

---

**dwpfy**  
*Release 1.1*

**Marius Greuel**

**Jul 20, 2022**



# INTRODUCTION

<b>1</b>	<b>Overview</b>	<b>1</b>
<b>2</b>	<b>Features</b>	<b>3</b>
<b>3</b>	<b>Installing DwfPy</b>	<b>5</b>
<b>4</b>	<b>Documentation</b>	<b>7</b>
<b>5</b>	<b>Examples</b>	<b>9</b>
<b>6</b>	<b>Getting help</b>	<b>11</b>
<b>7</b>	<b>Working with Devices</b>	<b>13</b>
7.1	Creating a Device Instance . . . . .	13
7.2	Using a Specific Device Instance . . . . .	13
7.3	Filtering Devices . . . . .	13
7.4	Enumerating Devices . . . . .	14
<b>8</b>	<b>Using the Oscilloscope</b>	<b>15</b>
8.1	Setting Up Channels . . . . .	15
8.2	Getting the Current ADC Sample . . . . .	16
8.3	Single Data Acquisition . . . . .	16
8.4	Recording Samples . . . . .	16
8.5	Setting Up an Edge Trigger . . . . .	17
<b>9</b>	<b>Using the Waveform Generator</b>	<b>19</b>
9.1	Setting up a new Waveform . . . . .	19
9.2	Setting up an Amplitude Modulation . . . . .	19
9.3	Setting up a Frequency Modulation . . . . .	20
<b>10</b>	<b>Using the Logic Analyzer</b>	<b>21</b>
10.1	Single Data Acquisition . . . . .	21
10.2	Recording Samples . . . . .	22
<b>11</b>	<b>Using the Pattern Generator</b>	<b>23</b>
11.1	Setting up a Constant Output . . . . .	23
11.2	Setting up a Clock Signal . . . . .	23
11.3	Setting up a Random Signal . . . . .	24
<b>12</b>	<b>Using the Digital I/O</b>	<b>25</b>
12.1	Reading from Inputs . . . . .	25
12.2	Writing to Outputs . . . . .	25

<b>13 Using the Power Supplies</b>	<b>27</b>
<b>14 Labelling Channels</b>	<b>29</b>
<b>15 dwfpy Design</b>	<b>31</b>
15.1 Bindings . . . . .	31
15.2 Low-Level API . . . . .	32
15.3 High-Level API . . . . .	32
<b>16 dwfpy</b>	<b>33</b>
16.1 Example . . . . .	33
16.2 Available subpackages . . . . .	33
16.3 dwfpy.analog_input . . . . .	34
16.4 dwfpy.analog_io . . . . .	52
16.5 dwfpy.analog_output . . . . .	56
16.6 dwfpy.analog_recorder . . . . .	67
16.7 dwfpy.application . . . . .	69
16.8 dwfpy.bindings . . . . .	71
16.9 dwfpy.configuration . . . . .	124
16.10 dwfpy.constants . . . . .	126
16.11 dwfpy.device . . . . .	140
16.12 dwfpy.device_info . . . . .	176
16.13 dwfpy.digital_input . . . . .	178
16.14 dwfpy.digital_io . . . . .	193
16.15 dwfpy.digital_output . . . . .	197
16.16 dwfpy.digital_recorder . . . . .	208
16.17 dwfpy.exceptions . . . . .	210
16.18 dwfpy.helpers . . . . .	211
16.19 dwfpy.protocols . . . . .	212
<b>Python Module Index</b>	<b>219</b>
<b>Index</b>	<b>221</b>

## 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.')
```



## **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](#).





## 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>.



## 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](#).



## EXAMPLES

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



## GETTING HELP

For issues with **DwfPy**, please visit the [dwfpy GitHub issue tracker](#).





## 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](#)

## 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](#).



## 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)
```



## 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 dwfp 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](#).

## 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 specify 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](#).

## 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](#).

## 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
```





## 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)
```



## 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 (arbitrary 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 arbitrary 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.

