MSL-Equipment Documentation

Release 0.1.0

Measurement Standards Laboratory

Jul 13, 2017

Contents

Manage and connect to equipment in the laboratory.

CHAPTER 1

Install MSL-Equipment

To install **MSL-Equipment** run:

```
pip install https://github.com/MSLNZ/msl-equipment/archive/master.zip
```

CHAPTER 2

Example Usage

The following illustrates how to use **MSL-Equipment**.

CHAPTER 3

MSL-Equipment API Documentation

The root package is

msl.equipment	Manage and connect to equipment in the laboratory.

which has the following user-facing modules to manage and connect to equipment

msl.equipment.config	Load a XML configuration file.
msl.equipment.database	Load equipment and connection records (rows) from databases.
msl.equipment.factory	Establish a connection to the equipment to send and receive messages.

and the following module that contains the package constants

msl.equipment.constants	MSL-Equipment package constants.

Package Structure

msl.equipment package

Manage and connect to equipment in the laboratory.

msl.equipment.**version_info = version_info(major=0, minor=1, micro=0)**
namedtuple – Contains the version information as a (major, minor, micro) tuple.

Subpackages

msl.equipment.resources package

MSL resources for connecting to equipment.

msl.equipment.resources.**recursive_find_resource_class**(*class_name*)
Find the Python resource class.

Parameters **class_name** (str) – The name of a resource class.

Returns The Python resource class (a subclass of Connection).

Raises IOError – If the Python resource class cannot be found.

msl.equipment.resources.**get_class**(*module_name*, *class_name*)
Returns the specified Python class.

Parameters

- **module_name** (str) – The name of a Python module.
- **class_name** (str) – The name of a Python class.

Returns The Python resource class or None if the class cannot be found.

msl.equipment.resources.**check_manufacture_model_resource_name**(*connection_record*)
Check if there is a resource class with the name equal to *connection_record.model* in the *connection_record.manufacturer.lower()* + '.' + connection_record.model.lower()' module.

For example, if the *connection_record* is for a Thorlabs FW102C filter wheel then check if a msl.equipment.resources.thorlabs.fw102C.FW102C resource class exists.

Parameters **connection_record** (*msl.equipment.record_types.ConnectionRecord*) – A *msl.equipment.record_types.ConnectionRecord* object.

Returns The Python resource class or None if there is no resource class available.

msl.equipment.resources.**find_sdk_class**(*connection_record*)
Find the Python class that is a wrapper around the SDK.

Parameters **connection_record** (*msl.equipment.record_types.ConnectionRecord*) – A *msl.equipment.record_types.ConnectionRecord* object.

Returns *The Python wrapper class around the manufacturer's SDK.*

Raises IOError – If the *msl.equipment.record_types.ConnectionRecord.address* value does not have the required format of SDK::PythonClassName::PathToLibrary.

msl.equipment.resources.**find_serial_class**(*connection_record*)
Find the Python resource class that is used for Serial communication.

Parameters **connection_record** (*msl.equipment.record_types.ConnectionRecord*) – A *msl.equipment.record_types.ConnectionRecord* object.

Returns

- The Python resource class that uses Serial communication or
- None if no resource class was specified in the address.

Subpackages

msl.equipment.resources.bentham package

Submodules

msl.equipment.resources.bentham.benhw32 module

A wrapper around the Bentham SDK benhw32_cdecl.dll.

class msl.equipment.resources.bentham.benhw32.**Bentham32**(*host*, *port*, *quiet*, ***kwargs*)

Bases: msl.loadlib.server32.Server32

A wrapper around the Bentham SDK benhw32_cdecl.dll.

Do not instantiate this class directly. Use *benhw64.Bentham*.

auto_measure()

auto_range()

build_group()

build_system_model(*path*)

close()

close_shutter()

component_select_wl(*p_id*, *wavelength*)

get(*hw_id*, *token*, *index*)

get_c_group(*n*)

get_component_list()

get_group(*group*)

get_hardware_type(*hw_id*)

get_mono_items(*hw_id*)

get_no_of_dark_currents()

get_zero_calibration_info()

group_add(*p_id*, *group*)

group_remove(*p_id*, *group*)

initialise()

load_setup(*path*)

measurement()

multi_auto_range()

multi_get_no_of_dark_currents(*group*)

multi_get_zero_calibration_info(*group*)

multi_initialise()

multi_measurement()

multi_select_wavelength(*wavelength*)

multi_zero_calibration(*start_wavelength*, *stop_wavelength*)

park()

read(*p_message*, *buffer_size*, *p_id*)

report_error()

save_setup(*p_file_name*)

select_wavelength(*wavelength*)

send(*p_message*, *p_id*)

set(*hw_id*, *token*, *index*, *value*)

trace(*on*)

use_group(*group*)

get_version()

zero_calibration(*start_wavelength*, *stop_wavelength*)

camera_get_zero_calibration_info(*p_id*)

camera_measurement(*p_id*, *num*)

camera_zero_calibration(*p_id*, *start_wavelength*, *stop_wavelength*)

delete_group(*n*)

display_advanced_window(*p_id*, *hinstance*)

display_setup_window(*p_id*, *hinstance*)

get_log(*log*)

get_log_size()

get_max_bw(*group*, *start_wavelength*, *stop_wavelength*)

get_min_step(*group*, *start_wavelength*, *stop_wavelength*)

get_n_groups()

get_str(*hw_id*, *token*, *index*, *s*)

mapped_logging(*i*)

multi_auto_measure()

multi_park()

start_log(*c_list*)

stop_log(*c_list*)

msl.equipment.resources.bentham.benhw64 module

A wrapper around the Bentham SDK benhw32_cdecl.dll.

class msl.equipment.resources.bentham.benhw64.**Bentham**(*record*)

Bases: *msl.equipment.connection.Connection*, msl.loadlib.client64.Client64

A wrapper around the *benhw32.Bentham32* class.

This class can be used with either a 32-bit or 64-bit Python interpreter to call the 32-bit functions in benhw32_cdecl.dll.

The *record.connection.properties* dictionary for a Bentham device supports the following key-value pairs:

```
'model': 'C:\path\to\System.cfg',  # default is ''
'setup': 'C:\path\to\System.atr',  # default is ''
```

If both properties are not defined in the **Connections** *Database* then you will have to call *build_system_model()*, *load_setup()* and *initialise()* to configure the SDK.

Do not instantiate this class directly. Use the factory method, *msl.equipment.factory.connect*, or the *record* object itself, *record.connect()*, to connect to the equipment.

Parameters **record** (*EquipmentRecord*) – An equipment record from an **Equipment-Register** *Database*.

auto_measure()

build_system_model(*path*)

disconnect()

errcheck(*result*, **args*, *append_msg=''*)

get(*hw_id*, *token*, *index*)

get_component_list()

get_hardware_type(*hw_id*)

get_mono_items(*hw_id*)

wavelength

initialise()

load_setup(*path*)

park()

select_wavelength(*wavelength*)

set(*hw_id*, *token*, *index*, *value*)

version()

zero_calibration(*start_wavelength*, *stop_wavelength*)

msl.equipment.resources.bentham.errors module

Error code definition file for Bentham Instruments Spectroradiometer Control DLL

msl.equipment.resources.bentham.tokens module

Attribute token definition file for Bentham Instruments Spectroradiometer Control DLL.

msl.equipment.resources.picotech package

Resources for equipment from Pico Technology.

Subpackages

msl.equipment.resources.picotech.picoscope package

Wrapper around the PicoScope SDK from Pico Technologies.

This package was initially created using Pico Technology SDK 64-bit v10.6.10.24

The latest SDK can be downloaded from here.

Submodules

msl.equipment.resources.picotech.picoscope.callbacks module

Callback functions in the Pico Technology SDK v10.6.10.24

msl.equipment.resources.picotech.picoscope.channel module

Contains the information about a PicoScope channel.

class msl.equipment.resources.picotech.picoscope.channel.**PicoScopeChannel**(*channel*, *enabled*, *coupling*, *voltage_range*, *voltage_offset*, *bandwidth*, *max_adu_va*

Bases: object

Contains the information about a PicoScope channel.

This class is used by *picoscope.PicoScope* and is not meant to be called directly.

Parameters

- **channel** (enum.IntEnum) – The channel number.
- **enabled** (bool) – Whether the channel is enabled.
- **coupling** (enum.IntEnum) – The coupling, e.g. AC or DC.
- **voltage_range** (float) – The voltage range, in Volts.
- **voltage_offset** (float) – The voltage offset, in Volts.
- **bandwidth** (enum.IntEnum or None) – The bandwidth used, if the PicoScope supports a BandwidthLimiter.
- **max_adu_value** (int) – The maximum analog-to-digital unit.

channel
enum.IntEnum – The channel number.

enabled
bool – Whether the channel is enabled.

coupling
enum.IntEnum – The coupling, e.g. AC or DC.

voltage_range
float – The voltage range, in Volts.

voltage_offset
float – The voltage offset, in Volts.

bandwidth
enum.IntEnum or None – The bandwidth used, if the PicoScope supports a BandwidthLimiter.

volts_per_adu
float – The connversion factor to convert ADU to volts

raw
numpy.ndarray – The raw data, in ADU

buffer
numpy.ndarray – The raw data, in ADU

volts
numpy.ndarray – The data, in volts

num_samples
int – The size of the data array.

allocate(*num_captures*, *num_samples*)
Allocate memory to save the data.

Parameters

- **num_captures** (int) – The number of captures
- **num_samples** (int) – The number of samples

msl.equipment.resources.picotech.picoscope.enums module

Enums defined in the Pico Technology SDK v10.6.10.24

class msl.equipment.resources.picotech.picoscope.enums.**PicoScopeInfoApi**
Bases: enum.IntEnum

Constants that can be passed to the `get_unit_info()` method.

DRIVER_VERSION = 0

USB_VERSION = 1

HARDWARE_VERSION = 2

VARIANT_INFO = 3

BATCH_AND_SERIAL = 4

CAL_DATE = 5

KERNEL_VERSION = 6

DIGITAL_HARDWARE_VERSION = 7

ANALOGUE_HARDWARE_VERSION = 8

FIRMWARE_VERSION_1 = 9

FIRMWARE_VERSION_2 = 10

class msl.equipment.resources.picotech.picoscope.enums.**PS2000Channel**
Bases: enum.IntEnum

An enumeration.

A = 0

B = 1

C = 2

D = 3

EXT = 4

MAX_CHANNELS = 4

NONE = 5

class msl.equipment.resources.picotech.picoscope.enums.**PS2000Range**
Bases: enum.IntEnum

An enumeration.

R_10MV = 0

R_20MV = 1

R_50MV = 2

R_100MV = 3

R_200MV = 4

R_500MV = 5

R_1V = 6

R_2V = 7

R_5V = 8

R_10V = 9

R_20V = 10

R_50V = 11

R_MAX = 12

class msl.equipment.resources.picotech.picoscope.enums.**PS2000TimeUnits**
Bases: enum.IntEnum

An enumeration.

FS = 0

PS = 1

NS = 2

US = 3

MS = 4

S = 5

MAX = 6

class msl.equipment.resources.picotech.picoscope.enums.**PS2000Error**
Bases: enum.IntEnum

An enumeration.

OK = 0

MAX_UNITS_OPENED = 1

MEM_FAIL = 2

NOT_FOUND = 3

FW_FAIL = 4

NOT_RESPONDING = 5

CONFIG_FAIL = 6

OS_NOT_SUPPORTED = 7

PICOPP_TOO_OLD = 8

class msl.equipment.resources.picotech.picoscope.enums.**PS2000Info**
Bases: enum.IntEnum

An enumeration.

DRIVER_VERSION = 0

USB_VERSION = 1

HARDWARE_VERSION = 2

VARIANT_INFO = 3

BATCH_AND_SERIAL = 4

CAL_DATE = 5

ERROR_CODE = 6

KERNEL_DRIVER_VERSION = 7

class msl.equipment.resources.picotech.picoscope.enums.**PS2000TriggerDirection**
Bases: enum.IntEnum

An enumeration.

RISING = 0

FALLING = 1

MAX = 2

class msl.equipment.resources.picotech.picoscope.enums.**PS2000OpenProgress**
Bases: enum.IntEnum

An enumeration.

FAIL = -1

PENDING = 0

COMPLETE = 1

class msl.equipment.resources.picotech.picoscope.enums.**PS2000EtsMode**
Bases: enum.IntEnum

An enumeration.

OFF = 0

FAST = 1

SLOW = 2

MAX = 3

class msl.equipment.resources.picotech.picoscope.enums.**PS2000ButtonState**
Bases: enum.IntEnum

An enumeration.

NO_PRESS = 0

SHORT_PRESS = 1

LONG_PRESS = 2

class msl.equipment.resources.picotech.picoscope.enums.**PS2000SweepType**
Bases: enum.IntEnum

An enumeration.

UP = 0

DOWN = 1

UPDOWN = 2

DOWNUP = 3

MAX = 4

class msl.equipment.resources.picotech.picoscope.enums.**PS2000WaveType**
Bases: enum.IntEnum

An enumeration.

SINE = 0

SQUARE = 1

TRIANGLE = 2

RAMPUP = 3

RAMPDOWN = 4

DC_VOLTAGE = 5

GAUSSIAN = 6

SINC = 7

HALF_SINE = 8

class msl.equipment.resources.picotech.picoscope.enums.**PS2000ThresholdDirection**
Bases: enum.IntEnum

An enumeration.

ABOVE = 0

BELOW = 1

ADV_RISING = 2

ADV_FALLING = 3

RISING_OR_FALLING = 4

INSIDE = 0

OUTSIDE = 1

ENTER = 2

EXIT = 3

ENTER_OR_EXIT = 4

ADV_NONE = 2

class msl.equipment.resources.picotech.picoscope.enums.**PS2000ThresholdMode**
Bases: enum.IntEnum

An enumeration.

LEVEL = 0

WINDOW = 1

class msl.equipment.resources.picotech.picoscope.enums.**PS2000TriggerState**
Bases: enum.IntEnum

An enumeration.

DONT_CARE = 0

TRUE = 1

FALSE = 2

MAX = 3

class msl.equipment.resources.picotech.picoscope.enums.**PS2000PulseWidthType**
Bases: enum.IntEnum

An enumeration.

NONE = 0

LESS_THAN = 1

GREATER_THAN = 2

IN_RANGE = 3

OUT_OF_RANGE = 4

class msl.equipment.resources.picotech.picoscope.enums.**PS2000AChannelBufferIndex**
Bases: enum.IntEnum

An enumeration.

A_MAX = 0

A_MIN = 1

B_MAX = 2

B_MIN = 3

C_MAX = 4

C_MIN = 5

D_MAX = 6

D_MIN = 7

MAX = 8

class msl.equipment.resources.picotech.picoscope.enums.**PS2000AChannel**
Bases: enum.IntEnum

An enumeration.

A = 0

B = 1

C = 2

D = 3

EXT = 4

MAX_CHANNELS = 4

AUX = 5

MAX_TRIGGER_SOURCES = 6

class msl.equipment.resources.picotech.picoscope.enums.**PS2000ATriggerOperand**
Bases: enum.IntEnum

An enumeration.

NONE = 0

OR = 1

AND = 2

THEN = 3

class msl.equipment.resources.picotech.picoscope.enums.**PS2000DigitalPort**
Bases: enum.IntEnum

An enumeration.

PORT0 = 128

PORT1 = 129

PORT2 = 130

PORT3 = 131

MAX = 4

class msl.equipment.resources.picotech.picoscope.enums.**PS2000ADigitalChannel**
Bases: enum.IntEnum

An enumeration.

CHANNEL_0 = 0

CHANNEL_1 = 1

CHANNEL_2 = 2

CHANNEL_3 = 3

CHANNEL_4 = 4

CHANNEL_5 = 5

CHANNEL_6 = 6

CHANNEL_7 = 7

CHANNEL_8 = 8

CHANNEL_9 = 9

CHANNEL_10 = 10

CHANNEL_11 = 11

CHANNEL_12 = 12

CHANNEL_13 = 13

CHANNEL_14 = 14

CHANNEL_15 = 15

CHANNEL_16 = 16

CHANNEL_17 = 17

CHANNEL_18 = 18

CHANNEL_19 = 19

CHANNEL_20 = 20

CHANNEL_21 = 21

CHANNEL_22 = 22

CHANNEL_23 = 23

CHANNEL_24 = 24

CHANNEL_25 = 25

CHANNEL_26 = 26

CHANNEL_27 = 27

CHANNEL_28 = 28

CHANNEL_29 = 29

CHANNEL_30 = 30

CHANNEL_31 = 31

CHANNEL_MAX = 32

class msl.equipment.resources.picotech.picoscope.enums.**PS2000ARange**
Bases: enum.IntEnum

An enumeration.

R_10MV = 0

R_20MV = 1

R_50MV = 2

R_100MV = 3

R_200MV = 4

R_500MV = 5

R_1V = 6

R_2V = 7

R_5V = 8

R_10V = 9

R_20V = 10

R_50V = 11

R_MAX = 12

class msl.equipment.resources.picotech.picoscope.enums.**PS2000ACoupling**
Bases: enum.IntEnum

An enumeration.

AC = 0

DC = 1

class msl.equipment.resources.picotech.picoscope.enums.**PS2000AChannelInfo**
Bases: enum.IntEnum

An enumeration.

RANGES = 0

class msl.equipment.resources.picotech.picoscope.enums.**PS2000AEtsMode**
Bases: enum.IntEnum

An enumeration.

OFF = 0

FAST = 1

SLOW = 2

MAX = 3

class msl.equipment.resources.picotech.picoscope.enums.**PS2000ATimeUnits**
Bases: enum.IntEnum

An enumeration.

FS = 0

PS = 1

NS = 2

US = 3

MS = 4

S = 5

MAX = 6

class msl.equipment.resources.picotech.picoscope.enums.**PS2000ASweepType**
Bases: enum.IntEnum

An enumeration.

UP = 0

DOWN = 1

UPDOWN = 2

DOWNUP = 3

MAX = 4

class msl.equipment.resources.picotech.picoscope.enums.**PS2000AWaveType**
Bases: enum.IntEnum

An enumeration.

SINE = 0

SQUARE = 1

TRIANGLE = 2

RAMP_UP = 3

RAMP_DOWN = 4

SINC = 5

GAUSSIAN = 6

HALF_SINE = 7

DC_VOLTAGE = 8

MAX = 9

class msl.equipment.resources.picotech.picoscope.enums.**PS2000AExtraOperations**
Bases: enum.IntEnum

An enumeration.

OFF = 0

WHITENOISE = 1

PRBS = 2

class msl.equipment.resources.picotech.picoscope.enums.**PS2000ASigGenTrigType**
Bases: enum.IntEnum

An enumeration.

RISING = 0

FALLING = 1

GATE_HIGH = 2

GATE_LOW = 3

class msl.equipment.resources.picotech.picoscope.enums.**PS2000ASigGenTrigSource**
Bases: enum.IntEnum

An enumeration.

NONE = 0

SCOPE_TRIG = 1

AUX_IN = 2

EXT_IN = 3

SOFT_TRIG = 4

class msl.equipment.resources.picotech.picoscope.enums.**PS2000AIndexMode**
Bases: enum.IntEnum

An enumeration.

SINGLE = 0

DUAL = 1

QUAD = 2

MAX = 3

class msl.equipment.resources.picotech.picoscope.enums.**PS2000A_ThresholdMode**
Bases: enum.IntEnum

An enumeration.

LEVEL = 0

WINDOW = 1

class msl.equipment.resources.picotech.picoscope.enums.**PS2000AThresholdDirection**
Bases: enum.IntEnum

An enumeration.

ABOVE = 0

BELOW = 1

RISING = 2

FALLING = 3

RISING_OR_FALLING = 4

ABOVE_LOWER = 5

BELOW_LOWER = 6

RISING_LOWER = 7

FALLING_LOWER = 8

INSIDE = 0

OUTSIDE = 1

ENTER = 2

EXIT = 3

ENTER_OR_EXIT = 4

POSITIVE_RUNT = 9

NEGATIVE_RUNT = 10

NONE = 2

class msl.equipment.resources.picotech.picoscope.enums.**PS2000ADigitalDirection**
Bases: enum.IntEnum

An enumeration.

DONT_CARE = 0

LOW = 1

HIGH = 2

RISING = 3

FALLING = 4

RISING_OR_FALLING = 5

MAX = 6

class msl.equipment.resources.picotech.picoscope.enums.**PS2000ATriggerState**
Bases: enum.IntEnum

An enumeration.

DONT_CARE = 0

TRUE = 1

FALSE = 2

MAX = 3

class msl.equipment.resources.picotech.picoscope.enums.**PS2000ARatioMode**
Bases: enum.IntEnum

An enumeration.

NONE = 0

AGGREGATE = 1

DECIMATE = 2

AVERAGE = 4

class msl.equipment.resources.picotech.picoscope.enums.**PS2000APulseWidthType**
Bases: enum.IntEnum

An enumeration.

NONE = 0

LESS_THAN = 1

GREATER_THAN = 2

IN_RANGE = 3

OUT_OF_RANGE = 4

class msl.equipment.resources.picotech.picoscope.enums.**PS2000AHoldOffType**
Bases: enum.IntEnum

An enumeration.

TIME = 0

MAX = 1

class msl.equipment.resources.picotech.picoscope.enums.**PS3000Channel**
Bases: enum.IntEnum

An enumeration.

A = 0

B = 1

C = 2

D = 3

EXT = 4

MAX_CHANNELS = 4

NONE = 5

MAX_TRIGGER_SOURCES = 6

class msl.equipment.resources.picotech.picoscope.enums.**PS3000Range**
Bases: enum.IntEnum

An enumeration.

