequency
[float] The bit rate in bits/second.

delay
[float, optional] The delay in seconds (default 0).

repetition
[int, optional] The repetition count. Set to 0 for unlimited repetitions (default).

output_mode
[str, optional] The output mode. Can be 'push-pull', 'open-drain', 'open-source', or 'three-state'.

idle_state
[str or DigitalOutputIdle, optional] The output idle state. Can be 'initial', 'low', 'high', or 'z'.

enabled
[bool, optional] If True, then the channel is enabled (default True).

configure
[bool, optional] If True, then the instrument is configured (default False).

start
[bool, optional] If True, then the instrument is started (default False).

Return type
None

16.15.3 dwfpy.digital_output.DigitalOutputTrigger

`class DigitalOutputTrigger`(*module*)

Bases: `object`

Represents the trigger unit of a digital output device.

Methods

Attributes

repeat	Gets or sets the repeat trigger option.
slope	Gets or sets the trigger slope for the instrument.
source	Gets or sets the current trigger source setting for the instrument.

property repeat: bool

Gets or sets the repeat trigger option. To include the trigger in wait-run repeat cycles, set 'repeat' to True.

Return type
bool

property slope: *TriggerSlope*

Gets or sets the trigger slope for the instrument.

Return type
TriggerSlope

property source: *TriggerSource*

Gets or sets the current trigger source setting for the instrument.

Return type
TriggerSource

16.16 dwfpy.digital_recorder

Recorder for Digital Input data.

Classes

DigitalRecorder(module)	Recorder for Digital Input data

16.16.1 dwfpy.digital_recorder.DigitalRecorder

class DigitalRecorder(*module*)

Bases: object

Recorder for Digital Input data

Methods

process	Checks the instrument status and processes a chunk of data if available.
record	Starts the recording and processes all samples until the recording is complete.
stream	Starts the streaming.

Attributes

corrupted_samples	Gets the number of corrupted samples.
data_samples	Gets the acquired data samples.
lost_samples	Gets the number of lost samples.
noise_samples	Gets the acquired noise samples.
requested_samples	Gets the number of requested samples for recording.
status	Gets the last acquisition status.
total_samples	Gets the total number of acquired and lost samples.

property corrupted_samples: int

Gets the number of corrupted samples.

Return type
int

property data_samples: tuple

Gets the acquired data samples.

Return type
tuple

property lost_samples: int

Gets the number of lost samples.

Return type
int

property noise_samples: tuple

Gets the acquired noise samples.

Return type
tuple

process()

Checks the instrument status and processes a chunk of data if available.

bool
If True, then if there is more data to process, and the function must be called again. If False, the recording is complete, and you must stop calling this function.

This function must be called repeatedly by the user to process the recording data. Failure to call this function in a timely manner will cause samples to get lost or corrupted.

Return type
bool

record(*callback=None*)

Starts the recording and processes all samples until the recording is complete.

callback
[function] A user-defined function that is called every time a data chunk is processed. Return True to continue recording, False to abort the recording.

This function blocks until the recording is complete.

Return type
None

property requested_samples: int

Gets the number of requested samples for recording.

Return type
int

property status: *Status*

Gets the last acquisition status.

Return type
Status

stream(*callback*)

Starts the streaming.

callback
[function] A user-defined function that is called every time a data chunk is processed. Return True to continue streaming, False to stop the streaming.

Return type
None

property total_samples: int

Gets the total number of acquired and lost samples.

Return type
int

16.17 dwfpy.exceptions

Exceptions for Digilent WaveForms.

Exceptions

DeviceNotFound(message)	Device not found error
DeviceNotOpenError(message)	Device is not open error
WaveformsError(message[, error])	Base class for Digilent WaveForms exceptions.