---

*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, off-
    set=1.41, configure=True)
```

## 16.2 Available subpackages

### bindings

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

### Modules

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

## 16.3 dwfpy.analog\_input

Analog Input module for Digilent WaveForms devices.

### Classes

<i>AnalogInput</i> (device)	Analog Input module (Oscilloscope).
<i>AnalogInputChannel</i> (module, channel)	Represents an Analog Input channel.
<i>AnalogInputTrigger</i> (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

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

## Attributes

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

<a href="#"><i>get_data</i></a>	Gets the acquired data samples.
<a href="#"><i>get_noise</i></a>	Gets the acquired noise samples.
<a href="#"><i>get_sample</i></a>	Gets the last ADC conversion sample.
<a href="#"><i>setup</i></a>	Sets up the channel for data acquisition.

#### Attributes

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

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

continues on next page

Table 1 – continued from previous page

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

<i>AnalogIo</i> (device)	Analog IO module.
<i>AnalogIoChannel</i> (module, channel)	Represents an Analog IO channel.
<i>AnalogIoChannelNode</i> (channel, node)	Represents an Analog IO channel node.

### 16.4.1 dwfpy.analog\_io.AnalogIo

**class** *AnalogIo*(device)

Bases: object

Analog IO module.

### Methods

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

### Attributes

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

<i>device</i>	Gets the device.
<i>index</i>	Gets the channel index.
<i>label</i>	Gets the channel label.
<i>module</i>	Gets the Analog IO module.
<i>name</i>	Gets the channel name.
<i>nodes</i>	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

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

<i>AnalogOutput</i> (device)	Analog Output module (Arbitrary Waveform Generator).
<i>AnalogOutputChannel</i> (module, channel)	Represents an Analog Output channel.
<i>AnalogOutputChannelNode</i> (channel, node)	Represents an Analog Output channel node.
<i>AnalogOutputChannelTrigger</i> (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

<code>channels</code>	Gets a collection of Analog Output channels.
<code>device</code>	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

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

## Attributes

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

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

#### Attributes

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

---

<i>slope</i>	Gets or sets the trigger slope for the instrument.
<i>source</i>	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

<i>AnalogRecorder</i> (module)	Recorder for Analog Input data
--------------------------------	--------------------------------

### 16.6.1 dwfpy.analog\_recorder.AnalogRecorder

**class** *AnalogRecorder*(module)

Bases: object

Recorder for Analog Input data

#### Methods

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

#### Attributes

<i>channels</i>	Gets a collection of data channels.
<i>corrupted_samples</i>	Gets the number of corrupted samples.
<i>lost_samples</i>	Gets the number of lost samples.
<i>requested_samples</i>	Gets the number of requested samples for recording.
<i>status</i>	Gets the last acquisition status.
<i>total_samples</i>	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

<i>Application</i> (*args, **kwargs)	WaveForms application.
--------------------------------------	------------------------

### 16.7.1 dwfpy.application.Application

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

Bases: *\_Singleton*

WaveForms application.

#### Methods

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

#### Attributes

<i>clock_mode</i>	Specifies the clock mode: 0 internal, 1 output, 2 input, 3 IO.
<i>enable_analog_out</i>	Enables or disables the analog audio output.
<i>enable_audio_output</i>	Enables or disables audio output.
<i>external_frequency</i>	Specifies the external frequency in Hz.
<i>frequency</i>	Gets or sets the frequency in Hz.
<i>led_brightness</i>	Gets or sets the Digital Discovery LED brightness.
<i>on_close_behavior</i>	Gets or sets a value indicating the device close behavior.
<i>usb_limit</i>	Gets or sets the USB current limitation in mA.
<i>usb_power_on_aux</i>	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

---

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

---

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

---

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

---

continues on next page

Table 2 – continued from previous page

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

continues on next page

Table 2 – continued from previous page

---

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

---

continues on next page



Table 2 – continued from previous page

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

continues on next page

Table 2 – continued from previous page

---

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

---

continues on next page

Table 2 – continued from previous page

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

continues on next page

Table 2 – continued from previous page

---

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

---

continues on next page

Table 2 – continued from previous page

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

continues on next page

Table 2 – continued from previous page

---

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

---

continues on next page

Table 2 – continued from previous page

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

continues on next page

Table 2 – continued from previous page

---

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

---

continues on next page



Table 2 – continued from previous page

<code>dwf_enum(*_)</code>	
<code>dwf_enum_config(*_)</code>	
<code>dwf_enum_config_info(*_)</code>	
<code>dwf_enum_config_info_str(*_)</code>	
<code>dwf_enum_device_is_opened(*_)</code>	
<code>dwf_enum_device_name(*_)</code>	
<code>dwf_enum_device_type(*_)</code>	
<code>dwf_enum_sn(*_)</code>	
<code>dwf_enum_user_name(*_)</code>	
<code>dwf_get_last_error(*_)</code>	
<code>dwf_get_last_error_msg(*_)</code>	
<code>dwf_get_version(*_)</code>	
<code>set_error_handler(handler)</code>	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

---

<code>Configuration(index)</code>	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

<code>analog_in_buffer_size</code>	Gets the Analog Input buffer size.
<code>analog_in_channel_count</code>	Gets the total number of Analog Input channels.
<code>analog_io_channel_count</code>	Gets the total number of Analog IO channels.
<code>analog_out_buffer_size</code>	Gets the Analog Output buffer size.
<code>analog_out_channel_count</code>	Gets the total number of Analog Output channels.
<code>digital_in_buffer_size</code>	Gets the Digital Input buffer size.
<code>digital_in_channel_count</code>	Gets the total number of Digital Input channels.
<code>digital_io_channel_count</code>	Gets the total number of Digital IO channels.
<code>digital_out_buffer_size</code>	Gets the Digital Output buffer size.
<code>digital_out_channel_count</code>	Gets the total number of Digital Output channels.
<code>text_info</code>	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 Diligent WaveForms API.

### Classes

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

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

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

**Attributes**

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

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

**Attributes**

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

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

## Attributes

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

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

**Attributes**

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

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

**Attributes**

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

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

## Attributes

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

---

<code>DeviceInfo([device_index])</code>	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

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

## Attributes

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

<a href="#"><i>DigitalInput</i>(device)</a>	Digital Input module (Logic Analyzer).
<a href="#"><i>DigitalInputChannel</i>(module, channel)</a>	Represents a Digital Input channel.
<a href="#"><i>DigitalInputChannelTrigger</i>(channel)</a>	Represents the trigger unit of a Digital Input channel.
<a href="#"><i>DigitalInputClock</i>(module)</a>	Represents the clock unit of a Digital Input device.
<a href="#"><i>DigitalInputTrigger</i>(module)</a>	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

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

## Attributes

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

<i>setup_reset_trigger</i>	Sets up the trigger reset condition for this channel.
<i>setup_trigger</i>	Sets up the trigger condition for this channel.

#### Attributes

<i>device</i>	Gets the device.
<i>index</i>	Gets the channel index.
<i>label</i>	Gets or sets the channel label.
<i>module</i>	Gets the Digital Input module.
<i>trigger</i>	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

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

<i>divider</i>	Gets or sets the configured clock divider value.
<i>divider_max</i>	Gets the maximum supported clock divider value.
<i>frequency</i>	Gets the internal clock frequency in Hz.
<i>source</i>	Gets or sets the clock source.
<i>source_info</i>	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

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

#### Attributes

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

<i>DigitalIo</i> (device)	Digital IO module.
<i>DigitalIoChannel</i> (module, channel)	Represents an Digital IO channel.

### 16.14.1 dwfpy.digital\_io.DigitalIo

**class** *DigitalIo*(*device*)

Bases: object

Digital IO module.

### Methods

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

## Attributes

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

<i>setup</i>	Sets up the channel.
--------------	----------------------

### Attributes

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

**property can\_read: bool**

Returns True if the pin can be read.

**Return type**  
bool

**property can\_write: bool**

Returns True if 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 output 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

<i>DigitalOutput</i> (device)	Digital Output module (Pattern Generator).
<i>DigitalOutputChannel</i> (module, channel)	Represents a Digital Output channel.
<i>DigitalOutputTrigger</i> (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

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

#### Attributes

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

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

## Attributes

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

<i>repeat</i>	Gets or sets the repeat trigger option.
<i>slope</i>	Gets or sets the trigger slope for the instrument.
<i>source</i>	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

<i>DigitalRecorder</i> (module)	Recorder for Digital Input data
---------------------------------	---------------------------------

### 16.16.1 dwfpy.digital\_recorder.DigitalRecorder

**class DigitalRecorder**(*module*)

Bases: `object`

Recorder for Digital Input data

## Methods

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

## Attributes

<i>corrupted_samples</i>	Gets the number of corrupted samples.
<i>data_samples</i>	Gets the acquired data samples.