R_10MV = 0

R_20MV = 1

R_50MV = 2

R_100MV = 3

R_200MV = 4

R_500MV = 5

R_1V = 6

R_2V = 7

R_5V = 8

R_10V = 9

R_20V = 10

R_50V = 11

R_100V = 12

R_200V = 13

R_400V = 14

R_MAX = 15

class msl.equipment.resources.picotech.picoscope.enums.**PS3000WaveTypes**
Bases: enum.IntEnum

An enumeration.

SQUARE = 0

TRIANGLE = 1

SINE = 2

MAX = 3

class msl.equipment.resources.picotech.picoscope.enums.**PS3000TimeUnits**
Bases: enum.IntEnum

An enumeration.

FS = 0

PS = 1

NS = 2

US = 3

MS = 4

S = 5

MAX = 6

class msl.equipment.resources.picotech.picoscope.enums.**PS3000Error**
Bases: enum.IntEnum

An enumeration.

OK = 0

MAX_UNITS_OPENED = 1

MEM_FAIL = 2

NOT_FOUND = 3

FW_FAIL = 4

NOT_RESPONDING = 5

CONFIG_FAIL = 6

OS_NOT_SUPPORTED = 7

PICOPP_TOO_OLD = 8

class msl.equipment.resources.picotech.picoscope.enums.**PS3000Info**
Bases: enum.IntEnum

An enumeration.

DRIVER_VERSION = 0

USB_VERSION = 1

HARDWARE_VERSION = 2

VARIANT_INFO = 3

BATCH_AND_SERIAL = 4

CAL_DATE = 5

ERROR_CODE = 6

KERNEL_DRIVER_VERSION = 7

class msl.equipment.resources.picotech.picoscope.enums.**PS3000TriggerDirection**
Bases: enum.IntEnum

An enumeration.

RISING = 0

FALLING = 1

MAX = 2

class msl.equipment.resources.picotech.picoscope.enums.**PS3000OpenProgress**
Bases: enum.IntEnum

An enumeration.

FAIL = -1

PENDING = 0

COMPLETE = 1

class msl.equipment.resources.picotech.picoscope.enums.**PS3000EtsMode**
Bases: enum.IntEnum

An enumeration.

OFF = 0

FAST = 1

SLOW = 2

MAX = 3

class msl.equipment.resources.picotech.picoscope.enums.**PS3000ThresholdDirection**
Bases: enum.IntEnum

An enumeration.

ABOVE = 0

BELOW = 1

RISING = 2

FALLING = 3

RISING_OR_FALLING = 4

INSIDE = 0

OUTSIDE = 1

ENTER = 2

EXIT = 3

ENTER_OR_EXIT = 4

NONE = 2

class msl.equipment.resources.picotech.picoscope.enums.**PS3000ThresholdMode**
Bases: enum.IntEnum

An enumeration.

LEVEL = 0

WINDOW = 1

class msl.equipment.resources.picotech.picoscope.enums.**PS3000TriggerState**
Bases: enum.IntEnum

An enumeration.

DONT_CARE = 0

TRUE = 1

FALSE = 2

MAX = 3

class msl.equipment.resources.picotech.picoscope.enums.**PS3000PulseWidthType**
Bases: enum.IntEnum

An enumeration.

NONE = 0

LESS_THAN = 1

GREATER_THAN = 2

IN_RANGE = 3

OUT_OF_RANGE = 4

class msl.equipment.resources.picotech.picoscope.enums.**PS3000ABandwidthLimiter**
Bases: enum.IntEnum

An enumeration.

BW_FULL = 0

BW_20MHZ = 1

class msl.equipment.resources.picotech.picoscope.enums.**PS3000AChannelBufferIndex**
Bases: enum.IntEnum

An enumeration.

A_MAX = 0

A_MIN = 1

B_MAX = 2

B_MIN = 3

C_MAX = 4

C_MIN = 5

D_MAX = 6

D_MIN = 7

MAX = 8

class msl.equipment.resources.picotech.picoscope.enums.**PS3000AChannel**
Bases: enum.IntEnum

An enumeration.

A = 0

B = 1

C = 2

D = 3

EXT = 4

MAX_CHANNELS = 4

AUX = 5

MAX_TRIGGER_SOURCES = 6

class msl.equipment.resources.picotech.picoscope.enums.**PS3000ADigitalPort**
Bases: enum.IntEnum

An enumeration.

PORT0 = 128

PORT1 = 129

PORT2 = 130

PORT3 = 131

MAX = 4

class msl.equipment.resources.picotech.picoscope.enums.**PS3000ADigitalChannel**
Bases: enum.IntEnum

An enumeration.

CHANNEL_0 = 0

CHANNEL_1 = 1

CHANNEL_2 = 2

CHANNEL_3 = 3

CHANNEL_4 = 4

CHANNEL_5 = 5

CHANNEL_6 = 6

CHANNEL_7 = 7

CHANNEL_8 = 8

CHANNEL_9 = 9

CHANNEL_10 = 10

CHANNEL_11 = 11

CHANNEL_12 = 12

CHANNEL_13 = 13

CHANNEL_14 = 14

CHANNEL_15 = 15

CHANNEL_16 = 16

CHANNEL_17 = 17

CHANNEL_18 = 18

CHANNEL_19 = 19

CHANNEL_20 = 20

CHANNEL_21 = 21

CHANNEL_22 = 22

CHANNEL_23 = 23

CHANNEL_24 = 24

CHANNEL_25 = 25

CHANNEL_26 = 26

CHANNEL_27 = 27

CHANNEL_28 = 28

CHANNEL_29 = 29

CHANNEL_30 = 30

CHANNEL_31 = 31

CHANNEL_MAX = 32

class msl.equipment.resources.picotech.picoscope.enums.**PS3000ARange**
Bases: enum.IntEnum

An enumeration.

R_10MV = 0

R_20MV = 1

R_50MV = 2

R_100MV = 3

R_200MV = 4

R_500MV = 5

R_1V = 6

R_2V = 7

R_5V = 8

R_10V = 9

R_20V = 10

R_50V = 11

R_MAX = 12

class msl.equipment.resources.picotech.picoscope.enums.**PS3000ACoupling**
Bases: enum.IntEnum

An enumeration.

AC = 0

DC = 1

class msl.equipment.resources.picotech.picoscope.enums.**PS3000AChannelInfo**
Bases: enum.IntEnum

An enumeration.

RANGES = 0

class msl.equipment.resources.picotech.picoscope.enums.**PS3000AEtsMode**
Bases: enum.IntEnum

An enumeration.

OFF = 0

FAST = 1

SLOW = 2

MAX = 3

class msl.equipment.resources.picotech.picoscope.enums.**PS3000ATimeUnits**
Bases: enum.IntEnum

An enumeration.

FS = 0

PS = 1

NS = 2

US = 3

MS = 4

S = 5

MAX = 6

class msl.equipment.resources.picotech.picoscope.enums.**PS3000ASweepType**
Bases: enum.IntEnum

An enumeration.

UP = 0

DOWN = 1

UPDOWN = 2

DOWNUP = 3

MAX = 4

class msl.equipment.resources.picotech.picoscope.enums.**PS3000AWaveType**
Bases: enum.IntEnum

An enumeration.

SINE = 0

SQUARE = 1

TRIANGLE = 2

RAMP_UP = 3

RAMP_DOWN = 4

SINC = 5

GAUSSIAN = 6

HALF_SINE = 7

DC_VOLTAGE = 8

MAX = 9

class msl.equipment.resources.picotech.picoscope.enums.**PS3000AExtraOperations**
Bases: enum.IntEnum

An enumeration.

OFF = 0

WHITENOISE = 1

PRBS = 2

class msl.equipment.resources.picotech.picoscope.enums.**PS3000ASigGenTrigType**
Bases: enum.IntEnum

An enumeration.

RISING = 0

FALLING = 1

GATE_HIGH = 2

GATE_LOW = 3

class msl.equipment.resources.picotech.picoscope.enums.**PS3000ASigGenTrigSource**
Bases: enum.IntEnum

An enumeration.

NONE = 0

SCOPE_TRIG = 1

AUX_IN = 2

EXT_IN = 3

SOFT_TRIG = 4

class msl.equipment.resources.picotech.picoscope.enums.**PS3000AIndexMode**
Bases: enum.IntEnum

An enumeration.

SINGLE = 0

DUAL = 1

QUAD = 2

MAX = 3

class msl.equipment.resources.picotech.picoscope.enums.**PS3000A_ThresholdMode**
Bases: enum.IntEnum

An enumeration.

LEVEL = 0

WINDOW = 1

class msl.equipment.resources.picotech.picoscope.enums.**PS3000AThresholdDirection**
Bases: enum.IntEnum

An enumeration.

ABOVE = 0

BELOW = 1

RISING = 2

FALLING = 3

RISING_OR_FALLING = 4

ABOVE_LOWER = 5

BELOW_LOWER = 6

RISING_LOWER = 7

FALLING_LOWER = 8

INSIDE = 0

OUTSIDE = 1

ENTER = 2

EXIT = 3

ENTER_OR_EXIT = 4

POSITIVE_RUNT = 9

NEGATIVE_RUNT = 10

NONE = 2

class msl.equipment.resources.picotech.picoscope.enums.**PS3000ADigitalDirection**
Bases: enum.IntEnum

An enumeration.

DONT_CARE = 0

LOW = 1

HIGH = 2

RISING = 3

FALLING = 4

RISING_OR_FALLING = 5

MAX = 6

class msl.equipment.resources.picotech.picoscope.enums.**PS3000ATriggerState**
Bases: enum.IntEnum

An enumeration.

DONT_CARE = 0

TRUE = 1

FALSE = 2

MAX = 3

class msl.equipment.resources.picotech.picoscope.enums.**PS3000ARatioMode**
Bases: enum.IntEnum

An enumeration.

NONE = 0

AGGREGATE = 1

DECIMATE = 2

AVERAGE = 4

class msl.equipment.resources.picotech.picoscope.enums.**PS3000APulseWidthType**
Bases: enum.IntEnum

An enumeration.

NONE = 0

LESS_THAN = 1

GREATER_THAN = 2

IN_RANGE = 3

OUT_OF_RANGE = 4

class msl.equipment.resources.picotech.picoscope.enums.**PS3000AHoldOffType**
Bases: enum.IntEnum

An enumeration.

TIME = 0

EVENT = 1

MAX = 2

class msl.equipment.resources.picotech.picoscope.enums.**PS4000ChannelBufferIndex**
Bases: enum.IntEnum

An enumeration.

A_MAX = 0

A_MIN = 1

B_MAX = 2

B_MIN = 3

C_MAX = 4

C_MIN = 5

D_MAX = 6

D_MIN = 7

MAX = 8

class msl.equipment.resources.picotech.picoscope.enums.**PS4000Channel**
Bases: enum.IntEnum

An enumeration.

A = 0

B = 1

C = 2

D = 3

EXT = 4

MAX_CHANNELS = 4

AUX = 5

MAX_TRIGGER_SOURCES = 6

class msl.equipment.resources.picotech.picoscope.enums.**PS4000Range**
Bases: enum.IntEnum

An enumeration.

R_10MV = 0

R_20MV = 1

R_50MV = 2

R_100MV = 3

R_200MV = 4

R_500MV = 5

R_1V = 6

R_2V = 7

R_5V = 8

R_10V = 9

R_20V = 10

R_50V = 11

R_100V = 12

MAX_RANGES = 13

RESISTANCE_100R = 13

RESISTANCE_1K = 14

RESISTANCE_10K = 15

RESISTANCE_100K = 16

RESISTANCE_1M = 17

MAX_RESISTANCES = 18

ACCELEROMETER_10MV = 18

ACCELEROMETER_20MV = 19

ACCELEROMETER_50MV = 20

ACCELEROMETER_100MV = 21

ACCELEROMETER_200MV = 22

ACCELEROMETER_500MV = 23

ACCELEROMETER_1V = 24

ACCELEROMETER_2V = 25

ACCELEROMETER_5V = 26

ACCELEROMETER_10V = 27

ACCELEROMETER_20V = 28

ACCELEROMETER_50V = 29

ACCELEROMETER_100V = 30

MAX_ACCELEROMETER = 31

TEMPERATURE_UPTO_40 = 31

TEMPERATURE_UPTO_70 = 32

TEMPERATURE_UPTO_100 = 33

TEMPERATURE_UPTO_130 = 34

MAX_TEMPERATURES = 35

RESISTANCE_5K = 35

RESISTANCE_25K = 36

RESISTANCE_50K = 37

MAX_EXTRA_RESISTANCES = 38

class msl.equipment.resources.picotech.picoscope.enums.**PS4000Probe**

Bases: enum.IntEnum

An enumeration.

NONE = 0

CURRENT_CLAMP_10A = 1

CURRENT_CLAMP_1000A = 2

TEMPERATURE_SENSOR = 3

CURRENT_MEASURING_DEVICE = 4

PRESSURE_SENSOR_50BAR = 5

PRESSURE_SENSOR_5BAR = 6

OPTICAL_SWITCH = 7

UNKNOWN = 8

MAX_PROBES = 8

class msl.equipment.resources.picotech.picoscope.enums.**PS4000ChannelInfo**
Bases: enum.IntEnum

An enumeration.

RANGES = 0

RESISTANCES = 1

ACCELEROMETER = 2

PROBES = 3

TEMPERATURES = 4

class msl.equipment.resources.picotech.picoscope.enums.**PS4000EtsMode**
Bases: enum.IntEnum

An enumeration.

OFF = 0

FAST = 1

SLOW = 2

MAX = 3

class msl.equipment.resources.picotech.picoscope.enums.**PS4000TimeUnits**
Bases: enum.IntEnum

An enumeration.

FS = 0

PS = 1

NS = 2

US = 3

MS = 4

S = 5

MAX = 6

class msl.equipment.resources.picotech.picoscope.enums.**PS4000SweepType**
Bases: enum.IntEnum

An enumeration.

UP = 0

DOWN = 1

UPDOWN = 2

DOWNUP = 3

MAX = 4

class msl.equipment.resources.picotech.picoscope.enums.**PS4000OperationTypes**
Bases: enum.IntEnum

An enumeration.

NONE = 0

WHITENOISE = 1

PRBS = 2

class msl.equipment.resources.picotech.picoscope.enums.**PS4000WaveType**
Bases: enum.IntEnum

An enumeration.

SINE = 0

SQUARE = 1

TRIANGLE = 2

RAMP_UP = 3

RAMP_DOWN = 4

SINC = 5

GAUSSIAN = 6

HALF_SINE = 7

DC_VOLTAGE = 8

WHITE_NOISE = 9

MAX = 10

class msl.equipment.resources.picotech.picoscope.enums.**PS4000SigGenTrigType**
Bases: enum.IntEnum

An enumeration.

RISING = 0

FALLING = 1

GATE_HIGH = 2

GATE_LOW = 3

class msl.equipment.resources.picotech.picoscope.enums.**PS4000SigGenTrigSource**
Bases: enum.IntEnum

An enumeration.

NONE = 0

SCOPE_TRIG = 1

AUX_IN = 2

EXT_IN = 3

SOFT_TRIG = 4

class msl.equipment.resources.picotech.picoscope.enums.**PS4000IndexMode**
Bases: enum.IntEnum

An enumeration.

SINGLE = 0

DUAL = 1

QUAD = 2

MAX = 3

class msl.equipment.resources.picotech.picoscope.enums.**PS4000ThresholdMode**
Bases: enum.IntEnum

An enumeration.

LEVEL = 0

WINDOW = 1

class msl.equipment.resources.picotech.picoscope.enums.**PS4000ThresholdDirection**
Bases: enum.IntEnum

An enumeration.

ABOVE = 0

BELOW = 1

RISING = 2

FALLING = 3

RISING_OR_FALLING = 4

ABOVE_LOWER = 5

BELOW_LOWER = 6

RISING_LOWER = 7

FALLING_LOWER = 8

INSIDE = 0

OUTSIDE = 1

ENTER = 2

EXIT = 3

ENTER_OR_EXIT = 4

POSITIVE_RUNT = 9

NEGATIVE_RUNT = 10

NONE = 2

class msl.equipment.resources.picotech.picoscope.enums.**PS4000TriggerState**
Bases: enum.IntEnum

An enumeration.

DONT_CARE = 0

TRUE = 1

FALSE = 2

MAX = 3

class msl.equipment.resources.picotech.picoscope.enums.**PS4000RatioMode**
Bases: enum.IntEnum

An enumeration.

NONE = 0

AGGREGATE = 1

AVERAGE = 2

class msl.equipment.resources.picotech.picoscope.enums.**PS4000PulseWidthType**
Bases: enum.IntEnum

An enumeration.

NONE = 0

LESS_THAN = 1

GREATER_THAN = 2

IN_RANGE = 3

OUT_OF_RANGE = 4

class msl.equipment.resources.picotech.picoscope.enums.**PS4000Ps4000HoldOffType**
Bases: enum.IntEnum

An enumeration.

TIME = 0

MAX = 1

class msl.equipment.resources.picotech.picoscope.enums.**PS4000FrequencyCounterRange**
Bases: enum.IntEnum

An enumeration.

FC_2K = 0

FC_20K = 1

FC_20 = 2

FC_200 = 3

FC_MAX = 4

class msl.equipment.resources.picotech.picoscope.enums.**PS4000AExtraOperations**
Bases: enum.IntEnum

An enumeration.

OFF = 0

WHITENOISE = 1

PRBS = 2

class msl.equipment.resources.picotech.picoscope.enums.**PS4000ABandwidthLimiter**
Bases: enum.IntEnum

An enumeration.

BW_FULL = 0

BW_20KHZ = 1

class msl.equipment.resources.picotech.picoscope.enums.**PS4000ACoupling**
Bases: enum.IntEnum

An enumeration.

AC = 0

DC = 1

class msl.equipment.resources.picotech.picoscope.enums.**PS4000AChannel**
Bases: enum.IntEnum

An enumeration.

A = 0

B = 1

C = 2

D = 3

MAX_4_CHANNELS = 4

E = 4

F = 5

G = 6

H = 7

EXT = 8

MAX_CHANNELS = 8

AUX = 9

MAX_TRIGGER_SOURCES = 10

PULSE_WIDTH_SOURCE = 268435456

class msl.equipment.resources.picotech.picoscope.enums.**PS4000AChannelBufferIndex**
Bases: enum.IntEnum

An enumeration.

A_MAX = 0

A_MIN = 1

B_MAX = 2

B_MIN = 3

C_MAX = 4

C_MIN = 5

D_MAX = 6

D_MIN = 7

E_MAX = 8

E_MIN = 9

F_MAX = 10

F_MIN = 11

G_MAX = 12

G_MIN = 13

H_MAX = 14

H_MIN = 15

MAX = 16

class msl.equipment.resources.picotech.picoscope.enums.**PS4000ARange**
Bases: enum.IntEnum

An enumeration.

R_10MV = 0

R_20MV = 1

R_50MV = 2

R_100MV = 3

R_200MV = 4

R_500MV = 5

R_1V = 6

R_2V = 7

R_5V = 8

R_10V = 9

R_20V = 10

R_50V = 11

R_100V = 12

R_200V = 13

R_MAX = 14

class msl.equipment.resources.picotech.picoscope.enums.**PS4000AResistanceRange**
Bases: enum.IntEnum

An enumeration.

R_315K = 512

R_1100K = 513

R_10M = 514

R_MAX = 3

R_ADCV = 268435456

class msl.equipment.resources.picotech.picoscope.enums.**PS4000AEtsMode**
Bases: enum.IntEnum

An enumeration.

OFF = 0

FAST = 1

SLOW = 2

MAX = 3

class msl.equipment.resources.picotech.picoscope.enums.**PS4000ATimeUnits**
Bases: enum.IntEnum

An enumeration.

FS = 0

PS = 1

NS = 2

US = 3

MS = 4

S = 5

MAX = 6

class msl.equipment.resources.picotech.picoscope.enums.**PS4000ASweepType**
Bases: enum.IntEnum

An enumeration.

UP = 0

DOWN = 1

UPDOWN = 2

DOWNUP = 3

MAX = 4

class msl.equipment.resources.picotech.picoscope.enums.**PS4000AWaveType**
Bases: enum.IntEnum

An enumeration.

SINE = 0

SQUARE = 1

TRIANGLE = 2

RAMP_UP = 3

RAMP_DOWN = 4

SINC = 5

GAUSSIAN = 6

HALF_SINE = 7

DC_VOLTAGE = 8

WHITE_NOISE = 9

MAX = 10

class msl.equipment.resources.picotech.picoscope.enums.**PS4000AChannelLed**
Bases: enum.IntEnum

An enumeration.

OFF = 0

RED = 1

GREEN = 2

class msl.equipment.resources.picotech.picoscope.enums.**PS4000AMetaType**
Bases: enum.IntEnum

An enumeration.

UNIT_INFO = 0

DEVICE_CAPABILITY = 1

DEVICE_SETTINGS = 2

SIGNAL_GENERATOR_SETTINGS = 3

class msl.equipment.resources.picotech.picoscope.enums.**PS4000AMetaOperation**
Bases: enum.IntEnum

An enumeration.

READ = 0

WRITE = 1

class msl.equipment.resources.picotech.picoscope.enums.**PS4000AMetaFormat**
Bases: enum.IntEnum

An enumeration.

COMMA_SEPERATED = 0

XML = 1

class msl.equipment.resources.picotech.picoscope.enums.**PS4000ASigGenTrigType**
Bases: enum.IntEnum

An enumeration.