16.17.1 dwfpy.exceptions.DeviceNotFound

exception DeviceNotFound(*message*)

Device not found error

16.17.2 dwfpy.exceptions.DeviceNotOpenError

exception DeviceNotOpenError(*message*)

Device is not open error

16.17.3 dwfpy.exceptions.WaveformsError

exception WaveformsError(*message*, *error=0*)

Base class for Digilent WaveForms exceptions.

16.18 dwfpy.helpers

Internal helper functions.

Classes

Helpers()	Internal helper class.

16.18.1 dwfpy.helpers.Helpers

class Helpers

Bases: `object`

Internal helper class.

Methods

c_char_to_string	**rtype** str
make_default	
map_acquisition_mode	
map_coupling	
map_digital_output_idle	
map_digital_output_mode	
map_digital_output_type	
map_enum_values	**rtype** tuple
map_filter	
map_function	
map_named_value	
map_state	
map_trigger_length_condition	
map_trigger_slope	
map_trigger_source	
pack_bits	

16.19 dwfpy.protocols

Protocols module for Digilent WaveForms devices.

Classes

Protocols(device)	Digital Protocols module.

16.19.1 dwfpy.protocols.Protocols

class Protocols(*device*)

Bases: `object`

Digital Protocols module.

Methods

Attributes

can	Gets the CAN protocol unit.
i2c	Gets the I2C protocol unit.
spi	Gets the SPI protocol unit.
uart	Gets the UART protocol unit.

class CAN(*device*)

Bases: `object`

CAN protocol.

property inverted: bool

Gets or sets the polarity.

Return type
`bool`

property pin_rx: int

Gets or sets the DIO channel to use for reception.

Return type
`int`

property pin_tx: int

Gets or sets the DIO channel to use for transmission.

Return type
`int`

property rate: float

Gets or sets the data rate.

Return type
`float`

read()

Returns the received CAN frames since the last call.

Return type
`Tuple[bytes, int, int, int, int]`

reset()

Resets the CAN configuration to default value.

Return type
None

setup(*pin_rx=None*, *pin_tx=None*, *rate=None*, *inverted=False*)

Sets up the CAN configuration.

Return type
None

write(*frame_id*, *extended*, *remote*, *buffer*)

Performs a CAN transmission.

Return type
None

class I2C(*device*)

Bases: object

I2C protocol.

clear()

Verifies and tries to solve eventual bus lockup. Returns true, if the bus is free.

Return type
bool

property pin_scl: int

Gets or sets the DIO channel to use for I2C clock.

Return type
int

property pin_sda: int

Gets or sets the DIO channel to use for I2C data.

Return type
int

property rate: float

Gets or sets the data rate.

Return type
float

read(*address*, *bytes_to_read*)

Performs an I2C read. Returns (rx_buffer, nak_index).

Return type
Tuple[bytes, int]

property read_nak: bool

Gets or sets a value indicating if the last read byte should be acknowledged or not.

Return type
bool

reset()

Resets the I2C configuration to default value.

Return type
None

setup(*pin_scl*, *pin_sda*, *rate=None*, *timeout=None*, *read_nak=None*, *stretch=None*)

Sets up the I2C configuration.

Return type
None

property stretch: bool

Enables or disables clock stretching.

Return type
bool

property timeout: float

Gets or sets the time-out.

Return type
float

write(*address*, *buffer*)

Performs an I2C write.

Return type
None

write_one(*address*, *data*)

Performs an I2C write of a single byte.

Return type
None

write_read(*address*, *buffer*, *bytes_to_read*)

Performs an I2C write/read. Returns (rx_buffer, nak_index).

Return type
Tuple[bytes, int]

class Spi(*device*)

Bases: object

SPI protocol.

property frequency: float

Gets or sets the DIO channel to use for SPI data.

Return type
float

property mode: int

Gets or sets the SPI mode.

Return type
int

property msb_first: bool

Gets or sets the bit order for SPI data.

Return type
bool

property pin_clock: int

Gets or sets the DIO channel to use for SPI clock.

Return type
int

property pin_dq0: int

Gets or sets the DIO channel to use for SPI data.