<i>lost_samples</i>	Gets the number of lost samples.
<i>noise_samples</i>	Gets the acquired noise samples.
<i>requested_samples</i>	Gets the number of requested samples for recording.
<i>status</i>	Gets the last acquisition status.
<i>total_samples</i>	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

<i>DeviceNotFound</i> (message)	Device not found error
<i>DeviceNotOpenError</i> (message)	Device is not open error
<i>WaveformsError</i> (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 Diligent WaveForms exceptions.

## 16.18 dwfpy.helpers

Internal helper functions.

### Classes

---

<code><i>Helpers()</i></code>	Internal helper class.
-------------------------------	------------------------

---

### 16.18.1 dwfpy.helpers.Helpers

**class** `Helpers`

Bases: `object`

Internal helper class.

## Methods

c_char_to_string	<b>rtype</b> str
make_default	
map_acquisition_mode	
map_coupling	
map_digital_output_idle	
map_digital_output_mode	
map_digital_output_type	
map_enum_values	<b>rtype</b> 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

<i>Protocols</i> (device)	Digital Protocols module.
---------------------------	---------------------------

### 16.19.1 dwfpy.protocols.Protocols

**class** *Protocols*(device)

Bases: object

Digital Protocols module.

#### Methods

#### Attributes

<i>can</i>	Gets the CAN protocol unit.
<i>i2c</i>	Gets the I2C protocol unit.
<i>spi</i>	Gets the SPI protocol unit.
<i>uart</i>	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\_ismo, 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

- [dwfpy](#), 33
- [dwfpy.analog\\_input](#), 34
- [dwfpy.analog\\_io](#), 52
- [dwfpy.analog\\_output](#), 56
- [dwfpy.analog\\_recorder](#), 67
- [dwfpy.application](#), 69
- [dwfpy.bindings](#), 71
- [dwfpy.configuration](#), 124
- [dwfpy.constants](#), 126
- [dwfpy.device](#), 140
- [dwfpy.device\\_info](#), 176
- [dwfpy.digital\\_input](#), 178
- [dwfpy.digital\\_io](#), 193
- [dwfpy.digital\\_output](#), 197
- [dwfpy.digital\\_recorder](#), 208
- [dwfpy.exceptions](#), 210
- [dwfpy.helpers](#), 211
- [dwfpy.protocols](#), 212





## A

- acquisition\_mode (*AnalogInput* property), 35
- acquisition\_mode (*DigitalInput* property), 180
- acquisition\_mode\_info (*AnalogInput* property), 35
- acquisition\_mode\_info (*DigitalInput* property), 180
- AcquisitionMode (*class in dwfpy.constants*), 126
- actual\_position (*AnalogInputTrigger* property), 48
- adc\_bits (*AnalogInputChannel* property), 43
- amplitude (*AnalogOutputChannelNode* property), 63
- amplitude\_max (*AnalogOutputChannelNode* property), 64
- amplitude\_min (*AnalogOutputChannelNode* property), 64
- analog\_in\_buffer\_size (*Configuration* property), 125
- analog\_in\_channel\_count (*Configuration* property), 125
- analog\_input (*AnalogDiscovery* property), 143
- analog\_input (*AnalogDiscovery2* property), 149
- analog\_input (*Device* property), 154
- analog\_input (*DeviceBase* property), 159
- analog\_input (*DigitalDiscovery* property), 164
- analog\_input (*ElectronicsExplorer* property), 173
- analog\_io (*AnalogDiscovery* property), 143
- analog\_io (*AnalogDiscovery2* property), 150
- analog\_io (*Device* property), 154
- analog\_io (*DeviceBase* property), 159
- analog\_io (*DigitalDiscovery* property), 164
- analog\_io (*ElectronicsExplorer* property), 173
- analog\_io\_channel\_count (*Configuration* property), 125
- analog\_out\_buffer\_size (*Configuration* property), 125
- analog\_out\_channel\_count (*Configuration* property), 125
- analog\_output (*AnalogDiscovery* property), 144
- analog\_output (*AnalogDiscovery2* property), 150
- analog\_output (*Device* property), 155
- analog\_output (*DeviceBase* property), 159
- analog\_output (*DigitalDiscovery* property), 164
- analog\_output (*ElectronicsExplorer* property), 173
- AnalogDiscovery (*class in dwfpy.device*), 141
- AnalogDiscovery.Supplies (*class in dwfpy.device*), 142
- AnalogDiscovery.Supplies.Negative (*class in dwfpy.device*), 142
- AnalogDiscovery.Supplies.Positive (*class in dwfpy.device*), 142
- AnalogDiscovery2 (*class in dwfpy.device*), 147
- AnalogDiscovery2.Supplies (*class in dwfpy.device*), 148
- AnalogDiscovery2.Supplies.Negative (*class in dwfpy.device*), 148
- AnalogDiscovery2.Supplies.Positive (*class in dwfpy.device*), 149
- AnalogImpedance (*class in dwfpy.constants*), 127
- AnalogInput (*class in dwfpy.analog\_input*), 34
- AnalogInputChannel (*class in dwfpy.analog\_input*), 43
- AnalogInputCoupling (*class in dwfpy.constants*), 129
- AnalogInputTrigger (*class in dwfpy.analog\_input*), 47
- AnalogIo (*class in dwfpy.analog\_io*), 52
- AnalogIoChannel (*class in dwfpy.analog\_io*), 54
- AnalogIoChannelNode (*class in dwfpy.analog\_io*), 55
- AnalogOutput (*class in dwfpy.analog\_output*), 57
- AnalogOutputChannel (*class in dwfpy.analog\_output*), 57
- AnalogOutputChannelNode (*class in dwfpy.analog\_output*), 63
- AnalogOutputChannelTrigger (*class in dwfpy.analog\_output*), 66
- AnalogOutputIdle (*class in dwfpy.constants*), 129
- AnalogOutputMode (*class in dwfpy.constants*), 129
- AnalogOutputNode (*class in dwfpy.constants*), 130
- AnalogRecorder (*class in dwfpy.analog\_recorder*), 67
- AnalogRecorder.ChannelData (*class in dwfpy.analog\_recorder*), 67
- application (*AnalogDiscovery* property), 144
- application (*AnalogDiscovery2* property), 150
- Application (*class in dwfpy.application*), 69
- application (*Device* property), 155
- application (*DeviceBase* property), 159
- application (*DigitalDiscovery* property), 164
- application (*ElectronicsExplorer* property), 173
- apply() (*AnalogOutputChannel* method), 58
- attenuation (*AnalogInputChannel* property), 44

`auto_configure` (*AnalogDiscovery* property), 144  
`auto_configure` (*AnalogDiscovery2* property), 150  
`auto_configure` (*Device* property), 155  
`auto_configure` (*DeviceBase* property), 159  
`auto_configure` (*DigitalDiscovery* property), 164  
`auto_configure` (*ElectronicsExplorer* property), 173  
`auto_reset` (*AnalogDiscovery* property), 144  
`auto_reset` (*AnalogDiscovery2* property), 150  
`auto_reset` (*Device* property), 155  
`auto_reset` (*DeviceBase* property), 160  
`auto_reset` (*DigitalDiscovery* property), 164  
`auto_reset` (*ElectronicsExplorer* property), 173  
`auto_timeout` (*AnalogInputTrigger* property), 48  
`auto_timeout` (*DigitalInputTrigger* property), 191  
`auto_timeout_max` (*AnalogInputTrigger* property), 48  
`auto_timeout_max` (*DigitalInputTrigger* property), 191  
`auto_timeout_min` (*AnalogInputTrigger* property), 48  
`auto_timeout_min` (*DigitalInputTrigger* property), 191  
`auto_timeout_steps` (*AnalogInputTrigger* property), 48  
`auto_timeout_steps` (*DigitalInputTrigger* property), 192  
`auto_triggered` (*AnalogInput* property), 35  
`auto_triggered` (*DigitalInput* property), 180  
`aux_current` (*AnalogDiscovery2* property), 150  
`aux_voltage` (*AnalogDiscovery2* property), 150

## B

`bandwidth` (*AnalogInputChannel* property), 44  
`buffer_size` (*AnalogInput* property), 35  
`buffer_size` (*DigitalInput* property), 180  
`buffer_size_max` (*AnalogInput* property), 35  
`buffer_size_max` (*DigitalInput* property), 180  
`buffer_size_min` (*AnalogInput* property), 36

## C

`can` (*Protocols* property), 218  
`can_read` (*DigitalIoChannel* property), 195  
`can_write` (*DigitalIoChannel* property), 195  
`channel` (*AnalogInputTrigger* property), 48  
`channel_max` (*AnalogInputTrigger* property), 48  
`channel_min` (*AnalogInputTrigger* property), 48  
`ChannelNodeType` (*class in dwfpy.constants*), 130  
`channels` (*AnalogInput* property), 36  
`channels` (*AnalogIo* property), 52  
`channels` (*AnalogOutput* property), 57  
`channels` (*AnalogRecorder* property), 67  
`channels` (*DigitalInput* property), 180  
`channels` (*DigitalIo* property), 194  
`channels` (*DigitalOutput* property), 197  
`clear()` (*Protocols.I2C* method), 214  
`clock` (*DigitalInput* property), 181  
`clock_frequency` (*DigitalOutput* property), 198  
`clock_mode` (*Application* property), 69

`close()` (*AnalogDiscovery* method), 144  
`close()` (*AnalogDiscovery2* method), 150  
`close()` (*Device* method), 155  
`close()` (*DeviceBase* method), 160  
`close()` (*DigitalDiscovery* method), 165  
`close()` (*ElectronicsExplorer* method), 173  
`close_all()` (*Device* static method), 155  
`condition` (*AnalogInputTrigger* property), 48  
`condition_info` (*AnalogInputTrigger* property), 49  
`configuration` (*AnalogDiscovery* property), 144  
`configuration` (*AnalogDiscovery2* property), 150  
`Configuration` (*class in dwfpy.configuration*), 124  
`configuration` (*Device* property), 155  
`configuration` (*DeviceBase* property), 160  
`configuration` (*DigitalDiscovery* property), 165  
`configuration` (*ElectronicsExplorer* property), 174  
`configurations` (*AnalogDiscovery* property), 144  
`configurations` (*AnalogDiscovery2* property), 150  
`configurations` (*Device* property), 155  
`configurations` (*DeviceBase* property), 160  
`configurations` (*DeviceInfo* property), 177  
`configurations` (*DigitalDiscovery* property), 165  
`configurations` (*ElectronicsExplorer* property), 174  
`configure()` (*AnalogInput* method), 36  
`configure()` (*AnalogIo* method), 53  
`configure()` (*AnalogOutputChannel* method), 58  
`configure()` (*DigitalInput* method), 181  
`configure()` (*DigitalIo* method), 194  
`configure()` (*DigitalOutput* method), 198  
`corrupted_samples` (*AnalogRecorder* property), 67  
`corrupted_samples` (*DigitalRecorder* property), 209  
`counter` (*AnalogInput* property), 36  
`counter_max` (*AnalogInput* property), 36  
`counter_max` (*DigitalOutputChannel* property), 201  
`counter_min` (*DigitalOutputChannel* property), 201  
`counter_status` (*AnalogInput* property), 36  
`coupling` (*AnalogInputChannel* property), 44  
`coupling_info` (*AnalogInputChannel* property), 44  
`current` (*ElectronicsExplorer.Supplies.Fixed* property), 169  
`current` (*ElectronicsExplorer.Supplies.Negative* property), 170  
`current` (*ElectronicsExplorer.Supplies.Positive* property), 170  
`current_limit` (*ElectronicsExplorer.Supplies.Negative* property), 170  
`current_limit` (*ElectronicsExplorer.Supplies.Positive* property), 170

## D

`data_bits` (*Protocols.Uart* property), 217  
`data_samples` (*AnalogRecorder.ChannelData* property), 67  
`data_samples` (*DigitalRecorder* property), 209

- data\_samples\_max (*AnalogOutputChannelNode* property), 64
- data\_samples\_min (*AnalogOutputChannelNode* property), 64
- device (*AnalogInput* property), 36
- device (*AnalogInputChannel* property), 44
- device (*AnalogIo* property), 53
- device (*AnalogIoChannel* property), 54
- device (*AnalogOutput* property), 57
- device (*AnalogOutputChannel* property), 58
- Device (class in *dwfpy.device*), 153
- device (*DigitalInput* property), 181
- device (*DigitalInputChannel* property), 187
- device (*DigitalIo* property), 194
- device (*DigitalIoChannel* property), 196
- device (*DigitalOutput* property), 198
- device (*DigitalOutputChannel* property), 201
- DeviceBase (class in *dwfpy.device*), 158
- DeviceId (class in *dwfpy.constants*), 131
- DeviceInfo (class in *dwfpy.device\_info*), 176
- DeviceNotFound, 211
- DeviceNotOpenError, 211
- DeviceType (class in *dwfpy.constants*), 131
- digital (*DigitalDiscovery.Supplies* property), 164
- digital\_frequency (*DigitalDiscovery* property), 165
- digital\_in\_buffer\_size (*Configuration* property), 125
- digital\_in\_channel\_count (*Configuration* property), 125
- digital\_input (*AnalogDiscovery* property), 144
- digital\_input (*AnalogDiscovery2* property), 151
- digital\_input (*Device* property), 155
- digital\_input (*DeviceBase* property), 160
- digital\_input (*DigitalDiscovery* property), 165
- digital\_input (*ElectronicsExplorer* property), 174
- digital\_io (*AnalogDiscovery* property), 144
- digital\_io (*AnalogDiscovery2* property), 151
- digital\_io (*Device* property), 155
- digital\_io (*DeviceBase* property), 160
- digital\_io (*DigitalDiscovery* property), 165