RISING = 0

FALLING = 1

GATE_HIGH = 2

GATE_LOW = 3

class msl.equipment.resources.picotech.picoscope.enums.**PS4000ASigGenTrigSource**
Bases: enum.IntEnum

An enumeration.

NONE = 0

SCOPE_TRIG = 1

AUX_IN = 2

EXT_IN = 3

SOFT_TRIG = 4

class msl.equipment.resources.picotech.picoscope.enums.**PS4000AIndexMode**
Bases: enum.IntEnum

An enumeration.

SINGLE = 0

DUAL = 1

QUAD = 2

MAX = 3

class msl.equipment.resources.picotech.picoscope.enums.**PS4000AThresholdMode**
Bases: enum.IntEnum

An enumeration.

LEVEL = 0

WINDOW = 1

class msl.equipment.resources.picotech.picoscope.enums.**PS4000AThresholdDirection**
Bases: enum.IntEnum

An enumeration.

ABOVE = 0

BELOW = 1

RISING = 2

FALLING = 3

RISING_OR_FALLING = 4

ABOVE_LOWER = 5

BELOW_LOWER = 6

RISING_LOWER = 7

FALLING_LOWER = 8

INSIDE = 0

OUTSIDE = 1

ENTER = 2

EXIT = 3

ENTER_OR_EXIT = 4

POSITIVE_RUNT = 9

NEGATIVE_RUNT = 10

NONE = 2

class msl.equipment.resources.picotech.picoscope.enums.**PS4000ATriggerState**
Bases: enum.IntEnum

An enumeration.

DONT_CARE = 0

TRUE = 1

FALSE = 2

MAX = 3

class msl.equipment.resources.picotech.picoscope.enums.**PS4000ASensorState**
Bases: enum.IntEnum

An enumeration.

CONNECT_STATE_FLOATING = 0

SENSOR_STATE_CONNECTED = 1

class msl.equipment.resources.picotech.picoscope.enums.**PS4000AFrequencyCounterRang**
Bases: enum.IntEnum

An enumeration.

FC_2K = 0

FC_20K = 1

FC_20 = 2

FC_200 = 3

FC_MAX = 4

class msl.equipment.resources.picotech.picoscope.enums.**PS4000AConditionsInfo**
Bases: enum.IntEnum

An enumeration.

CLEAR = 1

ADD = 2

class msl.equipment.resources.picotech.picoscope.enums.**PS4000ARatioMode**
Bases: enum.IntEnum

An enumeration.

NONE = 0

AGGREGATE = 1

DECIMATE = 2

AVERAGE = 4

DISTRIBUTION = 8

class msl.equipment.resources.picotech.picoscope.enums.**PS4000APulseWidthType**
Bases: enum.IntEnum

An enumeration.

NONE = 0

LESS_THAN = 1

GREATER_THAN = 2

IN_RANGE = 3

OUT_OF_RANGE = 4

class msl.equipment.resources.picotech.picoscope.enums.**PS4000AChannelInfo**
Bases: enum.IntEnum

An enumeration.

RANGES = 0

RESISTANCES = 1

class msl.equipment.resources.picotech.picoscope.enums.**PS4000APicoStringValue**
Bases: enum.IntEnum

An enumeration.

MEMORY = 0

MEMORY_NO_OF_SEGMENTS = 1

MEMORY_MAX_SAMPLES = 2

NO_OF_CHANNELS = 3

ARRAY_OF_CHANNELS = 4

CHANNEL = 5

CHANNEL_NAME = 6

CHANNEL_RANGE = 7

CHANNEL_COUPLING = 8

CHANNEL_ENABLED = 9

CHANNEL_ANALOGUE_OFFSET = 10

CHANNEL_BANDWIDTH = 11

TRIGGER = 12

TRIGGER_AUXIO_OUTPUT_ENABLED = 13

TRIGGER_AUTO_TRIGGER_MILLISECONDS = 14

TRIGGER_PROPERTIES = 15

NO_OF_TRIGGER_PROPERTIES = 16

TRIGGER_PROPERTIES_CHANNEL = 17

TRIGGER_PROPERTIES_THRESHOLD_UPPER = 18

TRIGGER_PROPERTIES_THRESHOLD_UPPER_HYSTERESIS = 19

TRIGGER_PROPERTIES_THRESHOLD_LOWER = 20

TRIGGER_PROPERTIES_THRESHOLD_LOWER_HYSTERESIS = 21

TRIGGER_PROPERTIES_THRESHOLD_MODE = 22

TRIGGER_ARRAY_OF_BLOCK_CONDITIONS = 23

TRIGGER_NO_OF_BLOCK_CONDITIONS = 24

TRIGGER_CONDITIONS = 25

TRIGGER_NO_OF_CONDITIONS = 26

TRIGGER_CONDITION_SOURCE = 27

TRIGGER_CONDITION_STATE = 28

TRIGGER_DIRECTION = 29

TRIGGER_NO_OF_DIRECTIONS = 30

TRIGGER_DIRECTION_CHANNEL = 31

TRIGGER_DIRECTION_DIRECTION = 32

TRIGGER_DELAY = 33

TRIGGER_DELAY_MS = 34

FREQUENCY_COUNTER = 35

FREQUENCY_COUNTER_ENABLED = 36

FREQUENCY_COUNTER_CHANNEL = 37

FREQUENCY_COUNTER_RANGE = 38

FREQUENCY_COUNTER_TRESHOLDMAJOR = 39

FREQUENCY_COUNTER_TRESHOLDMINOR = 40

PULSE_WIDTH_PROPERTIES = 41

PULSE_WIDTH_PROPERTIES_DIRECTION = 42

PULSE_WIDTH_PROPERTIES_LOWER = 43

PULSE_WIDTH_PROPERTIES_UPPER = 44

PULSE_WIDTH_PROPERTIES_TYPE = 45

PULSE_WIDTH_ARRAY_OF_BLOCK_CONDITIONS = 46

PULSE_WIDTH_NO_OF_BLOCK_CONDITIONS = 47

PULSE_WIDTH_CONDITIONS = 48

PULSE_WIDTH_NO_OF_CONDITIONS = 49

PULSE_WIDTH_CONDITIONS_SOURCE = 50

PULSE_WIDTH_CONDITIONS_STATE = 51

SAMPLE_PROPERTIES = 52

SAMPLE_PROPERTIES_PRE_TRIGGER_SAMPLES = 53

SAMPLE_PROPERTIES_POST_TRIGGER_SAMPLES = 54

SAMPLE_PROPERTIES_TIMEBASE = 55

SAMPLE_PROPERTIES_NO_OF_CAPTURES = 56

SAMPLE_PROPERTIES_RESOLUTION = 57

SAMPLE_PROPERTIES_OVERLAPPED = 58

SAMPLE_PROPERTIES_OVERLAPPED_DOWN_SAMPLE_RATIO = 59

SAMPLE_PROPERTIES_OVERLAPPED_DOWN_SAMPLE_RATIO_MODE = 60

SAMPLE_PROPERTIES_OVERLAPPED_REQUERSTED_NO_OF_SAMPLES = 61

SAMPLE_PROPERTIES_OVERLAPPED_SEGMENT_INDEX_FROM = 62

SAMPLE_PROPERTIES_OVERLAPPED_SEGMENT_INDEX_TO = 63

SIGNAL_GENERATOR = 64

SIGNAL_GENERATOR_BUILT_IN = 65

SIGNAL_GENERATOR_BUILT_IN_WAVE_TYPE = 66

SIGNAL_GENERATOR_BUILT_IN_START_FREQUENCY = 67

SIGNAL_GENERATOR_BUILT_IN_STOP_FREQUENCY = 68

SIGNAL_GENERATOR_BUILT_IN_INCREMENT = 69

SIGNAL_GENERATOR_BUILT_IN_DWELL_TIME = 70

SIGNAL_GENERATOR_AWG = 71

SIGNAL_GENERATOR_AWG_START_DELTA_PHASE = 72

SIGNAL_GENERATOR_AWG_STOP_DELTA_PHASE = 73

SIGNAL_GENERATOR_AWG_DELTA_PHASE_INCREMENT = 74

SIGNAL_GENERATOR_AWG_DWELL_COUNT = 75

SIGNAL_GENERATOR_AWG_INDEX_MODE = 76

SIGNAL_GENERATOR_AWG_WAVEFORM_SIZE = 77

SIGNAL_GENERATOR_ARRAY_OF_AWG_WAVEFORM_VALUES = 78

SIGNAL_GENERATOR_OFFSET_VOLTAGE = 79

SIGNAL_GENERATOR_PK_TO_PK = 80

SIGNAL_GENERATOR_OPERATION = 81

SIGNAL_GENERATOR_SHOTS = 82

SIGNAL_GENERATOR_SWEEPS = 83

SIGNAL_GENERATOR_SWEEP_TYPE = 84

SIGNAL_GENERATOR_TRIGGER_TYPE = 85

SIGNAL_GENERATOR_TRIGGER_SOURCE = 86

SIGNAL_GENERATOR_EXT_IN_THRESHOLD = 87

ETS = 88

ETS_STATE = 89

ETS_CYCLE = 90

ETS_INTERLEAVE = 91

ETS_SAMPLE_TIME_PICOSECONDS = 92

class msl.equipment.resources.picotech.picoscope.enums.**PS5000Channel**
Bases: enum.IntEnum

An enumeration.

A = 0

B = 1

C = 2

D = 3

EXT = 4

MAX_CHANNELS = 4

AUX = 5

MAX_TRIGGER_SOURCES = 6

class msl.equipment.resources.picotech.picoscope.enums.**PS5000ChannelBufferIndex**
Bases: enum.IntEnum

An enumeration.

A_MAX = 0

A_MIN = 1

B_MAX = 2

B_MIN = 3

C_MAX = 4

C_MIN = 5

D_MAX = 6

D_MIN = 7

MAX = 8

class msl.equipment.resources.picotech.picoscope.enums.**PS5000Range**
Bases: enum.IntEnum

An enumeration.

R_10MV = 0

R_20MV = 1

R_50MV = 2

R_100MV = 3

R_200MV = 4

R_500MV = 5

R_1V = 6

R_2V = 7

R_5V = 8

R_10V = 9

R_20V = 10

R_50V = 11

R_MAX = 12

class msl.equipment.resources.picotech.picoscope.enums.**PS5000EtsMode**
Bases: enum.IntEnum

An enumeration.

OFF = 0

FAST = 1

SLOW = 2

MAX = 3

class msl.equipment.resources.picotech.picoscope.enums.**PS5000TimeUnits**
Bases: enum.IntEnum

An enumeration.

FS = 0

PS = 1

NS = 2

US = 3

MS = 4

S = 5

MAX = 6

class msl.equipment.resources.picotech.picoscope.enums.**PS5000SweepType**
Bases: enum.IntEnum

An enumeration.

UP = 0

DOWN = 1

UPDOWN = 2

DOWNUP = 3

MAX = 4

class msl.equipment.resources.picotech.picoscope.enums.**PS5000WaveType**
Bases: enum.IntEnum

An enumeration.

SINE = 0

SQUARE = 1

TRIANGLE = 2

RAMP_UP = 3

RAMP_DOWN = 4

SINC = 5

GAUSSIAN = 6

HALF_SINE = 7

DC_VOLTAGE = 8

WHITE_NOISE = 9

MAX = 10

class msl.equipment.resources.picotech.picoscope.enums.**PS5000SigGenTrigType**
Bases: enum.IntEnum

An enumeration.

RISING = 0

FALLING = 1

GATE_HIGH = 2

GATE_LOW = 3

class msl.equipment.resources.picotech.picoscope.enums.**PS5000SigGenTrigSource**
Bases: enum.IntEnum

An enumeration.

NONE = 0

SCOPE_TRIG = 1

AUX_IN = 2

EXT_IN = 3

SOFT_TRIG = 4

class msl.equipment.resources.picotech.picoscope.enums.**PS5000IndexMode**
Bases: enum.IntEnum

An enumeration.

SINGLE = 0

DUAL = 1

QUAD = 2

MAX = 3

class msl.equipment.resources.picotech.picoscope.enums.**PS5000ThresholdMode**
Bases: enum.IntEnum

An enumeration.

LEVEL = 0

WINDOW = 1

class msl.equipment.resources.picotech.picoscope.enums.**PS5000ThresholdDirection**
Bases: enum.IntEnum

An enumeration.

ABOVE = 0

BELOW = 1

RISING = 2

FALLING = 3

RISING_OR_FALLING = 4

INSIDE = 0

OUTSIDE = 1

ENTER = 2

EXIT = 3

ENTER_OR_EXIT = 4

NONE = 2

class msl.equipment.resources.picotech.picoscope.enums.**PS5000TriggerState**
Bases: enum.IntEnum

An enumeration.

DONT_CARE = 0

TRUE = 1

FALSE = 2

MAX = 3

class msl.equipment.resources.picotech.picoscope.enums.**PS5000RatioMode**
Bases: enum.IntEnum

An enumeration.

NONE = 0

AGGREGATE = 1

DECIMATE = 2

AVERAGE = 4

DISTRIBUTION = 8

class msl.equipment.resources.picotech.picoscope.enums.**PS5000PulseWidthType**
Bases: enum.IntEnum

An enumeration.

NONE = 0

LESS_THAN = 1

GREATER_THAN = 2

IN_RANGE = 3

OUT_OF_RANGE = 4

class msl.equipment.resources.picotech.picoscope.enums.**PS5000ChannelInfo**
Bases: enum.IntEnum

An enumeration.

RANGES = 0

class msl.equipment.resources.picotech.picoscope.enums.**PS5000ADeviceResolution**
Bases: enum.IntEnum

An enumeration.

RES_8BIT = 0

RES_12BIT = 1

RES_14BIT = 2

RES_15BIT = 3

RES_16BIT = 4

class msl.equipment.resources.picotech.picoscope.enums.**PS5000AExtraOperations**
Bases: enum.IntEnum

An enumeration.

OFF = 0

WHITENOISE = 1

PRBS = 2

class msl.equipment.resources.picotech.picoscope.enums.**PS5000ABandwidthLimiter**
Bases: enum.IntEnum

An enumeration.

BW_FULL = 0

BW_20MHZ = 1

class msl.equipment.resources.picotech.picoscope.enums.**PS5000ACoupling**
Bases: enum.IntEnum

An enumeration.

AC = 0

DC = 1

class msl.equipment.resources.picotech.picoscope.enums.**PS5000AChannel**
Bases: enum.IntEnum

An enumeration.

A = 0

B = 1

C = 2

D = 3

EXT = 4

MAX_CHANNELS = 4

AUX = 5

MAX_TRIGGER_SOURCES = 6

class msl.equipment.resources.picotech.picoscope.enums.**PS5000AChannelBufferIndex**
Bases: enum.IntEnum

An enumeration.

A_MAX = 0

A_MIN = 1

B_MAX = 2

B_MIN = 3

C_MAX = 4

C_MIN = 5

D_MAX = 6

D_MIN = 7

MAX = 8

class msl.equipment.resources.picotech.picoscope.enums.**PS5000ARange**
Bases: enum.IntEnum

An enumeration.

R_10MV = 0

R_20MV = 1

R_50MV = 2

R_100MV = 3

R_200MV = 4

R_500MV = 5

R_1V = 6

R_2V = 7

R_5V = 8

R_10V = 9

R_20V = 10

R_50V = 11

R_MAX = 12

class msl.equipment.resources.picotech.picoscope.enums.**PS5000AEtsMode**
Bases: enum.IntEnum

An enumeration.

OFF = 0

FAST = 1

SLOW = 2

MAX = 3

class msl.equipment.resources.picotech.picoscope.enums.**PS5000ATimeUnits**
Bases: enum.IntEnum

An enumeration.

FS = 0

PS = 1

NS = 2

US = 3

MS = 4

S = 5

MAX = 6

class msl.equipment.resources.picotech.picoscope.enums.**PS5000ASweepType**
Bases: enum.IntEnum

An enumeration.

UP = 0

DOWN = 1

UPDOWN = 2

DOWNUP = 3

MAX = 4

class msl.equipment.resources.picotech.picoscope.enums.**PS5000AWaveType**
Bases: enum.IntEnum

An enumeration.

SINE = 0

SQUARE = 1

TRIANGLE = 2

RAMP_UP = 3

RAMP_DOWN = 4

SINC = 5

GAUSSIAN = 6

HALF_SINE = 7

DC_VOLTAGE = 8

WHITE_NOISE = 9

MAX = 10

class msl.equipment.resources.picotech.picoscope.enums.**PS5000ASigGenTrigType**
Bases: enum.IntEnum

An enumeration.

RISING = 0

FALLING = 1

GATE_HIGH = 2

GATE_LOW = 3

class msl.equipment.resources.picotech.picoscope.enums.**PS5000ASigGenTrigSource**
Bases: enum.IntEnum

An enumeration.

NONE = 0

SCOPE_TRIG = 1

AUX_IN = 2

EXT_IN = 3

SOFT_TRIG = 4

class msl.equipment.resources.picotech.picoscope.enums.**PS5000AIndexMode**
Bases: enum.IntEnum

An enumeration.

SINGLE = 0

DUAL = 1

QUAD = 2

MAX = 3

class msl.equipment.resources.picotech.picoscope.enums.**PS5000AThresholdMode**
Bases: enum.IntEnum

An enumeration.

LEVEL = 0

WINDOW = 1

class msl.equipment.resources.picotech.picoscope.enums.**PS5000AThresholdDirection**
Bases: enum.IntEnum

An enumeration.

ABOVE = 0

BELOW = 1

RISING = 2

FALLING = 3

RISING_OR_FALLING = 4

ABOVE_LOWER = 5

BELOW_LOWER = 6

RISING_LOWER = 7

FALLING_LOWER = 8

INSIDE = 0

OUTSIDE = 1

ENTER = 2

EXIT = 3

ENTER_OR_EXIT = 4

POSITIVE_RUNT = 9

NEGATIVE_RUNT = 10

NONE = 2

class msl.equipment.resources.picotech.picoscope.enums.**PS5000ATriggerState**
Bases: enum.IntEnum

An enumeration.

DONT_CARE = 0

TRUE = 1

FALSE = 2

MAX = 3

class msl.equipment.resources.picotech.picoscope.enums.**PS5000ATriggerWithinPreTrig**
Bases: enum.IntEnum

An enumeration.

DISABLE = 0

ARM = 1

class msl.equipment.resources.picotech.picoscope.enums.**PS5000ARatioMode**
Bases: enum.IntEnum

An enumeration.

NONE = 0

AGGREGATE = 1

DECIMATE = 2

AVERAGE = 4

DISTRIBUTION = 8

class msl.equipment.resources.picotech.picoscope.enums.**PS5000APulseWidthType**
Bases: enum.IntEnum

An enumeration.

NONE = 0

LESS_THAN = 1

GREATER_THAN = 2

IN_RANGE = 3

OUT_OF_RANGE = 4

class msl.equipment.resources.picotech.picoscope.enums.**PS5000AChannelInfo**
Bases: enum.IntEnum

An enumeration.

RANGES = 0

class msl.equipment.resources.picotech.picoscope.enums.**PS6000ExternalFrequency**
Bases: enum.IntEnum

An enumeration.

EF_OFF = 0

EF_5MHZ = 1

EF_10MHZ = 2

EF_20MHZ = 3

EF_25MHZ = 4

EF_MAX = 5

class msl.equipment.resources.picotech.picoscope.enums.**PS6000BandwidthLimiter**
Bases: enum.IntEnum

An enumeration.

BW_FULL = 0

BW_20MHZ = 1

BW_25MHZ = 2

class msl.equipment.resources.picotech.picoscope.enums.**PS6000Channel**
Bases: enum.IntEnum

An enumeration.

A = 0

B = 1

C = 2

D = 3

EXT = 4

MAX_CHANNELS = 4

AUX = 5

MAX_TRIGGER_SOURCES = 6

class msl.equipment.resources.picotech.picoscope.enums.**PS6000ChannelBufferIndex**
Bases: enum.IntEnum

An enumeration.

A_MAX = 0

A_MIN = 1

B_MAX = 2

B_MIN = 3

C_MAX = 4

C_MIN = 5

D_MAX = 6

D_MIN = 7

MAX = 8

class msl.equipment.resources.picotech.picoscope.enums.**PS6000Range**
Bases: enum.IntEnum

An enumeration.

R_10MV = 0

R_20MV = 1

R_50MV = 2

R_100MV = 3

R_200MV = 4

R_500MV = 5

R_1V = 6

R_2V = 7

R_5V = 8

R_10V = 9

R_20V = 10

R_50V = 11

R_MAX = 12

class msl.equipment.resources.picotech.picoscope.enums.**PS6000Coupling**
Bases: enum.IntEnum

An enumeration.

AC = 0

DC_1M = 1

DC_50R = 2

class msl.equipment.resources.picotech.picoscope.enums.**PS6000EtsMode**
Bases: enum.IntEnum

An enumeration.

OFF = 0

FAST = 1

SLOW = 2

MAX = 3

class msl.equipment.resources.picotech.picoscope.enums.**PS6000TimeUnits**
Bases: enum.IntEnum

An enumeration.

FS = 0

PS = 1

NS = 2

US = 3

MS = 4

S = 5

MAX = 6

class msl.equipment.resources.picotech.picoscope.enums.**PS6000SweepType**
Bases: enum.IntEnum

An enumeration.

UP = 0

DOWN = 1

UPDOWN = 2

DOWNUP = 3

MAX = 4

class msl.equipment.resources.picotech.picoscope.enums.**PS6000WaveType**
Bases: enum.IntEnum

An enumeration.