Return type
int

property pin_dq1: int

Gets or sets the DIO channel to use for SPI data.

Return type
int

property pin_dq2: int

Gets or sets the DIO channel to use for SPI data.

Return type
int

property pin_dq3: int

Gets or sets the DIO channel to use for SPI data.

Return type
int

property pin_select: int

Gets or sets the DIO channel to use for SPI clock.

Return type
int

read(*words_to_receive*, *dq_mode=None*, *bits_per_word=8*)

Performs a SPI read.

Return type
Union[bytes, array]

read_one(*dq_mode=None*, *bits_per_word=8*)

Performs a SPI reception of up to 32 bits.

Return type
None

reset()

Resets the SPI configuration to default value.

Return type
None

select(*level*, *pin_select=None*)

Control the SPI chip select.

Return type
None

set_idle(*pin*, *idle*)

Specifies the DQ signal idle output state. DQ2 and DQ3 may be used for alternative purpose like for write protect (should be driven low) or for hold (should be in high impendance).

Return type
None

setup(*pin_clock*, *pin_mosi*, *pin_miso=None*, *pin_select=None*, *frequency=None*, *mode=0*, *msb_first=True*)

Sets up the SPI pin configuration in standard mode.

Return type
None

setup_dual(*pin_clock*, *pin_dq0*, *pin_dq1*, *pin_select=None*, *frequency=None*, *mode=0*, *msb_first=True*)

Sets up the SPI pin configuration in Dual mode.

Return type
None

setup_quad(*pin_clock*, *pin_dq0*, *pin_dq1*, *pin_dq2*, *pin_dq3*, *pin_select=None*, *frequency=None*, *mode=0*, *msb_first=True*)

Sets up the SPI pin configuration in Quad mode.

Return type
None

setup_three_wire(*pin_clock*, *pin_siso*, *pin_select=None*, *frequency=None*, *mode=0*, *msb_first=True*)

Sets up the SPI pin configuration in Three-wire mode.

Return type
None

write(*buffer*, *dq_mode=None*, *bits_per_word=8*)

Performs a SPI write.

Return type
None

write_one(*data*, *dq_mode=None*, *bits_per_word=8*)

Performs a SPI transmit of up to 32 bits.

Return type
None

write_read(*buffer*, *words_to_receive*, *dq_mode=None*, *bits_per_word=8*)

Performs a SPI write/read.

Return type
Union[bytes, array]

class Uart(*device*)

Bases: object

UART protocol.

property data_bits: Optional[int]

Gets or sets the number of data bits.

Return type
Optional[int]

property inverted: bool

Gets or sets the polarity.

Return type
bool

property parity: str

Gets or sets the parity.

Return type
str

property pin_rx: Optional[int]

Gets or sets the DIO channel to use for reception.

Return type
Optional[int]

property pin_tx: Optional[int]

Gets or sets the DIO channel to use for transmission.

Return type
Optional[int]

property rate: float

Gets or sets the baud rate.

Return type
`float`

read(*buffer_size=8192*)

Returns the received characters since the last call. Returns (rx_buffer, parity).

Return type
`Tuple[bytes, int]`

reset()

Resets the UART configuration to default value.

Return type
`None`

setup(*pin_rx=None*, *pin_tx=None*, *rate=9600*, *data_bits=8*, *stop_bits=1*, *parity='n'*, *inverted=False*)

Sets up the UART configuration.

Return type
`None`

property stop_bits: float

Gets or sets the number of stop bits.

Return type
`float`

write(*buffer*)

Transmits the specified characters.

Return type
`None`

property can: *CAN*

Gets the CAN protocol unit.

Return type
CAN

property i2c: *I2C*

Gets the I2C protocol unit.

Return type
I2C

property spi: *Spi*

Gets the SPI protocol unit.

Return type
Spi

property uart: *Uart*

Gets the UART protocol unit.

Return type
Uart

PYTHON MODULE INDEX

d

INDEX

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W