- digital\_io (*ElectronicsExplorer* property), 174
- digital\_io\_channel\_count (*Configuration* property), 125
- digital\_out\_buffer\_size (*Configuration* property), 125
- digital\_out\_channel\_count (*Configuration* property), 125
- digital\_output (*AnalogDiscovery* property), 144
- digital\_output (*AnalogDiscovery2* property), 151
- digital\_output (*Device* property), 156
- digital\_output (*DeviceBase* property), 160
- digital\_output (*DigitalDiscovery* property), 165
- digital\_output (*ElectronicsExplorer* property), 174
- DigitalDiscovery (class in *dwfpy.device*), 162
- DigitalDiscovery.Supplies (class in *dwfpy.device*), 163
- DigitalDiscovery.Supplies.Digital (class in *dwfpy.device*), 163
- DigitalInput (class in *dwfpy.digital\_input*), 179
- DigitalInputChannel (class in *dwfpy.digital\_input*), 187
- DigitalInputChannelTrigger (class in *dwfpy.digital\_input*), 188
- DigitalInputClock (class in *dwfpy.digital\_input*), 190
- DigitalInputClockSource (class in *dwfpy.constants*), 132
- DigitalInputSampleMode (class in *dwfpy.constants*), 132
- DigitalInputTrigger (class in *dwfpy.digital\_input*), 191
- DigitalIo (class in *dwfpy.digital\_io*), 193
- DigitalIoChannel (class in *dwfpy.digital\_io*), 195
- DigitalOutput (class in *dwfpy.digital\_output*), 197
- DigitalOutputChannel (class in *dwfpy.digital\_output*), 200
- DigitalOutputIdle (class in *dwfpy.constants*), 132
- DigitalOutputMode (class in *dwfpy.constants*), 133
- DigitalOutputTrigger (class in *dwfpy.digital\_output*), 207
- DigitalOutputType (class in *dwfpy.constants*), 133
- DigitalRecorder (class in *dwfpy.digital\_recorder*), 208
- din\_pull\_up\_down (*DigitalDiscovery* property), 165
- dio\_drive\_strength (*DigitalDiscovery* property), 165
- dio\_first (*DigitalInput* property), 181
- dio\_pull\_enable (*DigitalDiscovery* property), 165
- dio\_pull\_up\_down (*DigitalDiscovery* property), 166
- dio\_slew (*DigitalDiscovery* property), 166
- divider (*DigitalInputClock* property), 190
- divider (*DigitalOutputChannel* property), 201
- divider\_max (*DigitalInputClock* property), 190
- divider\_max (*DigitalOutputChannel* property), 201
- divider\_min (*DigitalOutputChannel* property), 202
- DmmMode (class in *dwfpy.constants*), 134
- dwf\_analog\_in\_acquisition\_mode\_get() (in module *dwfpy.bindings*), 85
- dwf\_analog\_in\_acquisition\_mode\_info() (in module *dwfpy.bindings*), 85
- dwf\_analog\_in\_acquisition\_mode\_set() (in module *dwfpy.bindings*), 85
- dwf\_analog\_in\_bits\_info() (in module *dwfpy.bindings*), 86
- dwf\_analog\_in\_buffer\_size\_get() (in module *dwfpy.bindings*), 86
- dwf\_analog\_in\_buffer\_size\_info() (in module *dwfpy.bindings*), 86
- dwf\_analog\_in\_buffer\_size\_set() (in module *dwfpy.bindings*), 86
- dwf\_analog\_in\_channel\_attenuation\_get() (in

*module dwfpy.bindings*), 86

`dwf_analog_in_channel_attenuation_set()` (*in module dwfpy.bindings*), 86

`dwf_analog_in_channel_count()` (*in module dwfpy.bindings*), 86

`dwf_analog_in_channel_enable_get()` (*in module dwfpy.bindings*), 86

`dwf_analog_in_channel_enable_set()` (*in module dwfpy.bindings*), 86

`dwf_analog_in_channel_filter_get()` (*in module dwfpy.bindings*), 87

`dwf_analog_in_channel_filter_info()` (*in module dwfpy.bindings*), 87

`dwf_analog_in_channel_filter_set()` (*in module dwfpy.bindings*), 87

`dwf_analog_in_channel_offset_get()` (*in module dwfpy.bindings*), 87

`dwf_analog_in_channel_offset_info()` (*in module dwfpy.bindings*), 87

`dwf_analog_in_channel_offset_set()` (*in module dwfpy.bindings*), 87

`dwf_analog_in_channel_range_get()` (*in module dwfpy.bindings*), 87

`dwf_analog_in_channel_range_info()` (*in module dwfpy.bindings*), 87

`dwf_analog_in_channel_range_set()` (*in module dwfpy.bindings*), 87

`dwf_analog_in_channel_range_steps()` (*in module dwfpy.bindings*), 88

`dwf_analog_in_configure()` (*in module dwfpy.bindings*), 88

`dwf_analog_in_frequency_get()` (*in module dwfpy.bindings*), 88

`dwf_analog_in_frequency_info()` (*in module dwfpy.bindings*), 88

`dwf_analog_in_frequency_set()` (*in module dwfpy.bindings*), 88

`dwf_analog_in_noise_size_get()` (*in module dwfpy.bindings*), 88

`dwf_analog_in_noise_size_info()` (*in module dwfpy.bindings*), 88

`dwf_analog_in_noise_size_set()` (*in module dwfpy.bindings*), 88

`dwf_analog_in_record_length_get()` (*in module dwfpy.bindings*), 88

`dwf_analog_in_record_length_set()` (*in module dwfpy.bindings*), 89

`dwf_analog_in_reset()` (*in module dwfpy.bindings*), 89

`dwf_analog_in_sampling_delay_get()` (*in module dwfpy.bindings*), 89

`dwf_analog_in_sampling_delay_set()` (*in module dwfpy.bindings*), 89

`dwf_analog_in_sampling_slope_get()` (*in module dwfpy.bindings*), 89

`dwf_analog_in_sampling_slope_set()` (*in module dwfpy.bindings*), 89

`dwf_analog_in_sampling_source_get()` (*in module dwfpy.bindings*), 89

`dwf_analog_in_sampling_source_set()` (*in module dwfpy.bindings*), 89

`dwf_analog_in_status()` (*in module dwfpy.bindings*), 89

`dwf_analog_in_status_auto_triggered()` (*in module dwfpy.bindings*), 90

`dwf_analog_in_status_data()` (*in module dwfpy.bindings*), 90

`dwf_analog_in_status_data16()` (*in module dwfpy.bindings*), 90

`dwf_analog_in_status_data2()` (*in module dwfpy.bindings*), 90

`dwf_analog_in_status_index_write()` (*in module dwfpy.bindings*), 90

`dwf_analog_in_status_noise()` (*in module dwfpy.bindings*), 90

`dwf_analog_in_status_noise2()` (*in module dwfpy.bindings*), 90

`dwf_analog_in_status_record()` (*in module dwfpy.bindings*), 90

`dwf_analog_in_status_sample()` (*in module dwfpy.bindings*), 90

`dwf_analog_in_status_samples_left()` (*in module dwfpy.bindings*), 91

`dwf_analog_in_status_samples_valid()` (*in module dwfpy.bindings*), 91

`dwf_analog_in_trigger_auto_timeout_get()` (*in module dwfpy.bindings*), 91

`dwf_analog_in_trigger_auto_timeout_info()` (*in module dwfpy.bindings*), 91

`dwf_analog_in_trigger_auto_timeout_set()` (*in module dwfpy.bindings*), 91

`dwf_analog_in_trigger_channel_get()` (*in module dwfpy.bindings*), 91

`dwf_analog_in_trigger_channel_info()` (*in module dwfpy.bindings*), 91

`dwf_analog_in_trigger_channel_set()` (*in module dwfpy.bindings*), 91

`dwf_analog_in_trigger_condition_get()` (*in module dwfpy.bindings*), 91

`dwf_analog_in_trigger_condition_info()` (*in module dwfpy.bindings*), 92

`dwf_analog_in_trigger_condition_set()` (*in module dwfpy.bindings*), 92

`dwf_analog_in_trigger_filter_get()` (*in module dwfpy.bindings*), 92

`dwf_analog_in_trigger_filter_info()` (*in module dwfpy.bindings*), 92

`dwf_analog_in_trigger_filter_set()` (*in module*



<i>dwfpy.bindings</i> ), 92	<i>dwfpy.bindings</i> ), 95
<code>dwf_analog_in_trigger_force()</code> (in module <i>dwfpy.bindings</i> ), 92	<code>dwf_analog_io_channel_name()</code> (in module <i>dwfpy.bindings</i> ), 95
<code>dwf_analog_in_trigger_hold_off_get()</code> (in module <i>dwfpy.bindings</i> ), 92	<code>dwf_analog_io_channel_node_get()</code> (in module <i>dwfpy.bindings</i> ), 95
<code>dwf_analog_in_trigger_hold_off_info()</code> (in module <i>dwfpy.bindings</i> ), 92	<code>dwf_analog_io_channel_node_info()</code> (in module <i>dwfpy.bindings</i> ), 95
<code>dwf_analog_in_trigger_hold_off_set()</code> (in module <i>dwfpy.bindings</i> ), 92	<code>dwf_analog_io_channel_node_name()</code> (in module <i>dwfpy.bindings</i> ), 95
<code>dwf_analog_in_trigger_hysteresis_get()</code> (in module <i>dwfpy.bindings</i> ), 93	<code>dwf_analog_io_channel_node_set()</code> (in module <i>dwfpy.bindings</i> ), 96
<code>dwf_analog_in_trigger_hysteresis_info()</code> (in module <i>dwfpy.bindings</i> ), 93	<code>dwf_analog_io_channel_node_set_info()</code> (in module <i>dwfpy.bindings</i> ), 96
<code>dwf_analog_in_trigger_hysteresis_set()</code> (in module <i>dwfpy.bindings</i> ), 93	<code>dwf_analog_io_channel_node_status()</code> (in module <i>dwfpy.bindings</i> ), 96
<code>dwf_analog_in_trigger_length_condition_get()</code> (in module <i>dwfpy.bindings</i> ), 93	<code>dwf_analog_io_channel_node_status_info()</code> (in module <i>dwfpy.bindings</i> ), 96
<code>dwf_analog_in_trigger_length_condition_info()</code> (in module <i>dwfpy.bindings</i> ), 93	<code>dwf_analog_io_configure()</code> (in module <i>dwfpy.bindings</i> ), 96
<code>dwf_analog_in_trigger_length_condition_set()</code> (in module <i>dwfpy.bindings</i> ), 93	<code>dwf_analog_io_enable_get()</code> (in module <i>dwfpy.bindings</i> ), 96
<code>dwf_analog_in_trigger_length_get()</code> (in module <i>dwfpy.bindings</i> ), 93	<code>dwf_analog_io_enable_info()</code> (in module <i>dwfpy.bindings</i> ), 96
<code>dwf_analog_in_trigger_length_info()</code> (in module <i>dwfpy.bindings</i> ), 93	<code>dwf_analog_io_enable_set()</code> (in module <i>dwfpy.bindings</i> ), 96
<code>dwf_analog_in_trigger_length_set()</code> (in module <i>dwfpy.bindings</i> ), 93	<code>dwf_analog_io_enable_status()</code> (in module <i>dwfpy.bindings</i> ), 96
<code>dwf_analog_in_trigger_level_get()</code> (in module <i>dwfpy.bindings</i> ), 94	<code>dwf_analog_io_reset()</code> (in module <i>dwfpy.bindings</i> ), 97
<code>dwf_analog_in_trigger_level_info()</code> (in module <i>dwfpy.bindings</i> ), 94	<code>dwf_analog_io_status()</code> (in module <i>dwfpy.bindings</i> ), 97
<code>dwf_analog_in_trigger_level_set()</code> (in module <i>dwfpy.bindings</i> ), 94	<code>dwf_analog_out_configure()</code> (in module <i>dwfpy.bindings</i> ), 97
<code>dwf_analog_in_trigger_position_get()</code> (in module <i>dwfpy.bindings</i> ), 94	<code>dwf_analog_out_count()</code> (in module <i>dwfpy.bindings</i> ), 97
<code>dwf_analog_in_trigger_position_info()</code> (in module <i>dwfpy.bindings</i> ), 94	<code>dwf_analog_out_custom_am_fm_enable_get()</code> (in module <i>dwfpy.bindings</i> ), 97
<code>dwf_analog_in_trigger_position_set()</code> (in module <i>dwfpy.bindings</i> ), 94	<code>dwf_analog_out_custom_am_fm_enable_set()</code> (in module <i>dwfpy.bindings</i> ), 97
<code>dwf_analog_in_trigger_position_status()</code> (in module <i>dwfpy.bindings</i> ), 94	<code>dwf_analog_out_idle_get()</code> (in module <i>dwfpy.bindings</i> ), 97
<code>dwf_analog_in_trigger_source_get()</code> (in module <i>dwfpy.bindings</i> ), 94	<code>dwf_analog_out_idle_info()</code> (in module <i>dwfpy.bindings</i> ), 97
<code>dwf_analog_in_trigger_source_set()</code> (in module <i>dwfpy.bindings</i> ), 94	<code>dwf_analog_out_idle_set()</code> (in module <i>dwfpy.bindings</i> ), 97
<code>dwf_analog_in_trigger_type_get()</code> (in module <i>dwfpy.bindings</i> ), 95	<code>dwf_analog_out_limitation_get()</code> (in module <i>dwfpy.bindings</i> ), 98
<code>dwf_analog_in_trigger_type_info()</code> (in module <i>dwfpy.bindings</i> ), 95	<code>dwf_analog_out_limitation_info()</code> (in module <i>dwfpy.bindings</i> ), 98
<code>dwf_analog_in_trigger_type_set()</code> (in module <i>dwfpy.bindings</i> ), 95	<code>dwf_analog_out_limitation_set()</code> (in module <i>dwfpy.bindings</i> ), 98
<code>dwf_analog_io_channel_count()</code> (in module <i>dwfpy.bindings</i> ), 95	<code>dwf_analog_out_master_get()</code> (in module <i>dwfpy.bindings</i> ), 98
<code>dwf_analog_io_channel_info()</code> (in module	<code>dwf_analog_out_master_set()</code> (in module

<i>dwfpy.bindings</i> ), 98		<i>dwfpy.bindings</i> ), 101	
<code>dwf_analog_out_mode_get()</code> (in module <i>dwfpy.bindings</i> ), 98		<code>dwf_analog_out_repeat_get()</code> (in module <i>dwfpy.bindings</i> ), 101	
<code>dwf_analog_out_mode_set()</code> (in module <i>dwfpy.bindings</i> ), 98		<code>dwf_analog_out_repeat_info()</code> (in module <i>dwfpy.bindings</i> ), 101	
<code>dwf_analog_out_node_amplitude_get()</code> (in module <i>dwfpy.bindings</i> ), 98		<code>dwf_analog_out_repeat_set()</code> (in module <i>dwfpy.bindings</i> ), 101	
<code>dwf_analog_out_node_amplitude_info()</code> (in module <i>dwfpy.bindings</i> ), 98		<code>dwf_analog_out_repeat_status()</code> (in module <i>dwfpy.bindings</i> ), 101	
<code>dwf_analog_out_node_amplitude_set()</code> (in module <i>dwfpy.bindings</i> ), 99		<code>dwf_analog_out_repeat_trigger_get()</code> (in module <i>dwfpy.bindings</i> ), 102	