SINE = 0

SQUARE = 1

TRIANGLE = 2

RAMP_UP = 3

RAMP_DOWN = 4

SINC = 5

GAUSSIAN = 6

HALF_SINE = 7

DC_VOLTAGE = 8

MAX = 9

class msl.equipment.resources.picotech.picoscope.enums.**PS6000ExtraOperations**
Bases: enum.IntEnum

An enumeration.

OFF = 0

WHITENOISE = 1

PRBS = 2

class msl.equipment.resources.picotech.picoscope.enums.**PS6000SigGenTrigType**
Bases: enum.IntEnum

An enumeration.

RISING = 0

FALLING = 1

GATE_HIGH = 2

GATE_LOW = 3

class msl.equipment.resources.picotech.picoscope.enums.**PS6000SigGenTrigSource**
Bases: enum.IntEnum

An enumeration.

NONE = 0

SCOPE_TRIG = 1

AUX_IN = 2

EXT_IN = 3

SOFT_TRIG = 4

TRIGGER_RAW = 5

class msl.equipment.resources.picotech.picoscope.enums.**PS6000IndexMode**
Bases: enum.IntEnum

An enumeration.

SINGLE = 0

DUAL = 1

QUAD = 2

MAX = 3

class msl.equipment.resources.picotech.picoscope.enums.**PS6000ThresholdMode**
Bases: enum.IntEnum

An enumeration.

LEVEL = 0

WINDOW = 1

class msl.equipment.resources.picotech.picoscope.enums.**PS6000ThresholdDirection**
Bases: enum.IntEnum

An enumeration.

ABOVE = 0

BELOW = 1

RISING = 2

FALLING = 3

RISING_OR_FALLING = 4

ABOVE_LOWER = 5

BELOW_LOWER = 6

RISING_LOWER = 7

FALLING_LOWER = 8

INSIDE = 0

OUTSIDE = 1

ENTER = 2

EXIT = 3

ENTER_OR_EXIT = 4

POSITIVE_RUNT = 9

NEGATIVE_RUNT = 10

NONE = 2

class msl.equipment.resources.picotech.picoscope.enums.**PS6000TriggerState**
Bases: enum.IntEnum

An enumeration.

DONT_CARE = 0

TRUE = 1

FALSE = 2

MAX = 3

class msl.equipment.resources.picotech.picoscope.enums.**PS6000RatioMode**
Bases: enum.IntEnum

An enumeration.

NONE = 0

AGGREGATE = 1

AVERAGE = 2

DECIMATE = 4

DISTRIBUTION = 8

class msl.equipment.resources.picotech.picoscope.enums.**PS6000PulseWidthType**
Bases: enum.IntEnum

An enumeration.

NONE = 0

LESS_THAN = 1

GREATER_THAN = 2

IN_RANGE = 3

OUT_OF_RANGE = 4

msl.equipment.resources.picotech.picoscope.errors module

Exceptions and error codes defined in the Pico Technology SDK v10.6.10.24

msl.equipment.resources.picotech.picoscope.functions module

Functions defined in the Pico Technology SDK v10.6.10.24

msl.equipment.resources.picotech.picoscope.helper module

The functions in this module are only helper functions that were initially used for wrapping the PicoScope SDK in Python. There are no user-facing functions here, only those used by a developer.

These functions are used to

Print the following to stdout

- the #define constants
- the function signatures for the PicoScope subclasses
- functions with similar function signatures

Create the following files

- picoscope_enums.py
- picoscope_structs.py
- picoscope_callbacks.py
- picoscope_function_pointers.py

msl.equipment.resources.picotech.picoscope.helper.**parse_pico_scope_api_header**(*pa*
Parse a PicoScope header file.

Args: path (str): The path to a PicoScope header file

Returns: dict: {'enums': {}, 'defines': {}, 'functions': {}, 'structs': {}, 'functypes': {}}

msl.equipment.resources.picotech.picoscope.helper.**print_define_statements**(*header_d*
Print the #define constants in the PicoScope header files to stdout

The output is copied and pasted to the appropriate PicoScope subclass.

For example, the stdout text below

ps5000aApi

is copied to

class PicoScope5000A(PicoScope):

msl.equipment.resources.picotech.picoscope.helper.**create_picoscope_enums_file**(*hea*
pi-
cos
tus

Creates the _picoscope_enums.py file

msl.equipment.resources.picotech.picoscope.helper.**create_picoscope_structs_file**(
Creates the _picoscope_structs.py file

msl.equipment.resources.picotech.picoscope.helper.**check_enum_struct_names**()
Ensure that none of the items in ENUM_DATA_TYPE_NAMES are in STRUCT_DATA_TYPE_ALIASES

msl.equipment.resources.picotech.picoscope.helper.**create_callbacks_file**(*header_dict*
Create the _picoscope_callbacks.py file

msl.equipment.resources.picotech.picoscope.helper.**ctypes_map**(*dtype*, *hkey*)

msl.equipment.resources.picotech.picoscope.helper.**create_picoscope_functions_fil**
Create the _picoscope_functions.py file. The lists in this file are used for creating ctypes._FuncPtr objects

msl.equipment.resources.picotech.picoscope.helper.**print_class_def_signatures**(*head*

msl.equipment.resources.picotech.picoscope.helper.**print_common_functions**(*header_di*
ps_name)

msl.equipment.resources.picotech.picoscope.picoscope module

Base class for a PicoScope from Pico Technology.

msl.equipment.resources.picotech.picoscope.picoscope.**enumerate_units**()
Find the PicoScopes that are connected to the computer.

This function counts the number of PicoScopes connected to the computer, and returns a list of serial numbers as a string.

Note: It seems as though you cannot call this function after you have opened a connection to a PicoScope.

Returns `list` of `str` – A list of serial numbers of the PicoScopes that were found.

class `msl.equipment.resources.picotech.picoscope.picoscope.`**PicoScope**(*record*, *func_ptrs*)

Bases: *msl.equipment.connection_msl.ConnectionSDK*

Use the PicoScope SDK to communicate with the oscilloscope.

Do not instantiate this class directly. Use the factory method, *msl.equipment.factory.connect*, or the *record* object itself, *record.connect()*, to connect to the equipment.

The *record.connection.properties* dictionary for a PicoScope supports the following key-value pairs:

```
'open_unit': bool,  # default is True
'open_unit_async': bool,  # default is False
'auto_select_power': bool  # for PicoScopes that can be powered by an␣
↪AC adaptor or a USB cable, default is True
'resolution': '14bit',  # only valid for ps5000a, default is '8bit'
```

The SDK version that was initially used to create this base class and the PicoScope subclasses was *Pico Technology SDK 64-bit v10.6.10.24*

Parameters

- **record** (*EquipmentRecord*) – An equipment record from an **Equipment-Register** *Database*.
- **func_ptrs** (*functions*) – The appropriate function-pointer list for the SDK.

handle
Returns the handle to the SDK library.

channel
`dict` of *channel.PicoScopeChannel* – The information about each channel

dt
`float` – The time between voltage samples (i.e., delta t).

pre_trigger
`float` – The number of seconds that data was acquired for before the trigger event.

close_unit()
Disconnect from the PicoScope.

disconnect()
Disconnect from the PicoScope.

get_unit_info(*info=None*, *include_name=True*)
Retrieves information about the PicoScope.

This function retrieves information about the specified oscilloscope. If the device fails to open, or no device is opened only the driver version is available.

Parameters

- **info** (`PicoScopeInfo`, optional) – An enum value, or if `None` then request all information from the PicoScope.

- **include_name** (bool, optional) – If True then includes the enum member name as a prefix. For example, return CAL_DATE: 09Aug16 if True else 09Aug16.

Returns str – The requested information from the PicoScope.

is_ready()
Has the PicoScope collecting the requested number of samples?

This function may be used instead of a callback function to receive data from *run_block()*. To use this method, pass None as the callback parameter in *run_block()*. You must then poll the driver to see if it has finished collecting the requested samples.

Returns bool – Whether the PicoScope has collected the requested number of samples.

maximum_value()
int: This function returns the maximum ADC count.

minimum_value()
int: This function returns the minimum ADC count.

ping_unit()
Ping the PicoScope.

This function can be used to check that the already opened device is still connected to the USB port and communication is successful.

run_block(*pre_trigger=0.0*, *callback=None*, *segment_index=0*)
Start collecting data in block mode.

All input arguments are ignored for ps2000 and ps3000.

Parameters

- **pre_trigger** (float) – The number of seconds before the trigger event to start acquiring data.
- **segment_index** (int) – Specifies which memory segment to save the data to (see manual).
- **callback** (callbacks.BlockReady) – A BlockReady callback function.

run_streaming(*pre_trigger=0.0*, *auto_stop=True*, *factor=1*, *ratio_mode='NONE'*)
Start collecting data in streaming mode.

This function tells the oscilloscope to start collecting data in streaming mode. When data has been collected from the device it is down sampled if necessary and then delivered to the application. Call get_streaming_latest_values() to retrieve the data.

When a trigger is set, the total number of samples stored in the driver is the sum of *max_pre_trigger_samples* and *max_post_trigger_samples*. If *auto_stop* is false then this will become the maximum number of samples without down sampling.

The *ratio_mode* argument is ignored for ps4000 and ps5000.

Parameters

- **pre_trigger** (float) – The number of seconds before the trigger event to start acquiring data.
- **auto_stop** (bool) – A flag that specifies if the streaming should stop when all of samples have been captured.
- **factor** (int) – The down-sampling factor that will be applied to the raw data.
- **ratio_mode** (enum.IntEnum) – Which down-sampling mode to use.

set_channel(*channel*, *coupling='dc'*, *scale='10V'*, *offset=0.0*, *bandwidth='full'*, *enabled=True*)

Configure a channel.

This function specifies whether an input channel is to be enabled, its input coupling type, voltage range, analog offset and bandwidth limit. Some of the arguments within this function have model-specific values. Please consult the manual according to the model you have.

The *bandwidth* argument is only used for ps6000.

The *offset* and *bandwidth* arguments are ignored for ps2000, ps3000, ps4000 and ps5000.

Parameters

- **channel** (enum.IntEnum) – The channel to be configured
- **coupling** (enum.IntEnum) – The impedance and coupling type.
- **scale** (enum.IntEnum) – The input voltage range.
- **offset** (float) – A voltage to add to the input channel before digitization. The allowable range of offsets depends on the input range selected for the channel, as obtained from *get_analogue_offset()*.
- **bandwidth** (enum.IntEnum) – The bandwidth limiter to use.
- **enabled** (bool) – Whether to enable the channel.

set_timebase(*dt*, *duration*, *segment_index=0*, *oversample=0*)

Set the timebase information.

The *segment_index* is ignored for ps2000 and ps3000.

The *oversample* argument is ignored by ps2000a, ps3000a, ps4000a and ps5000a.

Parameters

- **dt** (float) – The sampling interval, in seconds.
- **duration** (float) – The number of seconds to acquire data for.
- **segment_index** (int) – Which memory segment to save the data to.
- **oversample** (int) – The amount of over-sample required.

Raises ConnectionError – If the timebase or duration is invalid.

set_trigger(*channel*, *threshold*, *delay=0.0*, *direction='rising'*, *timeout=0.1*, *enable=True*)

Set up the trigger.

Parameters

- **channel** (enum.IntEnum) – The trigger channel.

- **threshold** (float) – The threshold voltage to signal a trigger event.
- **delay** (float) – The time, in seconds, between the trigger occurring and the first sample.
- **direction** (enum.IntEnum) – The direction in which the signal must move to cause a trigger.
- **timeout** (float) – The time, in seconds, to wait to automatically create a trigger event if no trigger event occurs. If *timeout* <= 0 then wait indefinitely for a trigger. Only accurate to the nearest millisecond.
- **enable** (bool) – Set to False to disable the trigger for this channel. Not used for ps2000 or ps3000.

stop()
Stop the oscilloscope from sampling data.

If this function is called before a trigger event occurs, then the oscilloscope may not contain valid data.

wait_until_ready()
Blocking function to wait for the scope to finish acquiring data.

set_pulse_width_qualifier(*conditions*, *direction*, *lower*, *upper*, *pulse_width_type*)
This function sets up pulse width qualification, which can be used on its own for pulse width triggering or combined with other triggering to produce more complex triggers. The pulse width qualifier is set by defining a list of PwqConditions` structures.

msl.equipment.resources.picotech.picoscope.picoscope_2k3k module

Base class for the ps2000 and ps3000 PicoScopes.

class msl.equipment.resources.picotech.picoscope.picoscope_2k3k.**PicoScope2k3k**(*record*, *func_p*

Bases: *msl.equipment.resources.picotech.picoscope.picoscope.PicoScope*

Use the PicoScope SDK to communicate with ps2000 and ps3000 oscilloscopes.

Do not instantiate this class directly. Use the factory method, *msl.equipment.factory.connect*, or the *record* object itself, *record.connect()*, to connect to the equipment.

Parameters

- **record** (*EquipmentRecord*) – An equipment record from an **Equipment-Register** *Database*.
- **func_ptrs** (*functions*) – The appropriate function-pointer list for the SDK.

errcheck_zero(*result*, *func*, *args*)
If the SDK function returns 0 then raise an exception.

errcheck_one(*result*, *func*, *args*)
If the SDK function returns 1 then raise an exception.

errcheck_negative_one(*result*, *func*, *args*)
If the SDK function returns -1 then raise an exception.

flash_led()
Flashes the LED on the front of the oscilloscope three times and returns within one second.

get_streaming_last_values(*lp_get_overview_buffers_max_min*)
This function is used to collect the next block of values while fast streaming is running. You must call `run_streaming_ns()` beforehand to set up fast streaming.

get_streaming_values(*no_of_values*, *no_of_samples_per_aggregate*)
This function is used after the driver has finished collecting data in fast streaming mode. It allows you to retrieve data with different aggregation ratios, and thus zoom in to and out of any region of the data.

get_streaming_values_no_aggregation(*no_of_values*)
This function retrieves raw streaming data from the driver's data store after fast streaming has stopped.

get_timebase(*timebase*, *no_of_samples*, *oversample=0*)
This function discovers which timebases are available on the oscilloscope. You should set up the channels using `set_channel()` and, if required, ETS mode using *set_ets()* first. Then call this function with increasing values of timebase, starting from 0, until you find a timebase with a sampling interval and sample count close enough to your requirements.

get_times_and_values(*time_units*, *no_of_values*)
This function is used to get values and times in block mode after calling `run_block()`.

get_values(*num_values*)
This function is used to get values in compatible streaming mode after calling `run_streaming()`, or in block mode after calling `run_block()`.

open_unit()
This function opens a PicoScope 2000/3000 Series oscilloscope. The driver can support up to 64 oscilloscopes.

open_unit_async()
This function opens a PicoScope 2000/3000 Series oscilloscope without waiting for the operation to finish. You can find out when it has finished by periodically calling *open_unit_progress()*, which returns a value of 100 when the scope is open.

The driver can support up to 64 oscilloscopes.

open_unit_progress()
This function checks on the progress of *open_unit_async()*.

The function will return a value from 0 to 100, where 100 implies that the operation is complete.

overview_buffer_status()
This function indicates whether or not the overview buffers used by `run_streaming_ns()` have overrun. If an overrun occurs, you can choose to increase the overview_buffer_size argument that you pass in the next call to `run_streaming_ns()`.

set_adv_trigger_channel_directions(*channel_a*, *channel_b*, *channel_c*, *channel_d*, *ext*)
This function sets the direction of the trigger for each channel.

set_adv_trigger_delay(*delay*, *pre_trigger_delay*)
This function sets the pre-trigger and post-trigger delays. The default action, when both these delays are zero, is to start capturing data beginning with the trigger event and to stop a specified time later. The start of capture can be delayed by using a nonzero value of `delay`. Alternatively, the start of capture can be advanced to a time before the trigger event by using a negative value of `pre_trigger_delay`. If both arguments are non-zero then their effects are added together.

set_ets(*mode*, *ets_cycles*, *ets_interleave*)
This function is used to enable or disable ETS (equivalent time sampling) and to set the ETS parameters.

set_trigger(*source*, *threshold*, *direction*, *delay*, *auto_trigger_ms*)
This function just calls *set_trigger2()*, since Python supports a floating-point value for defining the `delay` parameter.

set_trigger2(*source*, *threshold*, *direction*, *delay*, *auto_trigger_ms*)
This function is used to enable or disable triggering and its parameters. It has the same behaviour as *set_trigger()*, except that the delay parameter is a floating-point value.

For oscilloscopes that support advanced triggering, see *set_adv_trigger_channel_conditions()* and related functions.

set_adv_trigger_channel_properties(*channel_properties*, *auto_trigger_milliseconds*)
This function is used to enable or disable triggering and set its parameters.

set_adv_trigger_channel_conditions(*conditions*)
This function sets up trigger conditions on the scope's inputs. The trigger is set up by defining a list of `picoscope_structs` `TriggerConditions` structures. Each structure is the AND of the states of one scope input.

msl.equipment.resources.picotech.picoscope.picoscope_api module

Base class for the PicoScopes that have a header file which ends with *Api.h.

Namely, ps2000aApi, ps3000aApi, ps4000Api, ps4000aApi, ps5000Api, ps5000aApi and ps6000Api.

class `msl.equipment.resources.picotech.picoscope.picoscope_api.`**PicoScopeApi**(*record*, *func_ptrs*)

Bases: *msl.equipment.resources.picotech.picoscope.picoscope.PicoScope*

Base class for the PicoScopes that have a header file which ends with *Api.h.

Use the PicoScope SDK to communicate with the ps2000a, ps3000a, ps4000, ps4000a, ps5000, ps5000a and ps6000 oscilloscopes.

Do not instantiate this class directly. Use the factory method, *msl.equipment.factory.connect*, or the *record* object itself, *record.connect()*, to connect to the equipment.

Parameters

- **record** (*EquipmentRecord*) – An equipment record from an **Equipment-Register** *Database*.
- **func_ptrs** (*functions*) – The appropriate function-pointer list for the SDK.

errcheck_api(*result*, *func*, *args*)
The SDK function returns PICO_OK if the function call was successful.

change_power_source(*power_state*)
Change the power source.

This function selects the power supply mode. You must call this function if any of the following conditions arises:

•USB power is required

•the AC power adapter is connected or disconnected during use

•a USB 3.0 scope is plugged into a USB 2.0 port (indicated if any function returns the `PICO_USB3_0_DEVICE_NON_USB3_0_PORT` status code)

This function is only valid for ps3000a, ps4000a and ps5000a.

current_power_source()
This function returns the current power state of the device.

This function is only valid for ps3000a, ps4000a and ps5000a.

flash_led(*action*)
This function flashes the LED on the front of the scope without blocking the calling thread. Calls to `run_streaming()` and `run_block()` cancel any flashing started by this function. It is not possible to set the LED to be constantly illuminated, as this state is used to indicate that the scope has not been initialized.

get_analogue_offset(*voltage_range*, *coupling*)
This function is used to get the maximum and minimum allowable analog offset for a specific voltage range.

This function is invalid for ps4000 and ps5000.

get_channel_information(*channel*, *info='ranges'*)
This function queries which ranges are available on a scope device.

This function is invalid for ps5000 and ps6000.

Parameters

- **channel** (`enum.IntEnum`) – 0=ChannelA, 1=ChannelB, ...
- **info** (`enum.IntEnum`) – A `ChannelInfo` enum value or enum member name.

get_max_down_sample_ratio(*num_unaggreated_samples*, *mode='None'*, *segment_index=0*)

Returns `int` – This function returns the maximum down-sampling ratio that can be used for a given number of samples in a given down-sampling mode.

get_max_segments()
This function is valid for ps2000a, ps3000a, ps4000a and ps5000a.

Returns `int` – This function returns the maximum number of segments allowed for the opened device. This number is the maximum value of `nsegments` that can be passed to *`memory_segments()`*.

get_no_of_captures()
This function finds out how many captures are available in rapid block mode after

run_block() has been called when either the collection completed or the collection of waveforms was interrupted by calling stop(). The returned value (nCaptures) can then be used to iterate through the number of segments using *get_values()*, or in a single call to *get_values_bulk()* where it is used to calculate the toSegmentIndex parameter.

Not valid for ps5000.

get_num_of_processed_captures()
This function finds out how many captures in rapid block mode have been processed after run_block() has been called when either the collection completed or the collection of waveforms was interrupted by calling stop(). The returned value (nCaptures) can then be used to iterate through the number of segments using *get_values()*, or in a single call to *get_values_bulk()* where it is used to calculate the toSegmentIndex parameter.

Not valid for ps4000 and ps5000.

get_streaming_latest_values(*lp_ps*)
This function instructs the driver to return the next block of values to your streaming_ready() callback. You must have previously called run_streaming() beforehand to set up streaming.

get_timebase(*timebase*, *num_samples=0*, *segment_index=0*, *oversample=0*)
Since Python supports the float data type, this function returns *get_timebase2()*. The timebase that is returned is in **seconds** (not ns).

This function calculates the sampling rate and maximum number of samples for a given timebase under the specified conditions. The result will depend on the number of channels enabled by the last call to set_channel().

The *oversample* argument is only used by ps4000, ps5000 and ps6000.

get_timebase2(*timebase*, *num_samples=0*, *segment_index=0*, *oversample=0*)
This function is an upgraded version of *get_timebase()*, and returns the time interval as a float rather than an int. This allows it to return sub-nanosecond time intervals. See *get_timebase()* for a full description.

The timebase that is returned is in **seconds** (not ns).