<code>dwf_analog_out_node_data_info()</code> (in module <i>dwfpy.bindings</i> ), 99		<code>dwf_analog_out_repeat_trigger_set()</code> (in module <i>dwfpy.bindings</i> ), 102	
<code>dwf_analog_out_node_data_set()</code> (in module <i>dwfpy.bindings</i> ), 99		<code>dwf_analog_out_reset()</code> (in module <i>dwfpy.bindings</i> ), 102	
<code>dwf_analog_out_node_enable_get()</code> (in module <i>dwfpy.bindings</i> ), 99		<code>dwf_analog_out_run_get()</code> (in module <i>dwfpy.bindings</i> ), 102	
<code>dwf_analog_out_node_enable_set()</code> (in module <i>dwfpy.bindings</i> ), 99		<code>dwf_analog_out_run_info()</code> (in module <i>dwfpy.bindings</i> ), 102	
<code>dwf_analog_out_node_frequency_get()</code> (in module <i>dwfpy.bindings</i> ), 99		<code>dwf_analog_out_run_set()</code> (in module <i>dwfpy.bindings</i> ), 102	
<code>dwf_analog_out_node_frequency_info()</code> (in module <i>dwfpy.bindings</i> ), 99		<code>dwf_analog_out_run_status()</code> (in module <i>dwfpy.bindings</i> ), 102	
<code>dwf_analog_out_node_frequency_set()</code> (in module <i>dwfpy.bindings</i> ), 99		<code>dwf_analog_out_status()</code> (in module <i>dwfpy.bindings</i> ), 102	
<code>dwf_analog_out_node_function_get()</code> (in module <i>dwfpy.bindings</i> ), 99		<code>dwf_analog_out_trigger_slope_get()</code> (in module <i>dwfpy.bindings</i> ), 102	
<code>dwf_analog_out_node_function_info()</code> (in module <i>dwfpy.bindings</i> ), 100		<code>dwf_analog_out_trigger_slope_set()</code> (in module <i>dwfpy.bindings</i> ), 103	
<code>dwf_analog_out_node_function_set()</code> (in module <i>dwfpy.bindings</i> ), 100		<code>dwf_analog_out_trigger_source_get()</code> (in module <i>dwfpy.bindings</i> ), 103	
<code>dwf_analog_out_node_info()</code> (in module <i>dwfpy.bindings</i> ), 100		<code>dwf_analog_out_trigger_source_set()</code> (in module <i>dwfpy.bindings</i> ), 103	
<code>dwf_analog_out_node_offset_get()</code> (in module <i>dwfpy.bindings</i> ), 100		<code>dwf_analog_out_wait_get()</code> (in module <i>dwfpy.bindings</i> ), 103	
<code>dwf_analog_out_node_offset_info()</code> (in module <i>dwfpy.bindings</i> ), 100		<code>dwf_analog_out_wait_info()</code> (in module <i>dwfpy.bindings</i> ), 103	
<code>dwf_analog_out_node_offset_set()</code> (in module <i>dwfpy.bindings</i> ), 100		<code>dwf_analog_out_wait_set()</code> (in module <i>dwfpy.bindings</i> ), 103	
<code>dwf_analog_out_node_phase_get()</code> (in module <i>dwfpy.bindings</i> ), 100		<code>dwf_device_auto_configure_get()</code> (in module <i>dwfpy.bindings</i> ), 103	
<code>dwf_analog_out_node_phase_info()</code> (in module <i>dwfpy.bindings</i> ), 100		<code>dwf_device_auto_configure_set()</code> (in module <i>dwfpy.bindings</i> ), 103	
<code>dwf_analog_out_node_phase_set()</code> (in module <i>dwfpy.bindings</i> ), 100		<code>dwf_device_close()</code> (in module <i>dwfpy.bindings</i> ), 103	
<code>dwf_analog_out_node_play_data()</code> (in module <i>dwfpy.bindings</i> ), 101		<code>dwf_device_close_all()</code> (in module <i>dwfpy.bindings</i> ), 104	
<code>dwf_analog_out_node_play_status()</code> (in module <i>dwfpy.bindings</i> ), 101		<code>dwf_device_config_open()</code> (in module <i>dwfpy.bindings</i> ), 104	
<code>dwf_analog_out_node_symmetry_get()</code> (in module <i>dwfpy.bindings</i> ), 101		<code>dwf_device_enable_set()</code> (in module <i>dwfpy.bindings</i> ), 104	
<code>dwf_analog_out_node_symmetry_info()</code> (in module <i>dwfpy.bindings</i> ), 101		<code>dwf_device_open()</code> (in module <i>dwfpy.bindings</i> ), 104	
<code>dwf_analog_out_node_symmetry_set()</code> (in module		<code>dwf_device_reset()</code> (in module <i>dwfpy.bindings</i> ), 104	
		<code>dwf_device_trigger_get()</code> (in module <i>dwfpy.bindings</i> ), 104	

<code>dwf_device_trigger_info()</code> (in module <code>dwfpy.bindings</code> ), 104	<code>dwf_digital_in_buffer_size_set()</code> (in module <code>dwfpy.bindings</code> ), 107
<code>dwf_device_trigger_pc()</code> (in module <code>dwfpy.bindings</code> ), 104	<code>dwf_digital_in_clock_source_get()</code> (in module <code>dwfpy.bindings</code> ), 107
<code>dwf_device_trigger_set()</code> (in module <code>dwfpy.bindings</code> ), 104	<code>dwf_digital_in_clock_source_info()</code> (in module <code>dwfpy.bindings</code> ), 107
<code>dwf_device_trigger_slope_info()</code> (in module <code>dwfpy.bindings</code> ), 105	<code>dwf_digital_in_clock_source_set()</code> (in module <code>dwfpy.bindings</code> ), 108
<code>dwf_digital_can_polarity_set()</code> (in module <code>dwfpy.bindings</code> ), 105	<code>dwf_digital_in_configure()</code> (in module <code>dwfpy.bindings</code> ), 108
<code>dwf_digital_can_rate_set()</code> (in module <code>dwfpy.bindings</code> ), 105	<code>dwf_digital_in_divider_get()</code> (in module <code>dwfpy.bindings</code> ), 108
<code>dwf_digital_can_reset()</code> (in module <code>dwfpy.bindings</code> ), 105	<code>dwf_digital_in_divider_info()</code> (in module <code>dwfpy.bindings</code> ), 108
<code>dwf_digital_can_rx()</code> (in module <code>dwfpy.bindings</code> ), 105	<code>dwf_digital_in_divider_set()</code> (in module <code>dwfpy.bindings</code> ), 108
<code>dwf_digital_can_rx_set()</code> (in module <code>dwfpy.bindings</code> ), 105	<code>dwf_digital_in_input_order_set()</code> (in module <code>dwfpy.bindings</code> ), 108
<code>dwf_digital_can_tx()</code> (in module <code>dwfpy.bindings</code> ), 105	<code>dwf_digital_in_internal_clock_info()</code> (in module <code>dwfpy.bindings</code> ), 108
<code>dwf_digital_can_tx_set()</code> (in module <code>dwfpy.bindings</code> ), 105	<code>dwf_digital_in_reset()</code> (in module <code>dwfpy.bindings</code> ), 108
<code>dwf_digital_i2c_clear()</code> (in module <code>dwfpy.bindings</code> ), 105	<code>dwf_digital_in_sample_format_get()</code> (in module <code>dwfpy.bindings</code> ), 108
<code>dwf_digital_i2c_rate_set()</code> (in module <code>dwfpy.bindings</code> ), 106	<code>dwf_digital_in_sample_format_set()</code> (in module <code>dwfpy.bindings</code> ), 109
<code>dwf_digital_i2c_read()</code> (in module <code>dwfpy.bindings</code> ), 106	<code>dwf_digital_in_sample_mode_get()</code> (in module <code>dwfpy.bindings</code> ), 109
<code>dwf_digital_i2c_read_nak_set()</code> (in module <code>dwfpy.bindings</code> ), 106	<code>dwf_digital_in_sample_mode_info()</code> (in module <code>dwfpy.bindings</code> ), 109
<code>dwf_digital_i2c_reset()</code> (in module <code>dwfpy.bindings</code> ), 106	<code>dwf_digital_in_sample_mode_set()</code> (in module <code>dwfpy.bindings</code> ), 109
<code>dwf_digital_i2c_scl_set()</code> (in module <code>dwfpy.bindings</code> ), 106	<code>dwf_digital_in_sample_sensible_get()</code> (in module <code>dwfpy.bindings</code> ), 109
<code>dwf_digital_i2c_sda_set()</code> (in module <code>dwfpy.bindings</code> ), 106	<code>dwf_digital_in_sample_sensible_set()</code> (in module <code>dwfpy.bindings</code> ), 109
<code>dwf_digital_i2c_write()</code> (in module <code>dwfpy.bindings</code> ), 106	<code>dwf_digital_in_status()</code> (in module <code>dwfpy.bindings</code> ), 109
<code>dwf_digital_i2c_write_one()</code> (in module <code>dwfpy.bindings</code> ), 106	<code>dwf_digital_in_status_auto_triggered()</code> (in module <code>dwfpy.bindings</code> ), 109
<code>dwf_digital_i2c_write_read()</code> (in module <code>dwfpy.bindings</code> ), 106	<code>dwf_digital_in_status_data()</code> (in module <code>dwfpy.bindings</code> ), 109
<code>dwf_digital_in_acquisition_mode_get()</code> (in module <code>dwfpy.bindings</code> ), 107	<code>dwf_digital_in_status_data2()</code> (in module <code>dwfpy.bindings</code> ), 110
<code>dwf_digital_in_acquisition_mode_info()</code> (in module <code>dwfpy.bindings</code> ), 107	<code>dwf_digital_in_status_index_write()</code> (in module <code>dwfpy.bindings</code> ), 110
<code>dwf_digital_in_acquisition_mode_set()</code> (in module <code>dwfpy.bindings</code> ), 107	<code>dwf_digital_in_status_noise2()</code> (in module <code>dwfpy.bindings</code> ), 110
<code>dwf_digital_in_bits_info()</code> (in module <code>dwfpy.bindings</code> ), 107	<code>dwf_digital_in_status_record()</code> (in module <code>dwfpy.bindings</code> ), 110
<code>dwf_digital_in_buffer_size_get()</code> (in module <code>dwfpy.bindings</code> ), 107	<code>dwf_digital_in_status_samples_left()</code> (in module <code>dwfpy.bindings</code> ), 110
<code>dwf_digital_in_buffer_size_info()</code> (in module <code>dwfpy.bindings</code> ), 107	<code>dwf_digital_in_status_samples_valid()</code> (in module <code>dwfpy.bindings</code> ), 110

<code>dwf_digital_in_trigger_auto_timeout_get()</code> (in module <code>dwfpy.bindings</code> ), 110	<code>dwf_digital_io_output_enable_info64()</code> (in module <code>dwfpy.bindings</code> ), 113
<code>dwf_digital_in_trigger_auto_timeout_info()</code> (in module <code>dwfpy.bindings</code> ), 110	<code>dwf_digital_io_output_enable_set()</code> (in module <code>dwfpy.bindings</code> ), 113
<code>dwf_digital_in_trigger_auto_timeout_set()</code> (in module <code>dwfpy.bindings</code> ), 110	<code>dwf_digital_io_output_enable_set64()</code> (in module <code>dwfpy.bindings</code> ), 113
<code>dwf_digital_in_trigger_count_set()</code> (in module <code>dwfpy.bindings</code> ), 111	<code>dwf_digital_io_output_get()</code> (in module <code>dwfpy.bindings</code> ), 114
<code>dwf_digital_in_trigger_get()</code> (in module <code>dwfpy.bindings</code> ), 111	<code>dwf_digital_io_output_get64()</code> (in module <code>dwfpy.bindings</code> ), 114
<code>dwf_digital_in_trigger_info()</code> (in module <code>dwfpy.bindings</code> ), 111	<code>dwf_digital_io_output_info()</code> (in module <code>dwfpy.bindings</code> ), 114
<code>dwf_digital_in_trigger_length_set()</code> (in module <code>dwfpy.bindings</code> ), 111	<code>dwf_digital_io_output_info64()</code> (in module <code>dwfpy.bindings</code> ), 114
<code>dwf_digital_in_trigger_match_set()</code> (in module <code>dwfpy.bindings</code> ), 111	<code>dwf_digital_io_output_set()</code> (in module <code>dwfpy.bindings</code> ), 114
<code>dwf_digital_in_trigger_position_get()</code> (in module <code>dwfpy.bindings</code> ), 111	<code>dwf_digital_io_output_set64()</code> (in module <code>dwfpy.bindings</code> ), 114
<code>dwf_digital_in_trigger_position_info()</code> (in module <code>dwfpy.bindings</code> ), 111	<code>dwf_digital_io_reset()</code> (in module <code>dwfpy.bindings</code> ), 114
<code>dwf_digital_in_trigger_position_set()</code> (in module <code>dwfpy.bindings</code> ), 111	<code>dwf_digital_io_status()</code> (in module <code>dwfpy.bindings</code> ), 114
<code>dwf_digital_in_trigger_prefill_get()</code> (in module <code>dwfpy.bindings</code> ), 111	<code>dwf_digital_out_configure()</code> (in module <code>dwfpy.bindings</code> ), 114
<code>dwf_digital_in_trigger_prefill_set()</code> (in module <code>dwfpy.bindings</code> ), 112	<code>dwf_digital_out_count()</code> (in module <code>dwfpy.bindings</code> ), 115
<code>dwf_digital_in_trigger_reset_set()</code> (in module <code>dwfpy.bindings</code> ), 112	<code>dwf_digital_out_counter_get()</code> (in module <code>dwfpy.bindings</code> ), 115
<code>dwf_digital_in_trigger_set()</code> (in module <code>dwfpy.bindings</code> ), 112	<code>dwf_digital_out_counter_info()</code> (in module <code>dwfpy.bindings</code> ), 115
<code>dwf_digital_in_trigger_slope_get()</code> (in module <code>dwfpy.bindings</code> ), 112	<code>dwf_digital_out_counter_init_get()</code> (in module <code>dwfpy.bindings</code> ), 115
<code>dwf_digital_in_trigger_slope_set()</code> (in module <code>dwfpy.bindings</code> ), 112	<code>dwf_digital_out_counter_init_set()</code> (in module <code>dwfpy.bindings</code> ), 115
<code>dwf_digital_in_trigger_source_get()</code> (in module <code>dwfpy.bindings</code> ), 112	<code>dwf_digital_out_counter_set()</code> (in module <code>dwfpy.bindings</code> ), 115
<code>dwf_digital_in_trigger_source_set()</code> (in module <code>dwfpy.bindings</code> ), 112	<code>dwf_digital_out_data_info()</code> (in module <code>dwfpy.bindings</code> ), 115