The *oversample* argument is only used by ps4000, ps5000 and ps6000.

get_trigger_time_offset(*segment_index=0*)
This function gets the time, as two 4-byte values, at which the trigger occurred. Call it after block-mode data has been captured or when data has been retrieved from a previous block-mode capture. A 64-bit version of this function, *get_trigger_time_offset64()*, is also available.

get_trigger_time_offset64(*segment_index=0*)
This function gets the time, as a single 64-bit value, at which the trigger occurred. Call it after block-mode data has been captured or when data has been retrieved from a previous block-mode capture. A 32-bit version of this function, *get_trigger_time_offset()*, is also available.

get_values(*num_samples=None*, *start_index=0*, *factor=1*, *ratio_mode='None'*, *segment_index=0*)
This function returns block-mode data, with or without down sampling, starting at the specified sample number. It is used to get the stored data from the driver after data collection has stopped.

Parameters

- **num_samples** (int or None) – The number of samples required. If None then automatically determine the number of samples to retrieve.
- **start_index** (int) – A zero-based index that indicates the start point for data collection. It is measured in sample intervals from the start of the buffer.
- **factor** (int) – The down-sampling factor that will be applied to the raw data.
- **ratio_mode** (enum.IntEnum) – Which down-sampling mode to use. A RatioMode enum.
- **segment_index** (int) – The zero-based number of the memory segment where the data is stored.

get_values_async(*lp_data_ready*, *num_samples=None*, *start_index=0*, *factor=1*, *ratio_mode='None'*, *segment_index=0*)

This function returns data either with or without down sampling, starting at the specified sample number. It is used to get the stored data from the scope after data collection has stopped. It returns the data using the lp_data_ready callback.

Parameters

- **lp_data_ready** (*ctypes callback function*) – A DataReady *callback* function.
- **num_samples** (int or None) – The number of samples required. If None then automatically determine the number of samples to retrieve.
- **start_index** (int) – A zero-based index that indicates the start point for data collection. It is measured in sample intervals from the start of the buffer.
- **factor** (int) – The downsampling factor that will be applied to the raw data.
- **ratio_mode** (enum.IntEnum) – Which down-sampling mode to use. A RatioMode enum.
- **segment_index** (int) – The zero-based number of the memory segment where the data is stored.

get_values_bulk(*from_segment_index=0*, *to_segment_index=None*, *factor=1*, *ratio_mode='None'*)

This function retrieves waveforms captured using rapid block mode. The waveforms must have been collected sequentially and in the same run.

The *down_sample_ratio* and *down_sample_ratio_mode* arguments are ignored for ps4000 and ps5000.

get_values_overlapped(*start_index*, *down_sample_ratio*, *down_sample_ratio_mode*, *segment_index*)

This function allows you to make a deferred data-collection request, which will later be executed, and the arguments validated, when you call run_block() in block mode. The advantage of this function is that the driver makes contact with the scope only once, when you call run_block(), compared with the two contacts that occur when you use the

conventional `run_block()`, *`get_values()`* calling sequence. This slightly reduces the dead time between successive captures in block mode.

This function is invalid for ps4000 and ps5000.

`get_values_overlapped_bulk`(*start_index*, *down_sample_ratio*, *down_sample_ratio_mode*, *from_segment_index*, *to_segment_index*)

This function allows you to make a deferred data-collection request, which will later be executed, and the arguments validated, when you call `run_block()` in rapid block mode. The advantage of this method is that the driver makes contact with the scope only once, when you call `run_block()`, compared with the two contacts that occur when you use the conventional `run_block()`, *`get_values_bulk()`* calling sequence. This slightly reduces the dead time between successive captures in rapid block mode.

This function is invalid for ps4000 and ps5000.

`get_values_trigger_time_offset_bulk`(*from_segment_index*, *to_segment_index*)

This function retrieves the time offsets, as lower and upper 32-bit values, for waveforms obtained in rapid block mode. This function is provided for use in programming environments that do not support 64- bit integers. If your programming environment supports this data type, it is easier to use *`get_values_trigger_time_offset_bulk64()`*.

`get_values_trigger_time_offset_bulk64`(*from_segment_index=0*, *to_segment_index=None*)

This function retrieves the 64-bit time offsets for waveforms captured in rapid block mode.

A 32-bit version of this function, *`get_values_trigger_time_offset_bulk()`*, is available for use with programming languages that do not support 64-bit integers

`hold_off`(*holdoff*, *holdoff_type*)

This function is for backward compatibility only and is not currently used.

This function is only defined for ps2000a, ps3000a and ps4000.

`is_led_flashing`()

This function reports whether or not the LED is flashing.

This function is supported by ps4000, ps4000a and ps5000.

`is_trigger_or_pulse_width_qualifier_enabled`()

This function discovers whether a trigger, or pulse width triggering, is enabled.

`memory_segments`(*num_segments*)

This function sets the number of memory segments that the scope will use. When the scope is opened, the number of segments defaults to 1, meaning that each capture fills the scopes available memory. This function allows you to divide the memory into a number of segments so that the scope can store several waveforms sequentially.

`no_of_streaming_values`()

This function returns the number of samples available after data collection in streaming mode. Call it after calling `stop()`.

`open_unit`(*auto_select_power=True*, *resolution='8Bit'*)

Open the PicoScope for communication.

This function opens a PicoScope attached to the computer. The maximum number of units that can be opened depends on the operating system, the kernel driver and the computer.

Parameters

- **auto_select_power** (bool, optional) – PicoScopes that can be powered by either DC power or by USB power may raise PICO_POWER_SUPPLY_NOT_CONNECTED if the DC power supply is not connected. Passing in True will automatically switch to the USB power source.
- **resolution** (str, optional) – The ADC resolution: 8, 12, 14, 15 or 16Bit. Only used by the PS5000A Series and it is ignored for all other PicoScope Series.

open_unit_async(*auto_select_power=True*, *resolution='8Bit'*)

This function opens a scope without blocking the calling thread. You can find out when it has finished by periodically calling *open_unit_progress()* until that function returns a value of 100.

Parameters

- **auto_select_power** (bool, optional) – PicoScopes that can be powered by either DC power or by USB power may raise PICO_POWER_SUPPLY_NOT_CONNECTED if the DC power supply is not connected. Passing in True will automatically switch to the USB power source.
- **resolution** (str, optional) – The ADC resolution: 8, 12, 14, 15 or 16Bit. Only used by the PS5000A Series and it is ignored for all other PicoScope Series.

open_unit_progress()

This function checks on the progress of a request made to *open_unit_async()* to open a scope. The return value is from 0 to 100, where 100 implies that the operation is complete.

set_bandwidth_filter(*channel*, *bandwidth*)

This function is reserved for future use.

This function is only valid for ps3000a, ps4000a and ps5000a.

set_data_buffer(*channel*, *buffer=None*, *mode='None'*, *segment_index=0*)

Set the data buffer for the specified channel.

The *mode* argument is ignored for ps4000 and ps5000. The *segment_index* argument is ignored for ps4000, ps5000 and ps6000.

Parameters

- **channel** (enum.IntEnum) – An enum value or member name from Channel.
- **buffer** (numpy.ndarray or None) – A int16, numpy array. If None then use a pre-allocated array.
- **mode** (enum.IntEnum) – An enum value or member name from RatioMode.
- **segment_index** (int) – The zero-based number of the memory segment where the data is stored.

set_data_buffer_bulk(*channel*, *buffer*, *waveform*, *mode='None'*)

This function allows you to associate a buffer with a specified waveform number and input

channel in rapid block mode. The number of waveforms captured is determined by the `nCaptures` argument sent to *`set_no_of_captures()`*. There is only one buffer for each waveform because the only down-sampling mode that requires two buffers, aggregation mode, is not available in rapid block mode. Call one of the GetValues functions to retrieve the data after capturing.

This function is only valid for ps4000, ps5000 and ps6000.

The *down_sample_ratio_mode* argument is ignored for ps4000 and ps5000.

set_data_buffers (*channel*, *buffer_length*, *mode*, *segment_index*)
This function tells the driver where to store the data, either unprocessed or down sampled, that will be returned after the next call to one of the GetValues functions. The function allows you to specify only a single buffer, so for aggregation mode, which requires two buffers, you need to call *`set_data_buffers()`* instead.

You must allocate memory for the buffer before calling this function.

The *mode* argument is ignored for ps4000 and ps5000.

The *segment_index* argument is ignored for ps4000, ps5000 and ps6000.

set_digital_port (*port*, *enabled*, *logic_level*)
This function is used to enable the digital port and set the logic level (the voltage at which the state transitions from 0 to 1).

This function is only used by ps2000a and ps3000a.

set_ets (*mode*, *ets_cycles*, *ets_interleave*)
This function is used to enable or disable ETS (equivalent-time sampling) and to set the ETS parameters. See ETS overview for an explanation of ETS mode.

set_ets_time_buffer (*buffer*)
This function tells the driver where to find your applications ETS time buffers. These buffers contain the 64-bit timing information for each ETS sample after you run a block-mode ETS capture.

Parameters **buffer** (*`ctypes array of c_int64`*) – An array of 64-bit words, each representing the time in picoseconds at which the sample was captured.

set_ets_time_buffers (*time_upper*, *time_lower*)
This function tells the driver where to find your applications ETS time buffers. These buffers contain the timing information for each ETS sample after you run a blockmode ETS capture. There are two buffers containing the upper and lower 32-bit parts of the timing information, to allow programming languages that do not support 64-bit data to retrieve the timings.

set_frequency_counter (*channel*, *enabled*, *range*, *threshold_major*, *threshold_minor*)
This function is only define in the header file and it is not in the manual. This function is only valid for ps4000 and ps4000a.

set_no_of_captures (*n_captures*)
This function sets the number of captures to be collected in one run of rapid block mode. If you do not call this function before a run, the driver will capture only one waveform. Once a value has been set, the value remains constant unless changed.

set_sig_gen_arbitrary(*waveform*, *sample_frequency=None*, *offset_voltage=0.0*, *pk_to_pk=None*, *start_delta_phase=None*, *stop_delta_phase=None*, *delta_phase_increment=0*, *dwell_count=None*, *sweep_type='up'*, *operation='off'*, *index_mode='single'*, *shots=None*, *sweeps=None*, *trigger_type='rising'*, *trigger_source='None'*, *ext_in_threshold=0*)

This function programs the signal generator to produce an arbitrary waveform.

set_sig_gen_built_in(*offset_voltage*, *pk_to_pk*, *wave_type*, *start_frequency*, *stop_frequency*, *increment*, *dwell_time*, *sweep_type*, *operation*, *shots*, *sweeps*, *trigger_type*, *trigger_source*, *ext_in_threshold*)

This function sets up the signal generator to produce a signal from a list of built-in waveforms. If different start and stop frequencies are specified, the device will sweep either up, down or up and down.

set_sig_gen_built_in_v2(*offset_voltage=0.0*, *pk_to_pk=1.0*, *wave_type='sine'*, *start_frequency=1.0*, *stop_frequency=None*, *increment=0.1*, *dwell_time=1.0*, *sweep_type='up'*, *operation='off'*, *shots=None*, *sweeps=None*, *trigger_type='rising'*, *trigger_source='None'*, *ext_in_threshold=0*)

This function is an upgraded version of *set_sig_gen_built_in()* with double-precision frequency arguments for more precise control at low frequencies.

This function is invalid for ps4000 and ps4000a.

Parameters

- **offset_voltage** (float) – The voltage offset, in **volts**, to be applied to the waveform.
- **pk_to_pk** (float) – The peak-to-peak voltage, in **volts**, of the waveform signal.
- **wave_type** (enum.IntEnum) – The type of waveform to be generated. A WaveType enum.
- **start_frequency** (float) – The frequency that the signal generator will initially produce.
- **stop_frequency** (float) – The frequency at which the sweep reverses direction or returns to the initial frequency.
- **increment** (float) – The amount of frequency increase or decrease in sweep mode.
- **dwell_time** (float) – The time, in seconds, for which the sweep stays at each frequency.
- **sweep_type** (enum.IntEnum) – Whether the frequency will sweep from *start_frequency* to *stop_frequency*, or in the opposite direction, or repeatedly reverse direction. One of: UP, DOWN, UPDOWN, DOWNUP
- **operation** (enum.IntEnum) – The type of waveform to be produced, specified by one of the following enumerated types (B models only): OFF, WHITENOISE, PRBS

- **shots** (int of None) – If None then start and run continuously after trigger occurs.
- **sweeps** (int of None) – If None then start a sweep and continue after trigger occurs.
- **trigger_type** (enum.IntEnum) – The type of trigger that will be applied to the signal generator. One of: RISING, FALLING, GATE_HIGH, GATE_LOW.
- **trigger_source** (enum.IntEnum or None) – The source that will trigger the signal generator. If None then run without waiting for trigger.
- **ext_in_threshold** (int) – Used to set trigger level for external trigger.

set_sig_gen_properties_arbitrary(*start_delta_phase*, *stop_delta_phase*, *delta_phase_increment*, *dwell_count*, *sweep_type*, *shots*, *sweeps*, *trigger_type*, *trigger_source*, *ext_in_threshold*, *offset_voltage=0*, *pk_to_pk=-1*)

This function reprograms the arbitrary waveform generator. All values can be reprogrammed while the signal generator is waiting for a trigger.

The *offset_voltage* and *pk_to_pk* arguments are only used for ps6000.

This function is invalid for ps4000 and ps5000.

set_sig_gen_properties_built_in(*start_frequency*, *stop_frequency*, *increment*, *dwell_time*, *sweep_type*, *shots*, *sweeps*, *trigger_type*, *trigger_source*, *ext_in_threshold*, *offset_voltage=0*, *pk_to_pk=-1*)

This function reprograms the signal generator. Values can be changed while the signal generator is waiting for a trigger.

The *offset_voltage* and *pk_to_pk* arguments are only used for ps6000.

This function is invalid for ps4000 and ps5000.

set_simple_trigger(*enable*, *source*, *threshold*, *direction*, *delay*, *auto_trigger_ms*)

This function simplifies arming the trigger. It supports only the LEVEL trigger types and does not allow more than one channel to have a trigger applied to it. Any previous pulse width qualifier is cancelled.

set_trigger_channel_directions(*a='rising'*, *b='rising'*, *c='rising'*, *d='rising'*, *ext='rising'*, *aux='rising'*)

This function sets the direction of the trigger for each channel.

ps4000a overrides this method because it has a different implementation.

set_trigger_delay(*delay*)

This function sets the post-trigger delay, which causes capture to start a defined time after the trigger event.

sig_gen_arbitrary_min_max_values()

This function returns the range of possible sample values and waveform buffer sizes that can be supplied to `set_sign_gen_arbitrary()` for setting up the arbitrary waveform generator (AWG).

sig_gen_frequency_to_phase (*frequency*, *index_mode*, *buffer_length*)
This function converts a frequency to a phase count for use with the arbitrary waveform generator (AWG). The value returned depends on the length of the buffer, the index mode passed and the device model. The phase count can then be sent to the driver through *set_sig_gen_arbitrary()* or *set_sig_gen_properties_arbitrary()*.

Parameters

- **frequency** (float) – The AWG sample frequency.
- **index_mode** (enum.IntEnum) – An IndexMode enum value or member name.
- **buffer_length** (int) – The size (number of samples) of the waveform.

Returns int – The phase count.

sig_gen_software_control (*state*)
This function causes a trigger event, or starts and stops gating. It is used when the signal generator is set to SOFT_TRIG.

trigger_within_pre_trigger_samples (*state*)
This function is in the header file, but it is not in the manual.

This function is only valid for ps4000 and ps5000a.

set_trigger_channel_conditions (*conditions*, *info='clear'*)
This function sets up trigger conditions on the scope's inputs. The trigger is defined by one or more picoscope_structs TriggerConditions structures that are then ORed together. Each structure is itself the AND of the states of one or more of the inputs. This AND-OR logic allows you to create any possible Boolean function of the scope's inputs.

The *info* parameter is only used for ps4000a and it is a PS4000AConditionsInfo enum.

set_trigger_channel_properties (*channel_properties*, *timeout=0.1*, *aux_output_enable=0*)
This function is used to enable or disable triggering and set its parameters.

Parameters

- **channel_properties** (list of picoscope_structs) – A list of TriggerChannelProperties structures describing the requested properties.
- **timeout** (float) – The time, in seconds, for which the scope device will wait before collecting data if no trigger event occurs. If this is set to zero, the scope device will wait indefinitely for a trigger.
- **aux_output_enable** (int) – Zero configures the AUXIO connector as a trigger input. Any other value configures it as a trigger output. Only used by ps5000.

msl.equipment.resources.picotech.picoscope.ps2000 module

A wrapper around the PicoScope ps2000 SDK.

class msl.equipment.resources.picotech.picoscope.ps2000.**PicoScope2000** (*record*)
Bases: *msl.equipment.resources.picotech.picoscope.picoscope_2k3k.PicoScope2k3k*

A wrapper around the PicoScope ps2000 SDK.

Parameters **record** (*EquipmentRecord*) – An equipment record from an **Equipment-Register** *Database*.

FIRST_USB = 1

LAST_USB = 127

MAX_UNITS = 127

MAX_TIMEBASE = 19

PS2105_MAX_TIMEBASE = 20

PS2104_MAX_TIMEBASE = 19

PS2200_MAX_TIMEBASE = 23

MAX_OVERSAMPLE = 256

PS2105_MAX_ETS_CYCLES = 250

PS2105_MAX_ETS_INTERLEAVE = 50

PS2104_MAX_ETS_CYCLES = 125

PS2104_MAX_ETS_INTERLEAVE = 25

PS2203_MAX_ETS_CYCLES = 250

PS2203_MAX_ETS_INTERLEAVE = 50

PS2204_MAX_ETS_CYCLES = 250

PS2204_MAX_ETS_INTERLEAVE = 40

PS2205_MAX_ETS_CYCLES = 250

PS2205_MAX_ETS_INTERLEAVE = 40

MIN_ETS_CYCLES_INTERLEAVE_RATIO = 1

MAX_ETS_CYCLES_INTERLEAVE_RATIO = 10

MIN_SIGGEN_FREQ = 0.0

MAX_SIGGEN_FREQ = 100000.0

MAX_VALUE = 32767

MIN_VALUE = -32767

LOST_DATA = -32768

last_button_press()
This function returns the last registered state of the pushbutton on the PicoScope 2104 or 2105 PC Oscilloscope and then resets the status to zero.

set_led(*state*)
This function turns the LED on the oscilloscope on and off, and controls its colour.

set_light(*state*)
This function controls the white light that illuminates the probe tip on a handheld oscilloscope.

set_sig_gen_arbitrary(*offset_voltage*, *pk_to_pk*, *start_delta_phase*, *stop_delta_phase*, *delta_phase_increment*, *dwell_count*, *arbitrary_waveform_size*, *sweep_type*, *sweeps*)

This function programs the signal generator to produce an arbitrary waveform.

set_sig_gen_built_in(*offset_voltage*, *pk_to_pk*, *wave_type*, *start_frequency*, *stop_frequency*, *increment*, *dwell_time*, *sweep_type*, *sweeps*)

This function sets up the signal generator to produce a signal from a list of built-in waveforms. If different start and stop frequencies are specified, the oscilloscope will sweep either up, down or up and down.

msl.equipment.resources.picotech.picoscope.ps2000a module

A wrapper around the PicoScope ps2000a SDK.

class msl.equipment.resources.picotech.picoscope.ps2000a.**PicoScope2000A**(*record*)

Bases: *msl.equipment.resources.picotech.picoscope.picoscope_api.PicoScopeApi*

A wrapper around the PicoScope ps2000a SDK.

Parameters **record** (*EquipmentRecord*) – An equipment record from an **Equipment-Register** *Database*.

PS2208_MAX_ETS_CYCLES = 500

PS2208_MAX_INTERLEAVE = 20

PS2207_MAX_ETS_CYCLES = 500

PS2207_MAX_INTERLEAVE = 20

PS2206_MAX_ETS_CYCLES = 250

PS2206_MAX_INTERLEAVE = 10

EXT_MAX_VALUE = 32767

EXT_MIN_VALUE = -32767

MAX_LOGIC_LEVEL = 32767

MIN_LOGIC_LEVEL = -32767

MIN_SIG_GEN_FREQ = 0.0

MAX_SIG_GEN_FREQ = 20000000.0

MAX_SIG_GEN_BUFFER_SIZE = 8192

MIN_SIG_GEN_BUFFER_SIZE = 1

MIN_DWELL_COUNT = 3

MAX_SWEEPS_SHOTS = 1073741823

MAX_ANALOGUE_OFFSET_50MV_200MV = 0.25

MIN_ANALOGUE_OFFSET_50MV_200MV = -0.25

MAX_ANALOGUE_OFFSET_500MV_2V = 2.5

MIN_ANALOGUE_OFFSET_500MV_2V = -2.5

MAX_ANALOGUE_OFFSET_5V_20V = 20.0

MIN_ANALOGUE_OFFSET_5V_20V = -20.0

SHOT_SWEEP_TRIGGER_CONTINUOUS_RUN = 4294967295

SINE_MAX_FREQUENCY = 1000000.0

SQUARE_MAX_FREQUENCY = 1000000.0

TRIANGLE_MAX_FREQUENCY = 1000000.0

SINC_MAX_FREQUENCY = 1000000.0

RAMP_MAX_FREQUENCY = 1000000.0

HALF_SINE_MAX_FREQUENCY = 1000000.0

GAUSSIAN_MAX_FREQUENCY = 1000000.0

PRBS_MAX_FREQUENCY = 1000000.0

PRBS_MIN_FREQUENCY = 0.03

MIN_FREQUENCY = 0.03

set_digital_analog_trigger_operand(*operand*)
This function is define in the header file, but it is not in the manual.

set_trigger_digital_port_properties(*directions*)
This function will set the individual Digital channels trigger directions. Each trigger direction consists of a channel name and a direction. If the channel is not included in the array of `PS2000ADigitalChannelDirections` the driver assumes the digital channel's trigger direction is `PS2000A_DIGITAL_DONT_CARE`.

msl.equipment.resources.picotech.picoscope.ps3000 module

A wrapper around the PicoScope ps3000 SDK.

class `msl.equipment.resources.picotech.picoscope.ps3000.`**PicoScope3000**(*record*)
Bases: *msl.equipment.resources.picotech.picoscope.picoscope_2k3k.PicoScope2k3k*

A wrapper around the PicoScope ps3000 SDK.

Parameters **record** (*EquipmentRecord*) – An equipment record from an **Equipment-Register** *Database*.

FIRST_USB = 1

LAST_USB = 127

MAX_UNITS = 127

PS3206_MAX_TIMEBASE = 21

PS3205_MAX_TIMEBASE = 20

PS3204_MAX_TIMEBASE = 19

PS3224_MAX_TIMEBASE = 19

PS3223_MAX_TIMEBASE = 19

PS3424_MAX_TIMEBASE = 19

PS3423_MAX_TIMEBASE = 19

PS3225_MAX_TIMEBASE = 18

PS3226_MAX_TIMEBASE = 19

PS3425_MAX_TIMEBASE = 19

PS3426_MAX_TIMEBASE = 19

MAX_OVERSAMPLE = 256

MAX_VALUE = 32767

MIN_VALUE = -32767

LOST_DATA = -32768

MIN_SIGGEN_FREQ = 0.093

MAX_SIGGEN_FREQ = 1000000

PS3206_MAX_ETS_CYCLES = 500

PS3206_MAX_ETS_INTERLEAVE = 100

PS3205_MAX_ETS_CYCLES = 250

PS3205_MAX_ETS_INTERLEAVE = 50

PS3204_MAX_ETS_CYCLES = 125

PS3204_MAX_ETS_INTERLEAVE = 25

MAX_ETS_CYCLES_INTERLEAVE_RATIO = 10

MIN_ETS_CYCLES_INTERLEAVE_RATIO = 1

PS300_MAX_ETS_SAMPLES = 100000

MAX_PULSE_WIDTH_QUALIFIER_COUNT = 16777215

MAX_HOLDOFF_COUNT = 8388607

release_stream_buffer()
Not found in the manual, but it is in the header file.

save_streaming_data(*lp_callback_func*, *data_buffer_size*)
This function sends all available streaming data to the `lp_callback_func` callback function in your application. Your callback function decides what to do with the data.

set_siggen(*wave_type*, *start_frequency*, *stop_frequency*, *increment*, *dwell_time*, *repeat*, *dual_slope*)
This function is used to enable or disable the signal generator and sweep functions.

streaming_ns_get_interval_stateless(*n_channels*)
Not found in the manual, but it is in the header file.

msl.equipment.resources.picotech.picoscope.ps3000a module

A wrapper around the PicoScope ps3000a SDK.

class msl.equipment.resources.picotech.picoscope.ps3000a.**PicoScope3000A**(*record*)

Bases: *msl.equipment.resources.picotech.picoscope.picoscope_api.PicoScopeApi*

A wrapper around the PicoScope ps3000a SDK.

Parameters **record** (*EquipmentRecord*) – An equipment record from an **Equipment-Register** *Database*.

MAX_OVERSAMPLE = 256

PS3207A_MAX_ETS_CYCLES = 500

PS3207A_MAX_INTERLEAVE = 40

PS3206A_MAX_ETS_CYCLES = 500

PS3206A_MAX_INTERLEAVE = 40

PS3206MSO_MAX_INTERLEAVE = 80

PS3205A_MAX_ETS_CYCLES = 250

PS3205A_MAX_INTERLEAVE = 20

PS3205MSO_MAX_INTERLEAVE = 40

PS3204A_MAX_ETS_CYCLES = 125

PS3204A_MAX_INTERLEAVE = 10

PS3204MSO_MAX_INTERLEAVE = 20

EXT_MAX_VALUE = 32767

EXT_MIN_VALUE = -32767

MAX_LOGIC_LEVEL = 32767

MIN_LOGIC_LEVEL = -32767

MIN_SIG_GEN_FREQ = 0.0

MAX_SIG_GEN_FREQ = 20000000.0

PS3207B_MAX_SIG_GEN_BUFFER_SIZE = 32768

PS3206B_MAX_SIG_GEN_BUFFER_SIZE = 16384

MAX_SIG_GEN_BUFFER_SIZE = 8192

MIN_SIG_GEN_BUFFER_SIZE = 1

MIN_DWELL_COUNT = 3

MAX_SWEEPS_SHOTS = 1073741823

MAX_ANALOGUE_OFFSET_50MV_200MV = 0.25

MIN_ANALOGUE_OFFSET_50MV_200MV = -0.25

MAX_ANALOGUE_OFFSET_500MV_2V = 2.5

MIN_ANALOGUE_OFFSET_500MV_2V = -2.5

MAX_ANALOGUE_OFFSET_5V_20V = 20.0

MIN_ANALOGUE_OFFSET_5V_20V = -20.0

SHOT_SWEEP_TRIGGER_CONTINUOUS_RUN = 4294967295

SINE_MAX_FREQUENCY = 1000000.0

SQUARE_MAX_FREQUENCY = 1000000.0

TRIANGLE_MAX_FREQUENCY = 1000000.0

SINC_MAX_FREQUENCY = 1000000.0

RAMP_MAX_FREQUENCY = 1000000.0

HALF_SINE_MAX_FREQUENCY = 1000000.0

GAUSSIAN_MAX_FREQUENCY = 1000000.0

PRBS_MAX_FREQUENCY = 1000000.0

PRBS_MIN_FREQUENCY = 0.03

MIN_FREQUENCY = 0.03

get_max_ets_values()
This function returns the maximum number of cycles and maximum interleaving factor that can be used for the selected scope device in ETS mode. These values are the upper limits for the `etsCycles` and `etsInterleave` arguments supplied to `set_ets()`.

get_trigger_info_bulk(*from_segment_index*, *to_segment_index*)
This function returns trigger information in rapid block mode.

Populates the `PS3000ATriggerInfo` structure.

set_pulse_width_digital_port_properties(*directions*)
This function will set the individual digital channels' pulse-width trigger directions. Each trigger direction consists of a channel name and a direction. If the channel is not included in the array of `PS3000ADigitalChannelDirections` the driver assumes the digital channel's pulse-width trigger direction is `PS3000A_DIGITAL_DONT_CARE`.

set_pulse_width_qualifier_v2(*conditions*, *direction*, *lower*, *upper*, *pulse_width_type*)
This function sets up pulse-width qualification, which can be used on its own for pulse width triggering or combined with level triggering or window triggering to produce more complex triggers. The pulse-width qualifier is set by defining one or more structures that are then ORed together. Each structure is itself the AND of the states of one or more of the inputs. This AND-OR logic allows you to create any possible Boolean function of the scope's inputs.

set_trigger_channel_conditions_v2(*conditions*)
This function sets up trigger conditions on the scope's inputs. The trigger is defined by one or more `PS3000ATriggerConditionsV2` structures that are then ORed together. Each structure is itself the AND of the states of one or more of the inputs. This AND-OR logic allows you to create any possible Boolean function of the scope's inputs.

If complex triggering is not required, use `set_simple_trigger()`.

set_trigger_digital_port_properties(*directions*)
This function will set the individual digital channels' trigger directions. Each trigger direction consists of a channel name and a direction. If the channel is not included in the array of `PS3000ADigitalChannelDirections` the driver assumes the digital channel's trigger direction is PS3000A_DIGITAL_DONT_CARE.

msl.equipment.resources.picotech.picoscope.ps4000 module

A wrapper around the PicoScope ps4000 SDK.

class msl.equipment.resources.picotech.picoscope.ps4000.**PicoScope4000**(*record*)
Bases: *msl.equipment.resources.picotech.picoscope.picoscope_api.PicoScopeApi*

A wrapper around the PicoScope ps4000 SDK.

Parameters **record** (*EquipmentRecord*) – An equipment record from an **Equipment-Register** *Database*.

MAX_OVERSAMPLE_12BIT = 16

MAX_OVERSAMPLE_8BIT = 256

PS4XXX_MAX_ETS_CYCLES = 400

PS4XXX_MAX_INTERLEAVE = 80

PS4262_MAX_VALUE = 32767

PS4262_MIN_VALUE = -32767

MAX_VALUE = 32764

MIN_VALUE = -32764

LOST_DATA = -32768

EXT_MAX_VALUE = 32767

EXT_MIN_VALUE = -32767

MAX_PULSE_WIDTH_QUALIFIER_COUNT = 16777215

MAX_DELAY_COUNT = 8388607

MIN_SIG_GEN_FREQ = 0.0

MAX_SIG_GEN_FREQ = 100000.0

MAX_SIG_GEN_FREQ_4262 = 20000.0

MIN_SIG_GEN_BUFFER_SIZE = 1

MAX_SIG_GEN_BUFFER_SIZE = 8192

MIN_DWELL_COUNT = 10

PS4262_MAX_WAVEFORM_BUFFER_SIZE = 4096

PS4262_MIN_DWELL_COUNT = 3

MAX_SWEEPS_SHOTS = 1073741823

get_probe()
This function is in the header file, but it is not in the manual.

get_trigger_channel_time_offset(*segment_index*, *channel*)
This function gets the time, as two 4-byte values, at which the trigger occurred, adjusted for the time skew of the specified channel relative to the trigger source. Call it after block-mode data has been captured or when data has been retrieved from a previous block-mode capture.

get_trigger_channel_time_offset64(*segment_index*, *channel*)
This function gets the time, as a single 8-byte value, at which the trigger occurred, adjusted for the time skew of the specified channel relative to the trigger source. Call it after block-mode data has been captured or when data has been retrieved from a previous block-mode capture.

get_values_trigger_channel_time_offset_bulk(*from_segment_index*, *to_segment_index*, *channel*)
This function retrieves the time offset, as lower and upper 32-bit values, for a group of waveforms obtained in rapid block mode, adjusted for the time skew relative to the trigger source. The array size for `timesUpper` and `timesLower` must be greater than or equal to the number of waveform time offsets requested. The segment indexes are inclusive.

get_values_trigger_channel_time_offset_bulk64(*from_segment_index*, *to_segment_index*, *channel*)
This function retrieves the time offset, as a 64-bit integer, for a group of waveforms captured in rapid block mode, adjusted for the time skew relative to the trigger source. The array size of `times` must be greater than or equal to the number of waveform time offsets requested. The segment indexes are inclusive.

open_unit_async_ex()
This function opens a scope device selected by serial number without blocking the calling thread. You can find out when it has finished by periodically calling `open_unit_progress()` until that function returns a non-zero value.

open_unit_ex()
This function opens a scope device. The maximum number of units that can be opened is determined by the operating system, the kernel driver and the PC's hardware.

run_streaming_ex(*sample_interval_time_units*, *max_pre_trigger_samples*, *max_post_pre_trigger_samples*, *auto_stop*, *down_sample_ratio*, *down_sample_ratio_mode*, *overview_buffer_size*)
This function tells the oscilloscope to start collecting data in streaming mode and with a specified data reduction mode. When data has been collected from the device it is aggregated and the values returned to the application. Call `get_streaming_latest_values()` to retrieve the data.

set_bw_filter(*channel*, *enable*)
This function enables or disables the bandwidth-limiting filter on the specified channel.

set_data_buffer_with_mode(*channel*, *buffer_length*, *mode*)
This function registers your data buffer, for non-aggregated data, with the PicoScope 4000 driver. You need to allocate the buffer before calling this function.

set_data_buffers_with_mode(*channel*, *buffer_length*, *mode*)
This function registers your data buffers, for receiving aggregated data, with the PicoScope 4000 driver. You need to allocate memory for the buffers before calling this function.

set_ext_trigger_range(*ext_range*)
This function sets the range of the external trigger.

set_probe(*probe*, *range_*)
This function is in the header file, but it is not in the manual.

msl.equipment.resources.picotech.picoscope.ps4000a module

A wrapper around the PicoScope ps4000a SDK.

class msl.equipment.resources.picotech.picoscope.ps4000a.**PicoScope4000A**(*record*)
Bases: *msl.equipment.resources.picotech.picoscope.picoscope_api.PicoScopeApi*

A wrapper around the PicoScope ps4000a SDK.

Parameters **record** (*EquipmentRecord*) – An equipment record from an **Equipment-Register** *Database*.

MAX_VALUE = 32767

MIN_VALUE = -32767

LOST_DATA = -32768

EXT_MAX_VALUE = 32767

EXT_MIN_VALUE = -32767

MAX_PULSE_WIDTH_QUALIFIER_COUNT = 16777215

MAX_DELAY_COUNT = 8388607

MAX_SIG_GEN_BUFFER_SIZE = 16384

MIN_SIG_GEN_BUFFER_SIZE = 10

MIN_DWELL_COUNT = 10

MAX_SWEEPS_SHOTS = 1073741823

AWG_DAC_FREQUENCY = 80000000.0

AWG_PHASE_ACCUMULATOR = 4294967296.0

MAX_ANALOGUE_OFFSET_50MV_200MV = 0.25

MIN_ANALOGUE_OFFSET_50MV_200MV = -0.25

MAX_ANALOGUE_OFFSET_500MV_2V = 2.5

MIN_ANALOGUE_OFFSET_500MV_2V = -2.5

MAX_ANALOGUE_OFFSET_5V_20V = 20.0

MIN_ANALOGUE_OFFSET_5V_20V = -20.0

SINE_MAX_FREQUENCY = 1000000.0

SQUARE_MAX_FREQUENCY = 1000000.0

TRIANGLE_MAX_FREQUENCY = 1000000.0

SINC_MAX_FREQUENCY = 1000000.0

RAMP_MAX_FREQUENCY = 1000000.0

HALF_SINE_MAX_FREQUENCY = 1000000.0

GAUSSIAN_MAX_FREQUENCY = 1000000.0

MIN_FREQUENCY = 0.03

apply_resistance_scaling(*channel*, *range_*, *buffer_length*)
This function is in the header file, but it is not in the manual.

connect_detect(*sensors*)
This function is in the header file, but it is not in the manual.

device_meta_data(*meta_type*, *operation*, *format_*)
This function is in the header file, but it is not in the manual.

get_common_mode_overflow()
This function is in the header file, but it is not in the manual.

get_string(*string_value*)
This function is in the header file, but it is not in the manual.

set_channel_led(*led_states*)
This function is in the header file, but it is not in the manual.

set_pulse_width_qualifier_conditions(*conditions*, *info*)
This function sets up the conditions for pulse width qualification, which can be used on its own for pulse width triggering or combined with window triggering to produce more complex triggers. Each call to this function creates a pulse width qualifier equal to the logical AND of the elements of the `conditions` array. Calling this function multiple times creates the logical OR of multiple AND operations. This AND-OR logic allows you to create any possible Boolean function of the scope's inputs.

Other settings of the pulse width qualifier are configured by calling *set_pulse_width_qualifier_properties()*.

set_pulse_width_qualifier_properties(*direction*, *lower*, *upper*, *pulse_width_type*)
This function configures the general properties of the pulse width qualifier.

set_trigger_channel_directions(*directions*)
This function sets the direction of the trigger for the specified channels.

msl.equipment.resources.picotech.picoscope.ps5000 module

A wrapper around the PicoScope ps5000 SDK.

class msl.equipment.resources.picotech.picoscope.ps5000.**PicoScope5000**(*record*)
Bases: *msl.equipment.resources.picotech.picoscope.picoscope_api.PicoScopeApi*

A wrapper around the PicoScope ps5000 SDK.

Parameters **record** (*EquipmentRecord*) – An equipment record from an **Equipment-Register** *Database*.

MAX_OVERSAMPLE_8BIT = 256

MAX_VALUE = 32512

MIN_VALUE = -32512

LOST_DATA = -32768

EXT_MAX_VALUE = 32767

EXT_MIN_VALUE = -32767

MAX_PULSE_WIDTH_QUALIFIER_COUNT = 16777215

MAX_DELAY_COUNT = 8388607

MAX_SIG_GEN_BUFFER_SIZE = 8192

MIN_SIG_GEN_BUFFER_SIZE = 10

MIN_DWELL_COUNT = 10

MAX_SWEEPS_SHOTS = 1073741823

SINE_MAX_FREQUENCY = 20000000.0

SQUARE_MAX_FREQUENCY = 20000000.0

TRIANGLE_MAX_FREQUENCY = 20000000.0

SINC_MAX_FREQUENCY = 20000000.0

RAMP_MAX_FREQUENCY = 20000000.0

HALF_SINE_MAX_FREQUENCY = 20000000.0

GAUSSIAN_MAX_FREQUENCY = 20000000.0

MIN_FREQUENCY = 0.03

msl.equipment.resources.picotech.picoscope.ps5000a module

A wrapper around the PicoScope ps5000a SDK.

class msl.equipment.resources.picotech.picoscope.ps5000a.**PicoScope5000A**(*record*)

Bases: *msl.equipment.resources.picotech.picoscope.picoscope_api.PicoScopeApi*

A wrapper around the PicoScope ps5000a SDK.

Parameters **record** (*EquipmentRecord*) – An equipment record from an **Equipment-Register** *Database*.

MAX_VALUE_8BIT = 32512

MIN_VALUE_8BIT = -32512

MAX_VALUE_16BIT = 32767

MIN_VALUE_16BIT = -32767

LOST_DATA = -32768

EXT_MAX_VALUE = 32767

EXT_MIN_VALUE = -32767

MAX_PULSE_WIDTH_QUALIFIER_COUNT = 16777215

MAX_DELAY_COUNT = 8388607

PS5X42A_MAX_SIG_GEN_BUFFER_SIZE = 16384

PS5X43A_MAX_SIG_GEN_BUFFER_SIZE = 32768

PS5X44A_MAX_SIG_GEN_BUFFER_SIZE = 49152

MIN_SIG_GEN_BUFFER_SIZE = 1

MIN_DWELL_COUNT = 3

MAX_SWEEPS_SHOTS = 1073741823

AWG_DAC_FREQUENCY = 200000000.0

AWG_PHASE_ACCUMULATOR = 4294967296.0

MAX_ANALOGUE_OFFSET_50MV_200MV = 0.25

MIN_ANALOGUE_OFFSET_50MV_200MV = -0.25

MAX_ANALOGUE_OFFSET_500MV_2V = 2.5

MIN_ANALOGUE_OFFSET_500MV_2V = -2.5

MAX_ANALOGUE_OFFSET_5V_20V = 20.0

MIN_ANALOGUE_OFFSET_5V_20V = -20.0

PS5244A_MAX_ETS_CYCLES = 500

PS5244A_MAX_ETS_INTERLEAVE = 40

PS5243A_MAX_ETS_CYCLES = 250

PS5243A_MAX_ETS_INTERLEAVE = 20

PS5242A_MAX_ETS_CYCLES = 125

PS5242A_MAX_ETS_INTERLEAVE = 10

SHOT_SWEEP_TRIGGER_CONTINUOUS_RUN = 4294967295

SINE_MAX_FREQUENCY = 20000000.0

SQUARE_MAX_FREQUENCY = 20000000.0

TRIANGLE_MAX_FREQUENCY = 20000000.0

SINC_MAX_FREQUENCY = 20000000.0

RAMP_MAX_FREQUENCY = 20000000.0

HALF_SINE_MAX_FREQUENCY = 20000000.0

GAUSSIAN_MAX_FREQUENCY = 20000000.0

MIN_FREQUENCY = 0.03

EXT_MAX_VOLTAGE = 5.0

get_device_resolution()

Returns enum.IntEnum – This function retrieves the resolution the specified device will run in.

get_trigger_info_bulk(*from_segment_index*, *to_segment_index*)
This function is in the header file, but it is not in the manual.

Populates the `PS5000ATriggerInfo` structure.

set_device_resolution(*resolution*)
This function sets the new resolution. When using 12 bits or more the memory is halved. When using 15-bit resolution only 2 channels can be enabled to capture data, and when using 16-bit resolution only one channel is available. If resolution is changed, any data captured that has not been saved will be lost. If `set_channel()` is not called, `run_block()` and `run_streaming()` may fail.

msl.equipment.resources.picotech.picoscope.ps6000 module

A wrapper around the PicoScope ps6000 SDK.

class msl.equipment.resources.picotech.picoscope.ps6000.**PicoScope6000**(*record*)
Bases: *msl.equipment.resources.picotech.picoscope.picoscope_api.PicoScopeApi*

A wrapper around the PicoScope ps6000 SDK.

Parameters **record** (*EquipmentRecord*) – An equipment record from an **Equipment-Register** *Database*.