<code>dwf_digital_io_configure()</code> (in module <code>dwfpy.bindings</code> ), 112	<code>dwf_digital_out_data_set()</code> (in module <code>dwfpy.bindings</code> ), 115
<code>dwf_digital_io_input_info()</code> (in module <code>dwfpy.bindings</code> ), 112	<code>dwf_digital_out_divider_get()</code> (in module <code>dwfpy.bindings</code> ), 115
<code>dwf_digital_io_input_info64()</code> (in module <code>dwfpy.bindings</code> ), 113	<code>dwf_digital_out_divider_info()</code> (in module <code>dwfpy.bindings</code> ), 116
<code>dwf_digital_io_input_status()</code> (in module <code>dwfpy.bindings</code> ), 113	<code>dwf_digital_out_divider_init_get()</code> (in module <code>dwfpy.bindings</code> ), 116
<code>dwf_digital_io_input_status64()</code> (in module <code>dwfpy.bindings</code> ), 113	<code>dwf_digital_out_divider_init_set()</code> (in module <code>dwfpy.bindings</code> ), 116
<code>dwf_digital_io_output_enable_get()</code> (in module <code>dwfpy.bindings</code> ), 113	<code>dwf_digital_out_divider_set()</code> (in module <code>dwfpy.bindings</code> ), 116
<code>dwf_digital_io_output_enable_get64()</code> (in module <code>dwfpy.bindings</code> ), 113	<code>dwf_digital_out_enable_get()</code> (in module <code>dwfpy.bindings</code> ), 116
<code>dwf_digital_io_output_enable_info()</code> (in module <code>dwfpy.bindings</code> ), 113	<code>dwf_digital_out_enable_set()</code> (in module <code>dwfpy.bindings</code> ), 116



<code>dwf_digital_out_idle_get()</code>	(in module <code>dwfpy.bindings</code> ), 116	<code>dwf_digital_out_wait_info()</code>	(in module <code>dwfpy.bindings</code> ), 119
<code>dwf_digital_out_idle_info()</code>	(in module <code>dwfpy.bindings</code> ), 116	<code>dwf_digital_out_wait_set()</code>	(in module <code>dwfpy.bindings</code> ), 119
<code>dwf_digital_out_idle_set()</code>	(in module <code>dwfpy.bindings</code> ), 116	<code>dwf_digital_spi_clock_set()</code>	(in module <code>dwfpy.bindings</code> ), 119
<code>dwf_digital_out_internal_clock_info()</code>	(in module <code>dwfpy.bindings</code> ), 117	<code>dwf_digital_spi_data_set()</code>	(in module <code>dwfpy.bindings</code> ), 120
<code>dwf_digital_out_output_get()</code>	(in module <code>dwfpy.bindings</code> ), 117	<code>dwf_digital_spi_frequency_set()</code>	(in module <code>dwfpy.bindings</code> ), 120
<code>dwf_digital_out_output_info()</code>	(in module <code>dwfpy.bindings</code> ), 117	<code>dwf_digital_spi_mode_set()</code>	(in module <code>dwfpy.bindings</code> ), 120
<code>dwf_digital_out_output_set()</code>	(in module <code>dwfpy.bindings</code> ), 117	<code>dwf_digital_spi_order_set()</code>	(in module <code>dwfpy.bindings</code> ), 120
<code>dwf_digital_out_repeat_get()</code>	(in module <code>dwfpy.bindings</code> ), 117	<code>dwf_digital_spi_read()</code>	(in module <code>dwfpy.bindings</code> ), 120
<code>dwf_digital_out_repeat_info()</code>	(in module <code>dwfpy.bindings</code> ), 117	<code>dwf_digital_spi_read16()</code>	(in module <code>dwfpy.bindings</code> ), 120
<code>dwf_digital_out_repeat_set()</code>	(in module <code>dwfpy.bindings</code> ), 117	<code>dwf_digital_spi_read32()</code>	(in module <code>dwfpy.bindings</code> ), 120
<code>dwf_digital_out_repeat_status()</code>	(in module <code>dwfpy.bindings</code> ), 117	<code>dwf_digital_spi_read_one()</code>	(in module <code>dwfpy.bindings</code> ), 120
<code>dwf_digital_out_repeat_trigger_get()</code>	(in module <code>dwfpy.bindings</code> ), 117	<code>dwf_digital_spi_reset()</code>	(in module <code>dwfpy.bindings</code> ), 120
<code>dwf_digital_out_repeat_trigger_set()</code>	(in module <code>dwfpy.bindings</code> ), 118	<code>dwf_digital_spi_select()</code>	(in module <code>dwfpy.bindings</code> ), 121
<code>dwf_digital_out_reset()</code>	(in module <code>dwfpy.bindings</code> ), 118	<code>dwf_digital_spi_write()</code>	(in module <code>dwfpy.bindings</code> ), 121
<code>dwf_digital_out_run_get()</code>	(in module <code>dwfpy.bindings</code> ), 118	<code>dwf_digital_spi_write16()</code>	(in module <code>dwfpy.bindings</code> ), 121
<code>dwf_digital_out_run_info()</code>	(in module <code>dwfpy.bindings</code> ), 118	<code>dwf_digital_spi_write32()</code>	(in module <code>dwfpy.bindings</code> ), 121
<code>dwf_digital_out_run_set()</code>	(in module <code>dwfpy.bindings</code> ), 118	<code>dwf_digital_spi_write_one()</code>	(in module <code>dwfpy.bindings</code> ), 121
<code>dwf_digital_out_run_status()</code>	(in module <code>dwfpy.bindings</code> ), 118	<code>dwf_digital_spi_write_read()</code>	(in module <code>dwfpy.bindings</code> ), 121
<code>dwf_digital_out_status()</code>	(in module <code>dwfpy.bindings</code> ), 118	<code>dwf_digital_spi_write_read16()</code>	(in module <code>dwfpy.bindings</code> ), 121
<code>dwf_digital_out_trigger_slope_get()</code>	(in module <code>dwfpy.bindings</code> ), 118	<code>dwf_digital_spi_write_read32()</code>	(in module <code>dwfpy.bindings</code> ), 121
<code>dwf_digital_out_trigger_slope_set()</code>	(in module <code>dwfpy.bindings</code> ), 118	<code>dwf_digital_uart_bits_set()</code>	(in module <code>dwfpy.bindings</code> ), 121
<code>dwf_digital_out_trigger_source_get()</code>	(in module <code>dwfpy.bindings</code> ), 119	<code>dwf_digital_uart_parity_set()</code>	(in module <code>dwfpy.bindings</code> ), 122
<code>dwf_digital_out_trigger_source_set()</code>	(in module <code>dwfpy.bindings</code> ), 119	<code>dwf_digital_uart_rate_set()</code>	(in module <code>dwfpy.bindings</code> ), 122
<code>dwf_digital_out_type_get()</code>	(in module <code>dwfpy.bindings</code> ), 119	<code>dwf_digital_uart_reset()</code>	(in module <code>dwfpy.bindings</code> ), 122
<code>dwf_digital_out_type_info()</code>	(in module <code>dwfpy.bindings</code> ), 119	<code>dwf_digital_uart_rx()</code>	(in module <code>dwfpy.bindings</code> ), 122
<code>dwf_digital_out_type_set()</code>	(in module <code>dwfpy.bindings</code> ), 119	<code>dwf_digital_uart_rx_set()</code>	(in module <code>dwfpy.bindings</code> ), 122
<code>dwf_digital_out_wait_get()</code>	(in module <code>dwfpy.bindings</code> ), 119	<code>dwf_digital_uart_stop_set()</code>	(in module <code>dwfpy.bindings</code> ), 122

`dwf_digital_uart_tx()` (in module `dwfpy.bindings`), 122  
`dwf_digital_uart_tx_set()` (in module `dwfpy.bindings`), 122  
`dwf_enum()` (in module `dwfpy.bindings`), 122  
`dwf_enum_config()` (in module `dwfpy.bindings`), 123  
`dwf_enum_config_info()` (in module `dwfpy.bindings`), 123  
`dwf_enum_config_info_str()` (in module `dwfpy.bindings`), 123  
`dwf_enum_device_is_opened()` (in module `dwfpy.bindings`), 123  
`dwf_enum_device_name()` (in module `dwfpy.bindings`), 123  
`dwf_enum_device_type()` (in module `dwfpy.bindings`), 123  
`dwf_enum_sn()` (in module `dwfpy.bindings`), 123  
`dwf_enum_user_name()` (in module `dwfpy.bindings`), 123  
`dwf_get_last_error()` (in module `dwfpy.bindings`), 123  
`dwf_get_last_error_msg()` (in module `dwfpy.bindings`), 124  
`dwf_get_version()` (in module `dwfpy.bindings`), 124  
`dwfpy`  
    module, 33  
`dwfpy.analog_input`  
    module, 34  
`dwfpy.analog_io`  
    module, 52  
`dwfpy.analog_output`  
    module, 56  
`dwfpy.analog_recorder`  
    module, 67  
`dwfpy.application`  
    module, 69  
`dwfpy.bindings`  
    module, 71  
`dwfpy.configuration`  
    module, 124  
`dwfpy.constants`  
    module, 126  
`dwfpy.device`  
    module, 140  
`dwfpy.device_info`  
    module, 176  
`dwfpy.digital_input`  
    module, 178  
`dwfpy.digital_io`  
    module, 193  
`dwfpy.digital_output`  
    module, 197  
`dwfpy.digital_recorder`  
    module, 208  
`dwfpy.exceptions`  
    module, 210  
`dwfpy.helpers`  
    module, 211  
`dwfpy.protocols`  
    module, 212

## E

`ElectronicsExplorer` (class in `dwfpy.device`), 168  
`ElectronicsExplorer.Supplies` (class in `dwfpy.device`), 169  
`ElectronicsExplorer.Supplies.Fixed` (class in `dwfpy.device`), 169  
`ElectronicsExplorer.Supplies.Negative` (class in `dwfpy.device`), 170  
`ElectronicsExplorer.Supplies.Positive` (class in `dwfpy.device`), 170  
`ElectronicsExplorer.Supplies.Reference1` (class in `dwfpy.device`), 171  
`ElectronicsExplorer.Supplies.Reference2` (class in `dwfpy.device`), 171  
`ElectronicsExplorer.Voltmeters` (class in `dwfpy.device`), 172  
`enable_analog_out` (Application property), 69  
`enable_audio_output` (Application property), 70  
`enable_repeat_trigger` (AnalogOutputChannel property), 58  
`enabled` (AnalogDiscovery.Supplies.Negative property), 142  
`enabled` (AnalogDiscovery.Supplies.Positive property), 143  
`enabled` (AnalogDiscovery2.Supplies.Negative property), 148  
`enabled` (AnalogDiscovery2.Supplies.Positive property), 149  
`enabled` (AnalogInputChannel property), 44  
`enabled` (AnalogOutputChannelNode property), 64  
`enabled` (DigitalIoChannel property), 196  
`enabled` (DigitalOutputChannel property), 202  
`enabled` (ElectronicsExplorer.Supplies.Fixed property), 169  
`enabled` (ElectronicsExplorer.Supplies.Negative property), 170  
`enabled` (ElectronicsExplorer.Supplies.Positive property), 171  
`enabled` (ElectronicsExplorer.Supplies.Reference1 property), 171  
`enabled` (ElectronicsExplorer.Supplies.Reference2 property), 171  
`enumerate()` (Device static method), 156  
`Error` (class in `dwfpy.constants`), 134  
`external_frequency` (Application property), 70

## F

falling\_edge (*DigitalInputChannelTrigger* property), 189

filter (*AnalogInputChannel* property), 44

filter (*AnalogInputTrigger* property), 49

filter\_info (*AnalogInputChannel* property), 44

filter\_info (*AnalogInputTrigger* property), 49

FilterMode (class in *dwfpy.constants*), 135

fixed (*ElectronicsExplorer.Supplies* property), 172

force\_trigger() (*AnalogInput* method), 36

frequency (*AnalogInput* property), 36

frequency (*AnalogOutputChannelNode* property), 64

frequency (*Application* property), 70

frequency (*DigitalInputClock* property), 190

frequency (*Protocols.Spi* property), 215

frequency\_max (*AnalogInput* property), 36

frequency\_max (*AnalogOutputChannelNode* property), 64

frequency\_min (*AnalogInput* property), 37

frequency\_min (*AnalogOutputChannelNode* property), 64

function (*AnalogOutputChannelNode* property), 64

Function (class in *dwfpy.constants*), 135

function\_info (*AnalogOutputChannelNode* property), 64

## G

get\_data() (*AnalogInputChannel* method), 44

get\_data() (*DigitalInput* method), 181

get\_device\_name() (*DeviceInfo* static method), 177

get\_last\_error() (*Application* static method), 70

get\_last\_error\_message() (*Application* static method), 70

get\_logger() (*Application* static method), 70

get\_noise() (*AnalogInputChannel* method), 44

get\_noise() (*DigitalInput* method), 181

get\_parameter() (*AnalogDiscovery* method), 144

get\_parameter() (*AnalogDiscovery2* method), 151

get\_parameter() (*Application* static method), 70

get\_parameter() (*Device* method), 156

get\_parameter() (*DeviceBase* method), 160

get\_parameter() (*DigitalDiscovery* method), 166

get\_parameter() (*ElectronicsExplorer* method), 174

get\_properties() (*DeviceInfo* method), 177

get\_sample() (*AnalogInputChannel* method), 44

get\_serial\_number() (*DeviceInfo* static method), 177

get\_trigger() (*AnalogDiscovery* method), 145

get\_trigger() (*AnalogDiscovery2* method), 151

get\_trigger() (*Device* method), 156

get\_trigger() (*DeviceBase* method), 160

get\_trigger() (*DigitalDiscovery* method), 166

get\_trigger() (*ElectronicsExplorer* method), 174

get\_trigger\_mask() (*DigitalInputTrigger* method), 192

get\_trigger\_mask\_info() (*DigitalInputTrigger* method), 192

get\_user\_name() (*DeviceInfo* static method), 177

get\_version() (*Application* static method), 70

GlobalParameter (class in *dwfpy.constants*), 136

## H

handle (*AnalogDiscovery* property), 145

handle (*AnalogDiscovery2* property), 151

handle (*Device* property), 156

handle (*DeviceBase* property), 160

handle (*DigitalDiscovery* property), 166

handle (*ElectronicsExplorer* property), 174