MAX_OVERSAMPLE_8BIT = 256

MAX_VALUE = 32512

MIN_VALUE = -32512

MAX_PULSE_WIDTH_QUALIFIER_COUNT = 16777215

MAX_SIG_GEN_BUFFER_SIZE = 16384

PS640X_C_D_MAX_SIG_GEN_BUFFER_SIZE = 65536

MIN_SIG_GEN_BUFFER_SIZE = 1

MIN_DWELL_COUNT = 3

MAX_SWEEPS_SHOTS = 1073741823

MAX_WAVEFORMS_PER_SECOND = 1000000

MAX_ANALOGUE_OFFSET_50MV_200MV = 0.5

MIN_ANALOGUE_OFFSET_50MV_200MV = -0.5

MAX_ANALOGUE_OFFSET_500MV_2V = 2.5

MIN_ANALOGUE_OFFSET_500MV_2V = -2.5

MAX_ANALOGUE_OFFSET_5V_20V = 20.0

MIN_ANALOGUE_OFFSET_5V_20V = -20.0

MAX_ETS_CYCLES = 250

MAX_INTERLEAVE = 50

PRBS_MAX_FREQUENCY = 20000000.0

SINE_MAX_FREQUENCY = 20000000.0

SQUARE_MAX_FREQUENCY = 20000000.0

TRIANGLE_MAX_FREQUENCY = 20000000.0

SINC_MAX_FREQUENCY = 20000000.0

RAMP_MAX_FREQUENCY = 20000000.0

HALF_SINE_MAX_FREQUENCY = 20000000.0

GAUSSIAN_MAX_FREQUENCY = 20000000.0

MIN_FREQUENCY = 0.03

get_values_bulk_asyc(*start_index*, *down_sample_ratio*, *down_sample_ratio_mode*, *from_segment_index*, *to_segment_index*)
This function is in the header file, but it is not in the manual.

set_data_buffers_bulk(*channel*, *buffer_length*, *waveform*, *down_sample_ratio_mode*)
This function tells the driver where to find the buffers for aggregated data for each waveform in rapid block mode. The number of waveforms captured is determined by the `nCaptures` argument sent to `set_no_of_captures()`. Call one of the GetValues functions to retrieve the data after capture. If you do not need two buffers, because you are not using aggregate mode, then you can optionally use `set_data_buffer_bulk()` instead.

set_external_clock(*frequency*, *threshold*)
This function tells the scope whether or not to use an external clock signal fed into the AUX input. The external clock can be used to synchronise one or more PicoScope 6000 units to an external source.

When the external clock input is enabled, the oscilloscope relies on the clock signal for all of its timing. The driver checks that the clock is running before starting a capture, but if the clock signal stops after the initial check, the oscilloscope will not respond to any further commands until it is powered down and back up again.

Note: if the AUX input is set as an external clock input then it cannot also be used as an external trigger input.

set_waveform_limiter(*n_waveforms_per_second*)
This function is in the header file, but it is not in the manual.

msl.equipment.resources.picotech.picoscope.structs module

Structures defined in the Pico Technology SDK v10.6.10.24

class `msl.equipment.resources.picotech.picoscope.structs.`**PS2000TriggerChannelPrope**
Bases: `_ctypes.Structure`

channel
Structure/Union member

hysteresis
Structure/Union member

thresholdMajor
Structure/Union member

thresholdMinor
Structure/Union member

thresholdMode
Structure/Union member

class msl.equipment.resources.picotech.picoscope.structs.**PS2000TriggerConditions**
Bases: _ctypes.Structure

channelA
Structure/Union member

channelB
Structure/Union member

channelC
Structure/Union member

channelD
Structure/Union member

external
Structure/Union member

pulseWidthQualifier
Structure/Union member

class msl.equipment.resources.picotech.picoscope.structs.**PS2000PwqConditions**
Bases: _ctypes.Structure

channelA
Structure/Union member

channelB
Structure/Union member

channelC
Structure/Union member

channelD
Structure/Union member

external
Structure/Union member

class msl.equipment.resources.picotech.picoscope.structs.**PS2000ATriggerConditions**
Bases: _ctypes.Structure

aux
Structure/Union member

channelA
Structure/Union member

channelB
Structure/Union member

channelC
Structure/Union member

channelD
Structure/Union member

digital
Structure/Union member

external
Structure/Union member

pulseWidthQualifier
Structure/Union member

class msl.equipment.resources.picotech.picoscope.structs.**PS2000APwqConditions**
Bases: _ctypes.Structure

aux
Structure/Union member

channelA
Structure/Union member

channelB
Structure/Union member

channelC
Structure/Union member

channelD
Structure/Union member

digital
Structure/Union member

external
Structure/Union member

class msl.equipment.resources.picotech.picoscope.structs.**PS2000ADigitalChannelDire**
Bases: _ctypes.Structure

channel
Structure/Union member

direction
Structure/Union member

class msl.equipment.resources.picotech.picoscope.structs.**PS2000ATriggerChannelProp**
Bases: _ctypes.Structure

channel
Structure/Union member

thresholdLower
Structure/Union member

thresholdLowerHysteresis
Structure/Union member

thresholdMode
Structure/Union member

thresholdUpper
Structure/Union member

thresholdUpperHysteresis
Structure/Union member

class msl.equipment.resources.picotech.picoscope.structs.**PS3000TriggerChannelPrope**
Bases: _ctypes.Structure

channel
Structure/Union member

hysteresis
Structure/Union member

thresholdMajor
Structure/Union member

thresholdMinor
Structure/Union member

thresholdMode
Structure/Union member

class msl.equipment.resources.picotech.picoscope.structs.**PS3000TriggerConditions**
Bases: _ctypes.Structure

channelA
Structure/Union member

channelB
Structure/Union member

channelC
Structure/Union member

channelD
Structure/Union member

external
Structure/Union member

pulseWidthQualifier
Structure/Union member

class msl.equipment.resources.picotech.picoscope.structs.**PS3000PwqConditions**
Bases: _ctypes.Structure

channelA
Structure/Union member

channelB
Structure/Union member

channelC
Structure/Union member

channelD
Structure/Union member

external
Structure/Union member

class msl.equipment.resources.picotech.picoscope.structs.**PS3000ATriggerConditions**
Bases: _ctypes.Structure

aux
Structure/Union member

channelA
Structure/Union member

channelB
Structure/Union member

channelC
Structure/Union member

channelD
Structure/Union member

external
Structure/Union member

pulseWidthQualifier
Structure/Union member

class msl.equipment.resources.picotech.picoscope.structs.**PS3000ATriggerConditionsV**
Bases: _ctypes.Structure

aux
Structure/Union member

channelA
Structure/Union member

channelB
Structure/Union member

channelC
Structure/Union member

channelD
Structure/Union member

digital
Structure/Union member

external
Structure/Union member

pulseWidthQualifier
Structure/Union member

class msl.equipment.resources.picotech.picoscope.structs.**PS3000APwqConditions**
Bases: _ctypes.Structure

aux
Structure/Union member

channelA
Structure/Union member

channelB
Structure/Union member

channelC
Structure/Union member

channelD
Structure/Union member

external
Structure/Union member

class msl.equipment.resources.picotech.picoscope.structs.**PS3000APwqConditionsV2**
Bases: _ctypes.Structure

aux
Structure/Union member

channelA
Structure/Union member

channelB
Structure/Union member

channelC
Structure/Union member

channelD
Structure/Union member

digital
Structure/Union member

external
Structure/Union member

class msl.equipment.resources.picotech.picoscope.structs.**PS3000ADigitalChannelDire**
Bases: _ctypes.Structure

channel
Structure/Union member

direction
Structure/Union member

class msl.equipment.resources.picotech.picoscope.structs.**PS3000ATriggerChannelProp**
Bases: _ctypes.Structure

channel
Structure/Union member

thresholdLower
Structure/Union member

thresholdLowerHysteresis
Structure/Union member

thresholdMode
Structure/Union member

thresholdUpper
Structure/Union member

thresholdUpperHysteresis
Structure/Union member

class msl.equipment.resources.picotech.picoscope.structs.**PS3000ATriggerInfo**
Bases: _ctypes.Structure

reserved0
Structure/Union member

reserved1
Structure/Union member

segmentIndex
Structure/Union member

status
Structure/Union member

timeStampCounter
Structure/Union member

timeUnits
Structure/Union member

triggerTime
Structure/Union member

class msl.equipment.resources.picotech.picoscope.structs.**PS4000TriggerConditions**
Bases: _ctypes.Structure

aux
Structure/Union member

channelA
Structure/Union member

channelB
Structure/Union member

channelC
Structure/Union member

channelD
Structure/Union member

external
Structure/Union member

pulseWidthQualifier
Structure/Union member

class msl.equipment.resources.picotech.picoscope.structs.**PS4000PwqConditions**
Bases: _ctypes.Structure

aux
Structure/Union member

channelA
Structure/Union member

channelB
Structure/Union member

channelC
Structure/Union member

channelD
Structure/Union member

external
Structure/Union member

class msl.equipment.resources.picotech.picoscope.structs.**PS4000TriggerChannelPrope**
Bases: _ctypes.Structure

channel
Structure/Union member

thresholdLower
Structure/Union member

thresholdLowerHysteresis
Structure/Union member

thresholdMode
Structure/Union member

thresholdUpper
Structure/Union member

thresholdUpperHysteresis
Structure/Union member

class msl.equipment.resources.picotech.picoscope.structs.**PS4000AChannelLedSetting**
Bases: _ctypes.Structure

channel
Structure/Union member

state
Structure/Union member

class msl.equipment.resources.picotech.picoscope.structs.**PS4000ADirection**
Bases: _ctypes.Structure

channel
Structure/Union member

direction
Structure/Union member

class msl.equipment.resources.picotech.picoscope.structs.**PS4000ACondition**
Bases: _ctypes.Structure

condition
Structure/Union member

source
Structure/Union member

class msl.equipment.resources.picotech.picoscope.structs.**PS4000ATriggerChannelProp**
Bases: _ctypes.Structure

channel
Structure/Union member

thresholdLower
Structure/Union member

thresholdLowerHysteresis
Structure/Union member

thresholdMode
Structure/Union member

thresholdUpper
Structure/Union member

thresholdUpperHysteresis
Structure/Union member

class msl.equipment.resources.picotech.picoscope.structs.**PS4000AConnectDetect**
Bases: _ctypes.Structure

channel
Structure/Union member

state
Structure/Union member

class msl.equipment.resources.picotech.picoscope.structs.**PS5000TriggerConditions**
Bases: _ctypes.Structure

aux
Structure/Union member

channelA
Structure/Union member

channelB
Structure/Union member

channelC
Structure/Union member

channelD
Structure/Union member

external
Structure/Union member

pulseWidthQualifier
Structure/Union member

class msl.equipment.resources.picotech.picoscope.structs.**PS5000PwqConditions**
Bases: _ctypes.Structure

aux
Structure/Union member

channelA
Structure/Union member

channelB
Structure/Union member

channelC
Structure/Union member

channelD
Structure/Union member

external
Structure/Union member

class msl.equipment.resources.picotech.picoscope.structs.**PS5000TriggerChannelPrope**
Bases: _ctypes.Structure

channel
Structure/Union member

hysteresis
Structure/Union member

thresholdMajor
Structure/Union member

thresholdMinor
Structure/Union member

thresholdMode
Structure/Union member

class msl.equipment.resources.picotech.picoscope.structs.**PS5000ATriggerInfo**
Bases: _ctypes.Structure

reserved0
Structure/Union member

reserved1
Structure/Union member

segmentIndex
Structure/Union member

status
Structure/Union member

timeUnits
Structure/Union member

triggerIndex
Structure/Union member

triggerTime
Structure/Union member

class msl.equipment.resources.picotech.picoscope.structs.**PS5000ATriggerConditions**
Bases: _ctypes.Structure

aux
Structure/Union member

channelA
Structure/Union member

channelB
Structure/Union member

channelC
Structure/Union member

channelD
Structure/Union member

external
Structure/Union member

pulseWidthQualifier
Structure/Union member

class msl.equipment.resources.picotech.picoscope.structs.**PS5000APwqConditions**
Bases: _ctypes.Structure

aux
Structure/Union member

channelA
Structure/Union member

channelB
Structure/Union member

channelC
Structure/Union member

channelD
Structure/Union member

external
Structure/Union member

class msl.equipment.resources.picotech.picoscope.structs.**PS5000ATriggerChannelProp**
Bases: _ctypes.Structure

channel
Structure/Union member

thresholdLower
Structure/Union member

thresholdLowerHysteresis
Structure/Union member

thresholdMode
Structure/Union member

thresholdUpper
Structure/Union member

thresholdUpperHysteresis
Structure/Union member

class msl.equipment.resources.picotech.picoscope.structs.**PS6000TriggerConditions**
Bases: _ctypes.Structure

aux
Structure/Union member

channelA
Structure/Union member

channelB
Structure/Union member

channelC
Structure/Union member

channelD
Structure/Union member

external
Structure/Union member

pulseWidthQualifier
Structure/Union member

class msl.equipment.resources.picotech.picoscope.structs.**PS6000PwqConditions**
Bases: _ctypes.Structure

aux
Structure/Union member

channelA
Structure/Union member

channelB
Structure/Union member

channelC
Structure/Union member

channelD
Structure/Union member

external
Structure/Union member

class msl.equipment.resources.picotech.picoscope.structs.**PS6000TriggerChannelPrope**
Bases: _ctypes.Structure

channel
Structure/Union member

hysteresisLower
Structure/Union member

hysteresisUpper
Structure/Union member

thresholdLower
Structure/Union member

thresholdMode
Structure/Union member

thresholdUpper
Structure/Union member

msl.equipment.resources.thorlabs package

Wrappers around APIs from Thorlabs.

Subpackages

msl.equipment.resources.thorlabs.kinesis package

Wrapper package around the Thorlabs.MotionControl.C_API.

The Kinesis software can be downloaded from the Thorlabs website

Submodules

msl.equipment.resources.thorlabs.kinesis.api_functions module

C Functions defined in the Thorlabs Kinesis software v1.11.0

msl.equipment.resources.thorlabs.kinesis.callbacks module

msl.equipment.resources.thorlabs.kinesis.callbacks.**MotionControlCallback**
A callback to register for a MotionControl message queue.

Example usage:

```
from msl.examples.equipment import EXAMPLES_DIR
from msl.equipment.database import load
from msl.equipment.resources.thorlabs import MotionControlCallback

@MotionControlCallback
def msg_callback():
    print('MotionControlCallback: ', flipper.convert_message(*flipper.
↪get_next_message()))
```

```
# The "equipment-configuration.xml" configuration file contains the␣
↪following element:
# <equipment alias="filter_flipper" manufacturer="Thorlabs" model=
↪"MFF101/M" serial="37871232"/>

db = load(os.path.join(EXAMPLES_DIR, 'equipment-configuration.xml'))
flipper = db.equipment['filter_flipper'].connect()
flipper.register_message_callback(msg_callback)

... do stuff with the `flipper` ...
```

alias of `CFunctionType`

msl.equipment.resources.thorlabs.kinesis.enums module

Enums defined in the Thorlabs Kinesis software v1.11.0

class msl.equipment.resources.thorlabs.kinesis.enums.**FT_Status**

Bases: enum.IntEnum

An enumeration.

FT_OK = 0

FT_InvalidHandle = 1

FT_DeviceNotFound = 2

FT_DeviceNotOpened = 3

FT_IOError = 4

FT_InsufficientResources = 5

FT_InvalidParameter = 6

FT_DeviceNotPresent = 7

FT_IncorrectDevice = 8

class msl.equipment.resources.thorlabs.kinesis.enums.**MOT_MotorTypes**

Bases: enum.IntEnum

An enumeration.

MOT_NotMotor = 0

MOT_DCMotor = 1

MOT_StepperMotor = 2

MOT_BrushlessMotor = 3

MOT_CustomMotor = 100

class msl.equipment.resources.thorlabs.kinesis.enums.**MOT_TravelModes**

Bases: enum.IntEnum

An enumeration.

MOT_TravelModeUndefined = 0

MOT_Linear = 1

MOT_Rotational = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**MOT_TravelDirection**
Bases: enum.IntEnum

An enumeration.

MOT_TravelDirectionDisabled = 0

MOT_Forwards = 1

MOT_Reverse = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**MOT_DirectionSense**
Bases: enum.IntEnum

An enumeration.

MOT_Normal = 0

MOT_Backwards = 1

class msl.equipment.resources.thorlabs.kinesis.enums.**MOT_HomeLimitSwitchDirection**
Bases: enum.IntEnum

An enumeration.

MOT_LimitSwitchDirectionUndefined = 0

MOT_ReverseLimitSwitch = 1

MOT_ForwardLimitSwitch = 4

class msl.equipment.resources.thorlabs.kinesis.enums.**MOT_JogModes**
Bases: enum.IntEnum

An enumeration.

MOT_JogModeUndefined = 0

MOT_Continuous = 1

MOT_SingleStep = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**MOT_StopModes**
Bases: enum.IntEnum

An enumeration.

MOT_StopModeUndefined = 0

MOT_Immediate = 1

MOT_Profiled = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**MOT_ButtonModes**
Bases: enum.IntEnum

An enumeration.

MOT_ButtonModeUndefined = 0

MOT_JogMode = 1

MOT_Preset = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**MOT_VelocityProfileModes**
Bases: enum.IntEnum

An enumeration.

MOT_Trapezoidal = 0

MOT_SCurve = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**MOT_LimitSwitchModes**
Bases: enum.IntEnum

An enumeration.

MOT_LimitSwitchModeUndefined = 0

MOT_LimitSwitchIgnoreSwitch = 1

MOT_LimitSwitchMakeOnContact = 2

MOT_LimitSwitchBreakOnContact = 3

MOT_LimitSwitchMakeOnHome = 4

MOT_LimitSwitchBreakOnHome = 5

MOT_PMD_Reserved = 6

MOT_LimitSwitchIgnoreSwitchSwapped = 129

MOT_LimitSwitchMakeOnContactSwapped = 130

MOT_LimitSwitchBreakOnContactSwapped = 131

MOT_LimitSwitchMakeOnHomeSwapped = 132

MOT_LimitSwitchBreakOnHomeSwapped = 133

class msl.equipment.resources.thorlabs.kinesis.enums.**MOT_LimitSwitchSWModes**
Bases: enum.IntEnum

An enumeration.

MOT_LimitSwitchSWModeUndefined = 0

MOT_LimitSwitchIgnored = 1

MOT_LimitSwitchStopImmediate = 2

MOT_LimitSwitchStopProfiled = 3

MOT_LimitSwitchIgnored_Rotational = 129

MOT_LimitSwitchStopImmediate_Rotational = 130

MOT_LimitSwitchStopProfiled_Rotational = 131

class msl.equipment.resources.thorlabs.kinesis.enums.**MOT_LimitsSoftwareApproachPol**
Bases: enum.IntEnum

An enumeration.

DisallowIllegalMoves = 0

AllowPartialMoves = 1

AllowAllMoves = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**MOT_CurrentLoopPhases**
Bases: enum.IntEnum

An enumeration.

MOT_PhaseA = 0

MOT_PhaseB = 1

MOT_PhaseAB = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**MOT_PID_LoopMode**
Bases: enum.IntEnum

An enumeration.

MOT_PIDLoopModeDisabled = 0

MOT_PIDOpenLoopMode = 1

MOT_PIDClosedLoopMode = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**NT_SignalState**
Bases: enum.IntEnum

An enumeration.

NT_BadSignal = 0

NT_GoodSignal = 1

class msl.equipment.resources.thorlabs.kinesis.enums.**NT_Mode**
Bases: enum.IntEnum

An enumeration.

NT_ModeUndefined = 0

NT_Piezo = 1

NT_Latch = 2

NT_Tracking = 3

NT_HorizontalTracking = 4

NT_VerticalTracking = 5

class msl.equipment.resources.thorlabs.kinesis.enums.**NT_ControlMode**
Bases: enum.IntEnum

An enumeration.

NT_ControlModeUndefined = 0

NT_OpenLoop = 1

NT_ClosedLoop = 2

NT_OpenLoopSmoothed = 3

NT_ClosedLoopSmoothed = 4

class msl.equipment.resources.thorlabs.kinesis.enums.**NT_FeedbackSource**
Bases: enum.IntEnum

An enumeration.

NT_FeedbackSourceUndefined = 0

NT_TIA = 1

NT_BNC_1v = 2

NT_BNC_2v = 3

NT_BNC_5v = 4

NT_BNC_10v = 5

class msl.equipment.resources.thorlabs.kinesis.enums.**NT_TIARange**
Bases: enum.IntEnum

An enumeration.

NT_TIARange1_3nA = 3

NT_TIARange2_10nA = 4

NT_TIARange3_30nA = 5

NT_TIARange4_100nA = 6

NT_TIARange5_300nA = 7

NT_TIARange6_1uA = 8

NT_TIARange7_3uA = 9

NT_TIARange8_10uA = 10

NT_TIARange9_30uA = 11

NT_TIARange10_100uA = 12

NT_TIARange11_300uA = 13

NT_TIARange12_1mA = 14

NT_TIARange13_3mA = 15

NT_TIARange14_10mA = 16

class msl.equipment.resources.thorlabs.kinesis.enums.**NT_OddOrEven**
Bases: enum.IntEnum

An enumeration.

NT_OddAndEven = 1

NT_Odd = 2

NT_Even = 3

class msl.equipment.resources.thorlabs.kinesis.enums.**NT_UnderOrOver**
Bases: enum.IntEnum

An enumeration.

NT_InRange = 1

NT_UnderRange = 2

NT_OverRange = 3

class msl.equipment.resources.thorlabs.kinesis.enums.**NT_CircleDiameterMode**
Bases: enum.IntEnum

An enumeration.

NT_ParameterCircleMode = 1

NT_AbsPowerCircleMode = 2

NT_LUTCircleMode = 3

class msl.equipment.resources.thorlabs.kinesis.enums.**NT_CircleAdjustment**
Bases: enum.IntEnum

An enumeration.