has\_properties (*DeviceInfo* property), 177

Helpers (class in *dwfpy.helpers*), 211

high\_counter (*DigitalOutputChannel* property), 202

high\_level (*DigitalInputChannelTrigger* property), 189

hold\_off (*AnalogInputTrigger* property), 49

hold\_off\_max (*AnalogInputTrigger* property), 49

hold\_off\_min (*AnalogInputTrigger* property), 49

hold\_off\_steps (*AnalogInputTrigger* property), 49

hysteresis (*AnalogInputTrigger* property), 49

hysteresis\_max (*AnalogInputTrigger* property), 49

hysteresis\_min (*AnalogInputTrigger* property), 49

hysteresis\_steps (*AnalogInputTrigger* property), 50

## I

i2c (*Protocols* property), 218

id (*AnalogDiscovery* property), 145

id (*AnalogDiscovery2* property), 151

id (*Device* property), 156

id (*DeviceBase* property), 161

id (*DeviceInfo* property), 177

id (*DigitalDiscovery* property), 166

id (*ElectronicsExplorer* property), 174

idle (*AnalogOutputChannel* property), 58

idle\_info (*AnalogOutputChannel* property), 59

idle\_state (*DigitalOutputChannel* property), 202

idle\_state\_info (*DigitalOutputChannel* property), 202

impedance (*AnalogInputChannel* property), 45

index (*AnalogInputChannel* property), 45

index (*AnalogIoChannel* property), 54

index (*AnalogIoChannelNode* property), 55

index (*AnalogOutputChannel* property), 59

index (*DigitalInputChannel* property), 187

index (*DigitalIoChannel* property), 196

index (*DigitalOutputChannel* property), 202

initial\_counter (*DigitalOutputChannel* property), 202

initial\_divider (*DigitalOutputChannel* property), 202

initial\_state (*DigitalOutputChannel* property), 202

input\_state (*DigitalIo* property), 194

`input_state` (*DigitalIoChannel* property), 196  
`input_state_mask` (*DigitalIo* property), 194  
`inverted` (*Protocols.CAN* property), 213  
`inverted` (*Protocols.Uart* property), 217  
`is_open` (*AnalogDiscovery* property), 145  
`is_open` (*AnalogDiscovery2* property), 151  
`is_open` (*Device* property), 156  
`is_open` (*DeviceBase* property), 161  
`is_open` (*DeviceInfo* property), 178  
`is_open` (*DigitalDiscovery* property), 166  
`is_open` (*ElectronicsExplorer* property), 174

## L

`label` (*AnalogInputChannel* property), 45  
`label` (*AnalogIoChannel* property), 54  
`label` (*AnalogOutputChannel* property), 59  
`label` (*DigitalInputChannel* property), 188  
`label` (*DigitalIoChannel* property), 196  
`label` (*DigitalOutputChannel* property), 202  
`led_brightness` (*Application* property), 70  
`length` (*AnalogInputTrigger* property), 50  
`length_condition` (*AnalogInputTrigger* property), 50  
`length_condition_info` (*AnalogInputTrigger* property), 50  
`length_max` (*AnalogInputTrigger* property), 50  
`length_min` (*AnalogInputTrigger* property), 50  
`length_steps` (*AnalogInputTrigger* property), 50  
`level` (*AnalogInputTrigger* property), 50  
`level_max` (*AnalogInputTrigger* property), 50  
`level_min` (*AnalogInputTrigger* property), 50  
`level_steps` (*AnalogInputTrigger* property), 51  
`limitation` (*AnalogOutputChannel* property), 59  
`limitation_max` (*AnalogOutputChannel* property), 59  
`limitation_min` (*AnalogOutputChannel* property), 59  
`lost_samples` (*AnalogRecorder* property), 68  
`lost_samples` (*DigitalRecorder* property), 209  
`low_counter` (*DigitalOutputChannel* property), 202  
`low_level` (*DigitalInputChannelTrigger* property), 189

## M

`master` (*AnalogOutputChannel* property), 59  
`master_enable` (*AnalogDiscovery.Supplies* property), 143  
`master_enable` (*AnalogDiscovery2.Supplies* property), 149  
`master_enable` (*AnalogIo* property), 53  
`master_enable` (*DigitalDiscovery.Supplies* property), 164  
`master_enable` (*ElectronicsExplorer.Supplies* property), 172  
`master_enable_can_read` (*AnalogIo* property), 53  
`master_enable_can_set` (*AnalogIo* property), 53  
`master_enable_status` (*AnalogDiscovery.Supplies* property), 143

`master_enable_status` (*AnalogDiscovery2.Supplies* property), 149  
`master_enable_status` (*AnalogIo* property), 53  
`master_enable_status` (*DigitalDiscovery.Supplies* property), 164  
`master_enable_status` (*ElectronicsExplorer.Supplies* property), 172  
`max_bits` (*DigitalOutputChannel* property), 203  
`mode` (*AnalogOutputChannel* property), 59  
`mode` (*Protocols.Spi* property), 215  
`module`  
    `dwfpy`, 33  
    `dwfpy.analog_input`, 34  
    `dwfpy.analog_io`, 52  
    `dwfpy.analog_output`, 56  
    `dwfpy.analog_recorder`, 67  
    `dwfpy.application`, 69  
    `dwfpy.bindings`, 71  
    `dwfpy.configuration`, 124  
    `dwfpy.constants`, 126  
    `dwfpy.device`, 140  
    `dwfpy.device_info`, 176  
    `dwfpy.digital_input`, 178  
    `dwfpy.digital_io`, 193  
    `dwfpy.digital_output`, 197  
    `dwfpy.digital_recorder`, 208  
    `dwfpy.exceptions`, 210  
    `dwfpy.helpers`, 211  
    `dwfpy.protocols`, 212  
`module` (*AnalogInputChannel* property), 45  
`module` (*AnalogIoChannel* property), 54  
`module` (*AnalogOutputChannel* property), 59  
`module` (*DigitalInputChannel* property), 188  
`module` (*DigitalIoChannel* property), 196  
`module` (*DigitalOutputChannel* property), 203  
`msb_first` (*Protocols.Spi* property), 215

## N

`name` (*AnalogDiscovery* property), 145  
`name` (*AnalogDiscovery2* property), 151  
`name` (*AnalogIoChannel* property), 54  
`name` (*AnalogIoChannelNode* property), 55  
`name` (*Device* property), 156  
`name` (*DeviceBase* property), 161  
`name` (*DeviceInfo* property), 178  
`name` (*DigitalDiscovery* property), 166  
`name` (*ElectronicsExplorer* property), 175  
`negative` (*AnalogDiscovery.Supplies* property), 143  
`negative` (*AnalogDiscovery2.Supplies* property), 149  
`negative` (*ElectronicsExplorer.Supplies* property), 172  
`node_info` (*AnalogIoChannelNode* property), 55  
`nodes` (*AnalogIoChannel* property), 54  
`nodes` (*AnalogOutputChannel* property), 59  
`noise_buffer_size` (*AnalogInput* property), 37



noise\_buffer\_size\_max (*AnalogInput* property), 37  
 noise\_samples (*DigitalRecorder* property), 209  
 normalize\_serial\_number() (*DeviceInfo* static method), 178

## O

offset (*AnalogInputChannel* property), 45  
 offset (*AnalogOutputChannelNode* property), 65  
 offset\_max (*AnalogInputChannel* property), 45  
 offset\_max (*AnalogOutputChannelNode* property), 65  
 offset\_min (*AnalogInputChannel* property), 45  
 offset\_min (*AnalogOutputChannelNode* property), 65  
 offset\_steps (*AnalogInputChannel* property), 45  
 on\_close\_behavior (*Application* property), 70  
 open() (*AnalogDiscovery* method), 145  
 open() (*AnalogDiscovery2* method), 151  
 open() (*Device* method), 156  
 open() (*DeviceBase* method), 161  
 open() (*DigitalDiscovery* method), 166  
 open() (*ElectronicsExplorer* method), 175  
 output\_enable (*DigitalIo* property), 194  
 output\_enable\_mask (*DigitalIo* property), 194  
 output\_mode (*DigitalOutputChannel* property), 203  
 output\_mode\_info (*DigitalOutputChannel* property), 203  
 output\_state (*DigitalIo* property), 194  
 output\_state (*DigitalIoChannel* property), 196  
 output\_state\_mask (*DigitalIo* property), 195  
 output\_type (*DigitalOutputChannel* property), 203  
 output\_type\_info (*DigitalOutputChannel* property), 203

## P

parity (*Protocols.Uart* property), 217  
 phase (*AnalogOutputChannelNode* property), 65  
 phase\_max (*AnalogOutputChannelNode* property), 65  
 phase\_min (*AnalogOutputChannelNode* property), 65  
 pin\_clock (*Protocols.Spi* property), 215  
 pin\_dq0 (*Protocols.Spi* property), 215  
 pin\_dq1 (*Protocols.Spi* property), 215  
 pin\_dq2 (*Protocols.Spi* property), 216  
 pin\_dq3 (*Protocols.Spi* property), 216  
 pin\_rx (*Protocols.CAN* property), 213  
 pin\_rx (*Protocols.Uart* property), 217  
 pin\_scl (*Protocols.I2C* property), 214  
 pin\_sda (*Protocols.I2C* property), 214  
 pin\_select (*Protocols.Spi* property), 216  
 pin\_tx (*Protocols.CAN* property), 213  
 pin\_tx (*Protocols.Uart* property), 217  
 play\_status (*AnalogOutputChannelNode* property), 65  
 position (*AnalogInputTrigger* property), 51  
 position (*DigitalInputTrigger* property), 192  
 position\_max (*AnalogInputTrigger* property), 51  
 position\_max (*DigitalInputTrigger* property), 192

position\_min (*AnalogInputTrigger* property), 51  
 position\_steps (*AnalogInputTrigger* property), 51  
 positive (*AnalogDiscovery.Supplies* property), 143  
 positive (*AnalogDiscovery2.Supplies* property), 149  
 positive (*ElectronicsExplorer.Supplies* property), 172  
 prefill (*DigitalInputTrigger* property), 192  
 process() (*AnalogRecorder* method), 68  
 process() (*DigitalRecorder* method), 209  
 protocols (*AnalogDiscovery* property), 145  
 protocols (*AnalogDiscovery2* property), 151  
 Protocols (*class in dwfpy.protocols*), 213  
 protocols (*Device* property), 156  
 protocols (*DeviceBase* property), 161  
 protocols (*DigitalDiscovery* property), 166  
 protocols (*ElectronicsExplorer* property), 175  
 Protocols.CAN (*class in dwfpy.protocols*), 213  
 Protocols.I2C (*class in dwfpy.protocols*), 214  
 Protocols.Spi (*class in dwfpy.protocols*), 215  
 Protocols.Uart (*class in dwfpy.protocols*), 217

## R

range (*AnalogInputChannel* property), 45  
 range\_info (*AnalogInputChannel* property), 45  
 range\_max (*AnalogInputChannel* property), 46  
 range\_min (*AnalogInputChannel* property), 46  
 range\_steps (*AnalogInputChannel* property), 46  
 rate (*Protocols.CAN* property), 213  
 rate (*Protocols.I2C* property), 214  
 rate (*Protocols.Uart* property), 218  
 read() (*Protocols.CAN* method), 213  
 read() (*Protocols.I2C* method), 214  
 read() (*Protocols.Spi* method), 216  
 read() (*Protocols.Uart* method), 218  
 read\_nak (*Protocols.I2C* property), 214  
 read\_one() (*Protocols.Spi* method), 216  
 read\_status() (*AnalogInput* method), 37  
 read\_status() (*AnalogIo* method), 53  
 read\_status() (*AnalogOutputChannel* method), 60  
 read\_status() (*DigitalInput* method), 181  
 read\_status() (*DigitalIo* method), 195  
 read\_status() (*DigitalOutput* method), 198  
 record() (*AnalogInput* method), 37  
 record() (*AnalogRecorder* method), 68  
 record() (*DigitalInput* method), 181  
 record() (*DigitalRecorder* method), 209  
 record\_length (*AnalogInput* property), 37  
 record\_status (*AnalogInput* property), 37  
 record\_status (*DigitalInput* property), 182  
 reference1 (*ElectronicsExplorer.Supplies* property), 172  
 reference2 (*ElectronicsExplorer.Supplies* property), 172  
 regulator\_current (*AnalogDiscovery.Supplies* property), 143

regulator\_voltage (*AnalogDiscovery.Supplies* property), 143

remaining\_samples (*AnalogInput* property), 38

remaining\_samples (*DigitalInput* property), 182

repeat (*DigitalOutputTrigger* property), 208

repeat\_count (*AnalogOutputChannel* property), 60

repeat\_count (*DigitalOutput* property), 198

repeat\_count\_max (*AnalogOutputChannel* property), 60

repeat\_count\_max (*DigitalOutput* property), 198

repeat\_count\_min (*AnalogOutputChannel* property), 60

repeat\_count\_min (*DigitalOutput* property), 198

repeat\_count\_status (*AnalogOutputChannel* property), 60

repeat\_count\_status (*DigitalOutput* property), 198

repetition (*DigitalOutputChannel* property), 203

repetition\_max (*DigitalOutputChannel* property), 203

requested\_samples (*AnalogRecorder* property), 68

requested\_samples (*DigitalRecorder* property), 210

reset() (*AnalogDiscovery* method), 145

reset() (*AnalogDiscovery2* method), 152

reset() (*AnalogInput* method), 38

reset() (*AnalogIo* method), 53

reset() (*AnalogOutputChannel* method), 60

reset() (*Device* method), 157

reset() (*DeviceBase* method), 161

reset() (*DigitalDiscovery* method), 166

reset() (*DigitalInput* method), 182

reset() (*DigitalIo* method), 195

reset() (*DigitalOutput* method), 198

reset() (*ElectronicsExplorer* method), 175

reset() (*Protocols.CAN* method), 213

reset() (*Protocols.I2C* method), 214

reset() (*Protocols.Spi* method), 216

reset() (*Protocols.Uart* method), 218

revision (*AnalogDiscovery* property), 145

revision (*AnalogDiscovery2* property), 152

revision (*Device* property), 157

revision (*DeviceBase* property), 161

revision (*DeviceInfo* property), 178

revision (*DigitalDiscovery* property), 167

revision (*ElectronicsExplorer* property), 175

rising\_edge (*DigitalInputChannelTrigger* property), 189

run\_length (*AnalogOutputChannel* property), 60

run\_length (*DigitalOutput* property), 198

run\_length\_max (*AnalogOutputChannel* property), 60

run\_length\_max (*DigitalOutput* property), 199

run\_length\_min (*AnalogOutputChannel* property), 60

run\_length\_min (*DigitalOutput* property), 199

run\_length\_status (*AnalogOutputChannel* property), 60

run\_length\_status (*DigitalOutput* property), 199

## S

sample\_format (*DigitalInput* property), 182

sample\_mode (*DigitalInput* property), 182

sample\_mode\_info (*DigitalInput* property), 182

sample\_rate (*DigitalInput* property), 182

sample\_sensible (*DigitalInput* property), 182

sampling\_delay (*AnalogInputTrigger* property), 51

sampling\_slope (*AnalogInputTrigger* property), 51

sampling\_source (*AnalogInputTrigger* property), 51

scan\_screen() (*AnalogInput* method), 38

scan\_shift() (*AnalogInput* method), 38

select() (*Protocols.Spi* method), 216

serial\_number (*AnalogDiscovery* property), 145

serial\_number (*AnalogDiscovery2* property), 152

serial\_number (*Device* property), 157

serial\_number (*DeviceBase* property), 161

serial\_number (*DeviceInfo* property), 178

serial\_number (*DigitalDiscovery* property), 167

serial\_number (*ElectronicsExplorer* property), 175

set\_counter() (*DigitalInputTrigger* method), 192

set\_counter() (*DigitalOutputChannel* method), 203

set\_custom\_bits() (*DigitalOutputChannel* method), 203

set\_data\_samples() (*AnalogOutputChannelNode* method), 65

set\_error\_handler() (in module *dwfpy.bindings*), 124

set\_idle() (*Protocols.Spi* method), 216

set\_initial\_state\_and\_counter() (*DigitalOutputChannel* method), 204

set\_length() (*DigitalInputTrigger* method), 192

set\_match() (*DigitalInputTrigger* method), 192

set\_parameter() (*AnalogDiscovery* method), 146

set\_parameter() (*AnalogDiscovery2* method), 152

set\_parameter() (*Application* static method), 71

set\_parameter() (*Device* method), 157

set\_parameter() (*DeviceBase* method), 161

set\_parameter() (*DigitalDiscovery* method), 167

set\_parameter() (*ElectronicsExplorer* method), 175

set\_play\_samples() (*AnalogOutputChannelNode* method), 65

set\_reset\_mask() (*DigitalInputTrigger* method), 192

set\_trigger() (*AnalogDiscovery* method), 146

set\_trigger() (*AnalogDiscovery2* method), 152

set\_trigger() (*Device* method), 157

set\_trigger() (*DeviceBase* method), 161

set\_trigger() (*DigitalDiscovery* method), 167

set\_trigger() (*ElectronicsExplorer* method), 175

set\_trigger\_mask() (*DigitalInputTrigger* method), 193

setup() (*AnalogDiscovery.Supplies.Negative* method), 142

setup() (*AnalogDiscovery.Supplies.Positive* method), 143

- `setup()` (*AnalogDiscovery2.Supplies.Negative method*), 148
- `setup()` (*AnalogDiscovery2.Supplies.Positive method*), 149
- `setup()` (*AnalogInputChannel method*), 46
- `setup()` (*AnalogOutputChannel method*), 60
- `setup()` (*DigitalDiscovery.Supplies.Digital method*), 163
- `setup()` (*DigitalIoChannel method*), 196
- `setup()` (*DigitalOutput method*), 199
- `setup()` (*DigitalOutputChannel method*), 204
- `setup()` (*ElectronicsExplorer.Supplies.Fixed method*), 169
- `setup()` (*ElectronicsExplorer.Supplies.Negative method*), 170
- `setup()` (*ElectronicsExplorer.Supplies.Positive method*), 171
- `setup()` (*ElectronicsExplorer.Supplies.Reference1 method*), 171
- `setup()` (*ElectronicsExplorer.Supplies.Reference2 method*), 172
- `setup()` (*Protocols.CAN method*), 214
- `setup()` (*Protocols.I2C method*), 214
- `setup()` (*Protocols.Spi method*), 216
- `setup()` (*Protocols.Uart method*), 218
- `setup_acquisition()` (*AnalogInput method*), 38
- `setup_acquisition()` (*DigitalInput method*), 183
- `setup_am()` (*AnalogOutputChannel method*), 61
- `setup_channel()` (*AnalogInput method*), 39
- `setup_clock()` (*DigitalOutputChannel method*), 204
- `setup_constant()` (*DigitalOutputChannel method*), 205
- `setup_counter_trigger()` (*DigitalInput method*), 183
- `setup_custom()` (*DigitalOutputChannel method*), 205
- `setup_dual()` (*Protocols.Spi method*), 216
- `setup_edge_trigger()` (*AnalogInput method*), 39
- `setup_edge_trigger()` (*DigitalInput method*), 183
- `setup_fm()` (*AnalogOutputChannel method*), 62
- `setup_glitch_trigger()` (*DigitalInput method*), 184
- `setup_length_trigger()` (*DigitalInput method*), 184
- `setup_level_trigger()` (*DigitalInput method*), 184
- `setup_more_trigger()` (*DigitalInput method*), 184
- `setup_pulse()` (*DigitalOutputChannel method*), 206
- `setup_pulse_trigger()` (*AnalogInput method*), 40
- `setup_quad()` (*Protocols.Spi method*), 217
- `setup_random()` (*DigitalOutputChannel method*), 207
- `setup_reset_trigger()` (*DigitalInputChannel method*), 188
- `setup_three_wire()` (*Protocols.Spi method*), 217
- `setup_timeout_trigger()` (*DigitalInput method*), 185
- `setup_transition_trigger()` (*AnalogInput method*), 40
- `setup_trigger()` (*DigitalInput method*), 185
- `setup_trigger()` (*DigitalInputChannel method*), 188
- `setup_trigger()` (*DigitalOutput method*), 199
- `setup_window_trigger()` (*AnalogInput method*), 41
- `single()` (*AnalogInput method*), 42
- `single()` (*DigitalInput method*), 185
- `slope` (*AnalogOutputChannelTrigger property*), 66
- `slope` (*DigitalInputTrigger property*), 193
- `slope` (*DigitalOutputTrigger property*), 208
- `source` (*AnalogInputTrigger property*), 51
- `source` (*AnalogOutputChannelTrigger property*), 66
- `source` (*DigitalInputClock property*), 190
- `source` (*DigitalInputTrigger property*), 193
- `source` (*DigitalOutputTrigger property*), 208
- `source_info` (*DigitalInputClock property*), 190
- `spi` (*Protocols property*), 218
- `status` (*AnalogIoChannelNode property*), 55
- `status` (*AnalogRecorder property*), 68
- `Status` (*class in dwfpy.constants*), 137
- `status` (*DigitalRecorder property*), 210
- `status_max` (*AnalogIoChannelNode property*), 55
- `status_min` (*AnalogIoChannelNode property*), 55
- `status_steps` (*AnalogIoChannelNode property*), 56
- `stop_bits` (*Protocols.Uart property*), 218
- `stream()` (*DigitalInput method*), 186
- `stream()` (*DigitalRecorder method*), 210
- `stretch` (*Protocols.I2C property*), 215
- `supplies` (*AnalogDiscovery property*), 146
- `supplies` (*AnalogDiscovery2 property*), 152
- `supplies` (*DigitalDiscovery property*), 167
- `supplies` (*ElectronicsExplorer property*), 175
- `supports_falling_edge` (*DigitalInputChannelTrigger property*), 189
- `supports_high_level` (*DigitalInputChannelTrigger property*), 189
- `supports_low_level` (*DigitalInputChannelTrigger property*), 189
- `supports_rising_edge` (*DigitalInputChannelTrigger property*), 190
- `symmetry` (*AnalogOutputChannelNode property*), 65
- `symmetry_max` (*AnalogOutputChannelNode property*), 66
- `symmetry_min` (*AnalogOutputChannelNode property*), 66

## T

- `temperature` (*AnalogDiscovery property*), 146
- `temperature` (*AnalogDiscovery2 property*), 152
- `text_info` (*Configuration property*), 125
- `time` (*AnalogInput property*), 42
- `time` (*DigitalInput property*), 186
- `timeout` (*Protocols.I2C property*), 215
- `timeout_max` (*AnalogInput property*), 42
- `total_samples` (*AnalogRecorder property*), 68
- `total_samples` (*DigitalRecorder property*), 210
- `trigger` (*AnalogInput property*), 42



`trigger` (*AnalogOutputChannel* property), 62  
`trigger` (*DigitalInput* property), 186  
`trigger` (*DigitalInputChannel* property), 188  
`trigger` (*DigitalOutput* property), 199  
`trigger_info` (*AnalogDiscovery* property), 146  
`trigger_info` (*AnalogDiscovery2* property), 152  
`trigger_info` (*Device* property), 157  
`trigger_info` (*DeviceBase* property), 161  
`trigger_info` (*DigitalDiscovery* property), 167  
`trigger_info` (*ElectronicsExplorer* property), 175  
`trigger_pc()` (*AnalogDiscovery* method), 146  
`trigger_pc()` (*AnalogDiscovery2* method), 152  
`trigger_pc()` (*Device* method), 157  
`trigger_pc()` (*DeviceBase* method), 162  
`trigger_pc()` (*DigitalDiscovery* method), 167  
`trigger_pc()` (*ElectronicsExplorer* method), 176  
`trigger_slope_info` (*AnalogDiscovery* property), 146  
`trigger_slope_info` (*AnalogDiscovery2* property), 152  
`trigger_slope_info` (*Device* property), 157  
`trigger_slope_info` (*DeviceBase* property), 162  
`trigger_slope_info` (*DigitalDiscovery* property), 167  
`trigger_slope_info` (*ElectronicsExplorer* property), 176  
`TriggerLengthCondition` (class in *dwfpy.constants*), 138  
`TriggerSlope` (class in *dwfpy.constants*), 138  
`TriggerSource` (class in *dwfpy.constants*), 138  
`TriggerType` (class in *dwfpy.constants*), 139  
`type` (*AnalogInputTrigger* property), 51  
`type` (*AnalogOutputChannelNode* property), 66  
`type_info` (*AnalogInputTrigger* property), 52

## U

`uart` (*Protocols* property), 218  
`unit` (*AnalogIoChannelNode* property), 56  
`usb_current` (*AnalogDiscovery* property), 146  
`usb_current` (*AnalogDiscovery2* property), 153  
`usb_current` (*DigitalDiscovery* property), 167  
`usb_limit` (*Application* property), 71  
`usb_power_on_aux` (*Application* property), 71  
`usb_voltage` (*AnalogDiscovery* property), 146  
`usb_voltage` (*AnalogDiscovery2* property), 153  
`usb_voltage` (*DigitalDiscovery* property), 167  
`user_name` (*AnalogDiscovery* property), 146  
`user_name` (*AnalogDiscovery2* property), 153  
`user_name` (*Device* property), 157  
`user_name` (*DeviceBase* property), 162  
`user_name` (*DeviceInfo* property), 178  
`user_name` (*DigitalDiscovery* property), 168  
`user_name` (*ElectronicsExplorer* property), 176

## V

`valid_samples` (*AnalogInput* property), 42

`valid_samples` (*DigitalInput* property), 187  
`value` (*AnalogIoChannelNode* property), 56  
`value_max` (*AnalogIoChannelNode* property), 56  
`value_min` (*AnalogIoChannelNode* property), 56  
`value_steps` (*AnalogIoChannelNode* property), 56  
`vio_current` (*DigitalDiscovery* property), 168  
`vio_voltage` (*DigitalDiscovery* property), 168  
`voltage` (*AnalogDiscovery2.Supplies.Negative* property), 149  
`voltage` (*AnalogDiscovery2.Supplies.Positive* property), 149  
`voltage` (*DigitalDiscovery.Supplies.Digital* property), 164  
`voltage` (*ElectronicsExplorer.Supplies.Fixed* property), 170  
`voltage` (*ElectronicsExplorer.Supplies.Negative* property), 170  
`voltage` (*ElectronicsExplorer.Supplies.Positive* property), 171  
`voltage` (*ElectronicsExplorer.Supplies.Reference1* property), 171  
`voltage` (*ElectronicsExplorer.Supplies.Reference2* property), 172  
`voltmeter1` (*ElectronicsExplorer.Voltmeters* property), 173  
`voltmeter2` (*ElectronicsExplorer.Voltmeters* property), 173  
`voltmeter3` (*ElectronicsExplorer.Voltmeters* property), 173  
`voltmeter4` (*ElectronicsExplorer.Voltmeters* property), 173  
`voltmeters` (*ElectronicsExplorer* property), 176

## W

`wait_for_status()` (*AnalogInput* method), 42  
`wait_for_status()` (*DigitalInput* method), 187  
`wait_length` (*AnalogOutputChannel* property), 62  
`wait_length` (*DigitalOutput* property), 200  
`wait_length_max` (*AnalogOutputChannel* property), 62  
`wait_length_max` (*DigitalOutput* property), 200  
`wait_length_min` (*AnalogOutputChannel* property), 63  
`wait_length_min` (*DigitalOutput* property), 200  
`WaveformsError`, 211  
`Window` (class in *dwfpy.constants*), 140  
`write()` (*Protocols.CAN* method), 214  
`write()` (*Protocols.I2C* method), 215  
`write()` (*Protocols.Spi* method), 217  
`write()` (*Protocols.Uart* method), 218  
`write_index` (*AnalogInput* property), 42  
`write_index` (*DigitalInput* property), 187  
`write_one()` (*Protocols.I2C* method), 215  
`write_one()` (*Protocols.Spi* method), 217  
`write_read()` (*Protocols.I2C* method), 215  
`write_read()` (*Protocols.Spi* method), 217