NT_LinearCircleAdjustment = 1

NT_LogCircleAdjustment = 2

NT_SquareCircleAdjustment = 3

NT_CubeCircleAdjustment = 4

class msl.equipment.resources.thorlabs.kinesis.enums.**NT_TIARangeMode**
Bases: enum.IntEnum

An enumeration.

NT_TIARangeModeUndefined = 0

NT_AutoRangeAtSelected = 1

NT_ManualRangeAtSelected = 2

NT_ManualRangeAtParameter = 3

NT_AutoRangeAtParameter = 4

class msl.equipment.resources.thorlabs.kinesis.enums.**NT_LowPassFrequency**
Bases: enum.IntEnum

An enumeration.

NT_LowPassNone = 0

NT_LowPass_1Hz = 1

NT_LowPass_3Hz = 2

NT_LowPass_10Hz = 3

NT_LowPass_30Hz = 4

NT_LowPass_100Hz = 5

class msl.equipment.resources.thorlabs.kinesis.enums.**NT_VoltageRange**
Bases: enum.IntEnum

An enumeration.

NT_VoltageRangeUndefined = 0

NT_VoltageRange_5v = 1

NT_VoltageRange_10v = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**NT_OutputVoltageRoute**
Bases: enum.IntEnum

An enumeration.

NT_SMAOnly = 1

NT_HubOrSMA = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**NT_PowerInputUnits**
Bases: enum.IntEnum

An enumeration.

NT_Amps = 0

NT_Watts = 1

NT_Db = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**NT_SMA_Units**
Bases: enum.IntEnum

An enumeration.

NT_Voltage = 0

NT_FullRange = 1

NT_UserDefined = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**BNT_CurrentLimit**
Bases: enum.IntEnum

An enumeration.

NT_CurrentLimit_100mA = 0

NT_CurrentLimit_250mA = 1

NT_CurrentLimit_500mA = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**BNT_OutputLowPassFilter**
Bases: enum.IntEnum

An enumeration.

NT_OutputFilter_10Hz = 0

NT_OutputFilter_100Hz = 1

NT_OutputFilter_5kHz = 2

NT_OutputFilter_None = 3

class msl.equipment.resources.thorlabs.kinesis.enums.**BNT_FeedbackSignalSelection**
Bases: enum.IntEnum

An enumeration.

NT_FeedbackSignalDC = 0

NT_FeedbackSignalAC = 65535

class msl.equipment.resources.thorlabs.kinesis.enums.**BNT_BNCTriggerModes**
Bases: enum.IntEnum

An enumeration.

NT_BNCModeTrigger = 0

NT_BNCModeLVOut = 65535

class msl.equipment.resources.thorlabs.kinesis.enums.**PZ_ControlModeTypes**
Bases: enum.IntEnum

An enumeration.

PZ_Undefined = 0

PZ_OpenLoop = 1

PZ_CloseLoop = 2

PZ_OpenLoopSmooth = 3

PZ_CloseLoopSmooth = 4

class msl.equipment.resources.thorlabs.kinesis.enums.**PZ_InputSourceFlags**
Bases: enum.IntEnum

An enumeration.

PZ_SoftwareOnly = 0

PZ_ExternalSignal = 1

PZ_Potentiometer = 2

PZ_All = 3

class msl.equipment.resources.thorlabs.kinesis.enums.**PZ_OutputLUTModes**
Bases: enum.IntEnum

An enumeration.

PZ_Continuous = 1

PZ_Fixed = 2

PZ_OutputTrigEnable = 4

PZ_InputTrigEnable = 8

PZ_OutputTrigSenseHigh = 16

PZ_InputTrigSenseHigh = 32

PZ_OutputGated = 64

PZ_OutputTrigRepeat = 128

class msl.equipment.resources.thorlabs.kinesis.enums.**PPC_DerivFilterState**
Bases: enum.IntEnum

An enumeration.

DerivFilterOn = 1

DerivFilterOff = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**PPC_NotchFilterState**
Bases: enum.IntEnum

An enumeration.

NotchFilterOn = 1

NotchFilterOff = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**PPC_NotchFilterChannel**
Bases: enum.IntEnum

An enumeration.

NotchFilter1 = 1

NotchFilter2 = 2

NotchFilterBoth = 3

class msl.equipment.resources.thorlabs.kinesis.enums.**PPC_IOControlMode**
Bases: enum.IntEnum

An enumeration.

SWOnly = 0

ExtBNC = 1

Joystick = 2

JoystickBnc = 3

class msl.equipment.resources.thorlabs.kinesis.enums.**PPC_IOOutputMode**
Bases: enum.IntEnum

An enumeration.

HV = 1

PosRaw = 2

PosCorrected = 3

class msl.equipment.resources.thorlabs.kinesis.enums.**PPC_IOOutputBandwidth**
Bases: enum.IntEnum

An enumeration.

OP_Unfiltered = 1

OP_200Hz = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**PPC_IOFeedbackSourceDefinitio**
Bases: enum.IntEnum

An enumeration.

StrainGauge = 1

Capacitive = 2

Optical = 3

class msl.equipment.resources.thorlabs.kinesis.enums.**PPC_DisplayIntensity**
Bases: enum.IntEnum

An enumeration.

Bright = 1

Dim = 2

Off = 3

class msl.equipment.resources.thorlabs.kinesis.enums.**FF_Positions**
Bases: enum.IntEnum

An enumeration.

FF_PositionError = 0

Position1 = 1

Position2 = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**FF_IOModes**
Bases: enum.IntEnum

An enumeration.

FF_ToggleOnPositiveEdge = 1

FF_SetPositionOnPositiveEdge = 2

FF_OutputHighAtSetPosition = 4

FF_OutputHighWhemMoving = 8

class msl.equipment.resources.thorlabs.kinesis.enums.**FF_SignalModes**
Bases: enum.IntEnum

An enumeration.

FF_InputButton = 1

FF_InputLogic = 2

FF_InputSwap = 4

FF_OutputLevel = 16

FF_OutputPulse = 32

FF_OutputSwap = 64

class msl.equipment.resources.thorlabs.kinesis.enums.**KMOT_JoystickDirectionSense**
Bases: enum.IntEnum

An enumeration.

KMOT_JS_Positive = 1

KMOT_JS_Negative = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**KMOT_JoyStickMode**
Bases: enum.IntEnum

An enumeration.

KMOT_JS_Velocity = 1

KMOT_JS_Jog = 2

KMOT_JS_MoveAbsolute = 3

class msl.equipment.resources.thorlabs.kinesis.enums.**KMOT_TriggerPortMode**
Bases: enum.IntEnum

An enumeration.

KMOT_TrigDisabled = 0

KMOT_TrigIn_GPI = 1

KMOT_TrigIn_RelativeMove = 2

KMOT_TrigIn_AbsoluteMove = 3

KMOT_TrigIn_Home = 4

KMOT_TrigOut_GPO = 10

KMOT_TrigOut_InMotion = 11

KMOT_TrigOut_AtMaxVelocity = 12

KMOT_TrigOut_AtPositionSteps = 13

KMOT_TrigOut_Synch = 14

class msl.equipment.resources.thorlabs.kinesis.enums.**KMOT_TriggerPortPolarity**
Bases: enum.IntEnum

An enumeration.

KMOT_TrigPolarityHigh = 1

KMOT_TrigPolarityLow = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**LS_InputSourceFlags**
Bases: enum.IntEnum

An enumeration.

LS_SoftwareOnly = 0

LS_ExternalSignal = 1

LS_Potentiometer = 4

class msl.equipment.resources.thorlabs.kinesis.enums.**LS_DisplayUnits**
Bases: enum.IntEnum

An enumeration.

LS_mAmps = 1

LS_mWatts = 2

LS_mDb = 3

class msl.equipment.resources.thorlabs.kinesis.enums.**KLS_OpMode**
Bases: enum.IntEnum

An enumeration.

KLS_ConstantPower = 0

KLS_ConstantCurrent = 1

class msl.equipment.resources.thorlabs.kinesis.enums.**KLS_TriggerMode**
Bases: enum.IntEnum

An enumeration.

KLS_Disabled = 0

KLS_Input = 1

KLS_ModulationTrigger = 2

KLS_SetPower = 3

KLS_Output = 10

KLS_LaserOn = 11

KLS_InterlockEnabled = 12

KLS_SetPointChange = 13

KLS_HighStability = 14

KLS_LowStability = 15

class msl.equipment.resources.thorlabs.kinesis.enums.**KLS_TrigPolarity**
Bases: enum.IntEnum

An enumeration.

KLS_TrigPol_High = 1

KLS_TrigPol_Low = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**KPZ_JoystickDirectionSense**
Bases: enum.IntEnum

An enumeration.

KPZ_JS_Positive = 1

KPZ_JS_Negative = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**KPZ_JoyStickMode**
Bases: enum.IntEnum

An enumeration.

KPZ_JS_MoveAtVoltage = 1

KPZ_JS_JogVoltage = 2

KPZ_JS_SetVoltage = 3

class msl.equipment.resources.thorlabs.kinesis.enums.**KPZ_JoyStickChangeRate**
Bases: enum.IntEnum

An enumeration.

KPZ_JS_High = 1

KPZ_JS_Medium = 2

KPZ_JS_Low = 3

class msl.equipment.resources.thorlabs.kinesis.enums.**KPZ_TriggerPortMode**
Bases: enum.IntEnum

An enumeration.

KPZ_TrigDisabled = 0

KPZ_TrigIn_GPI = 1

KPZ_TrigIn_VoltageStepUp = 2

KPZ_TrigIn_VoltageStepDown = 3

KPZ_TrigOut_GPO = 10

class msl.equipment.resources.thorlabs.kinesis.enums.**KPZ_TriggerPortPolarity**
Bases: enum.IntEnum

An enumeration.

KPZ_TrigPolarityHigh = 1

KPZ_TrigPolarityLow = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**HubAnalogueModes**
Bases: enum.IntEnum

An enumeration.

AnalogueCh1 = 1

AnalogueCh2 = 2

ExtSignalSMA = 3

class msl.equipment.resources.thorlabs.kinesis.enums.**QD_OperatingMode**
Bases: enum.IntEnum

An enumeration.

QD_ModeUndefined = 0

QD_Monitor = 1

QD_OpenLoop = 2

QD_ClosedLoop = 3

QD_AutoOpenClosedLoop = 4

class msl.equipment.resources.thorlabs.kinesis.enums.**QD_LowVoltageRoute**
Bases: enum.IntEnum

An enumeration.

QD_RouteUndefined = 0

QD_SMAOnly = 1

QD_HubAndSMA = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**QD_OpenLoopHoldValues**
Bases: enum.IntEnum

An enumeration.

QD_HoldOnZero = 1

QD_HoldOnLastValue = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**QD_FilterEnable**
Bases: enum.IntEnum

An enumeration.

QD_Undefined = 0

QD_Enabled = 1

QD_Disabled = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**QD_KPA_TrigModes**
Bases: enum.IntEnum

An enumeration.

QD_Trig_Disabled = 0

QD_TrigIn_GPI = 1

QD_TrigIn_LoopOpenClose = 2

KD_TrigOut_GPO = 10

KD_TrigOut_Sum = 11

KD_TrigOut_Diff = 12

KD_TrigOut_SumDiff = 13

class msl.equipment.resources.thorlabs.kinesis.enums.**QD_KPA_TrigPolarities**
Bases: enum.IntEnum

An enumeration.

GD_Trig_High = 1

GD_Trig_Low = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**SC_OperatingModes**
Bases: enum.IntEnum

An enumeration.

SC_Manual = 1

SC_Single = 2

SC_Auto = 3

SC_Triggered = 4

class msl.equipment.resources.thorlabs.kinesis.enums.**SC_OperatingStates**
Bases: enum.IntEnum

An enumeration.

SC_Active = 1

SC_Inactive = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**SC_SolenoidStates**
Bases: enum.IntEnum

An enumeration.

SC_SolenoidOpen = 1

SC_SolenoidClosed = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**KSC_TriggerPortMode**
Bases: enum.IntEnum

An enumeration.

KSC_TrigDisabled = 0

KSC_TrigIn_GPI = 1

KSC_TrigOut_GPO = 10

class msl.equipment.resources.thorlabs.kinesis.enums.**KSC_TriggerPortPolarity**
Bases: enum.IntEnum

An enumeration.

KSC_TrigPolarityHigh = 1

KSC_TrigPolarityLow = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**TIM_Channels**
Bases: enum.IntEnum

An enumeration.

Channel1 = 1

Channel2 = 2

Channel3 = 3

Channel4 = 4

class msl.equipment.resources.thorlabs.kinesis.enums.**TIM_JogMode**
Bases: enum.IntEnum

An enumeration.

JogContinuous = 1

JogStep = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**TIM_ButtonsMode**
Bases: enum.IntEnum

An enumeration.

Jog = 1

Position = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**TIM_Direction**
Bases: enum.IntEnum

An enumeration.

Forward = 1

Reverse = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**LD_InputSourceFlags**
Bases: enum.IntEnum

An enumeration.

LD_SoftwareOnly = 1

LD_ExternalSignal = 2

LD_Potentiometer = 4

class msl.equipment.resources.thorlabs.kinesis.enums.**LD_DisplayUnits**
Bases: enum.IntEnum

An enumeration.

LD_ILim = 1

LD_ILD = 2

LD_IPD = 3

LD_PLD = 4

class msl.equipment.resources.thorlabs.kinesis.enums.**LD_TIA_RANGES**
Bases: enum.IntEnum

An enumeration.

LD_TIA_10uA = 1

LD_TIA_100uA = 2

LD_TIA_1mA = 4

LD_TIA_10mA = 8

class msl.equipment.resources.thorlabs.kinesis.enums.**LD_POLARITY**
Bases: enum.IntEnum

An enumeration.

LD_CathodeGrounded = 1

LD_AnodeGrounded = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**TSG_Hub_Analogue_Modes**
Bases: enum.IntEnum

An enumeration.

TSG_HubChannel1 = 1

TSG_HubChannel2 = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**TSG_Display_Modes**
Bases: enum.IntEnum

An enumeration.

TSG_Undefined = 0

TSG_Position = 1

TSG_Voltage = 2

TSG_Force = 3

class msl.equipment.resources.thorlabs.kinesis.enums.**TC_SensorTypes**
Bases: enum.IntEnum

An enumeration.

TC_Transducer = 0

TC_TH20kOhm = 1

TC_TH200kOhm = 2

class msl.equipment.resources.thorlabs.kinesis.enums.**TC_DisplayModes**
Bases: enum.IntEnum

An enumeration.

TC_ActualTemperature = 0

TC_TargetTemperature = 1

TC_TempDifference = 2

TC_Current = 3

msl.equipment.resources.thorlabs.kinesis.errors module

Device and Low Level Error Codes defined in the Thorlabs Kinesis software v1.11.0

msl.equipment.resources.thorlabs.kinesis.filter_flipper module

This module provides all the functionality required to control a Filter Flipper (MFF101).

class msl.equipment.resources.thorlabs.kinesis.filter_flipper.**FilterFlipper**(*record*)
Bases: *msl.equipment.resources.thorlabs.kinesis.motion_control.MotionControl*

A wrapper around Thorlabs.MotionControl.FilterFlipper

Parameters **record** (*EquipmentRecord*) – An equipment record from an **Equipment-Register** *Database*.

MIN_TRANSIT_TIME = 300

MAX_TRANSIT_TIME = 2800

MIN_PULSE_WIDTH = 10

MAX_PULSE_WIDTH = 200

open()
Open the device for communication.

Raises ConnectionError – If not successful.

close()
Disconnect and close the device.

check_connection()
Check connection.

Returns bool – Whether the USB is listed by the FTDI controller.

identify()
Sends a command to the device to make it identify itself.

get_hardware_info()
Gets the hardware information from the device.

Returns *structs.TLI_HardwareInformation* – The hardware information.

Raises ConnectionError – If not successful.

get_firmware_version()
Gets version number of the device firmware.

Returns str – The firmware version.

get_software_version()
Gets version number of the device software.

Returns str – The device software version.

load_settings()
Update device with stored settings.

The settings are read from ThorlabsDefaultSettings.xml, which gets created when the Kinesis software is installed.

Raises ConnectionError – If not successful.

persist_settings()
Persist the devices current settings.

Raises ConnectionError – If not successful.

get_number_positions()
Get number of positions available from the device.

Returns int – The number of positions.

home()
Home the device.

Homing the device will set the device to a known state and determine the home position.

Raises ConnectionError – If not successful.

move_to_position(*position*, *wait=False*)
Move the device to the specified position (index).

Parameters

- **position** (int) – The required position. Must be 1 or 2.
- **wait** (bool) – Whether to wait until the movement is complete before returning to the calling program.

Raises ConnectionError – If not successful.

get_position()
Get the current position (index).

Returns int – The current position, 1 or 2.

get_io_settings()
Gets the I/O settings from filter flipper.

Returns *FF_IOSettings* – The Filter Flipper I/O settings.

Raises ConnectionError – If not successful.

request_io_settings()
Requests the I/O settings from the filter flipper.

Raises ConnectionError – If not successful.

set_io_settings(*transit_time=500*, *oper1=<FF_IOModes.FF_ToggleOnPositiveEdge: 1>*, *sig1=<FF_SignalModes.FF_InputButton: 1>*, *pw1=200*, *oper2=<FF_IOModes.FF_ToggleOnPositiveEdge: 1>*, *sig2=<FF_SignalModes.FF_OutputLevel: 16>*, *pw2=200*)
Sets the settings on filter flipper.

Parameters

- **transit_time** (int) – Time taken to get from one position to other in milliseconds.
- **oper1** (*FF_IOModes*) – I/O 1 Operating Mode.
- **sig1** (*FF_SignalModes*) – I/O 1 Signal Mode.
- **pw1** (int) – Digital I/O 1 pulse width in milliseconds.
- **oper2** (*FF_IOModes*) – I/O 2 Operating Mode.
- **sig2** (*FF_SignalModes*) – I/O 2 Signal Mode.
- **pw2** (int) – Digital I/O 2 pulse width in milliseconds.

Raises ConnectionError – If not successful.

get_transit_time()
Gets the transit time.

Returns int – The transit time in milliseconds.

set_transit_time(*transit_time*)
Sets the transit time.

Parameters **transit_time** (int) – The transit time in milliseconds.

Raises ConnectionError – If not successful.

request_status()
Request status bits.

This needs to be called to get the device to send it's current status.

This is called automatically if Polling is enabled for the device using *start_polling()*.

Raises ConnectionError – If not successful.

get_status_bits()
Get the current status bits.

This returns the latest status bits received from the device. To get new status bits, use *request_status()* or use the polling function, *start_polling()*

Returns int – The status bits from the device.

start_polling(*milliseconds*)
Starts the internal polling loop.

This function continuously requests position and status messages.

Parameters **milliseconds** (int) – The polling rate, in milliseconds.

Raises ConnectionError – If not successful.

polling_duration()
Gets the polling loop duration.

Returns int – The time between polls in milliseconds or 0 if polling is not active.

stop_polling()
Stops the internal polling loop.

time_since_last_msg_received()
Gets the time, in milliseconds, since tha last message was received.

This can be used to determine whether communications with the device is still good.

Returns int – The time, in milliseconds, since the last message was received.

enable_last_msg_timer(*enable*, *msg_timeout=0*)
Enables the last message monitoring timer.

This can be used to determine whether communications with the device is still good.

Parameters

- **enable** (bool) – True to enable monitoring otherwise False to disable.
- **msg_timeout** (int) – The last message error timeout in ms. Set to 0 to disable.

has_last_msg_timer_overrun()
Queries if the time since the last message has exceeded the lastMsgTimeout set by *enable_last_msg_timer()*.

This can be used to determine whether communications with the device is still good.

Returns bool – True if last message timer has elapsed or False if monitoring is not enabled or if time of last message received is less than msg_timeout.

request_settings()
Requests that all settings are downloaded from the device.

This function requests that the device upload all it's settings to the DLL.

Raises ConnectionError – If not successful.

clear_message_queue()
Clears the device message queue.

register_message_callback(*callback*)
Registers a callback on the message queue.

Parameters **callback** (*callbacks.MotionControlCallback*) – A function to be called whenever messages are received.

message_queue_size()
Gets the size of the message queue.

Returns int – The number of messages in the queue.

get_next_message()
Get the next Message Queue item. See *messages*.

Returns

- int – The message type.
- int – The message ID.
- int – The message data.

Raises ConnectionError – If not successful.

wait_for_message()
Wait for next Message Queue item. See *messages*.

Returns

- int – The message type.
- int – The message ID.
- int – The message data.

Raises ConnectionError – If not successful.

msl.equipment.resources.thorlabs.kinesis.integrated_stepper_motors module

This module provides all the functionality required to control a number of **Integrated Stepper Motors** including:

- Long Travel Stages (LTS150 and LTS300)
- Lab Jack (MLJ050)
- Cage Rotator (K10CR1)

class msl.equipment.resources.thorlabs.kinesis.integrated_stepper_motors.**UnitType**
Bases: enum.IntEnum

An Integrated Stepper Motor device unit.

Used to *get_real_value_from_device_unit* and to *get_device_unit_from_real_value*.

DISTANCE = 0

VELOCITY = 1

ACCELERATION = 2

class msl.equipment.resources.thorlabs.kinesis.integrated_stepper_motors.**IntegratedStepperMotors**

Bases: *msl.equipment.resources.thorlabs.kinesis.motion_control.MotionControl*

A wrapper around Thorlabs.MotionControl.IntegratedStepperMotors

The *record.connection.properties* dictionary for a IntegratedStepperMotors device supports the following key-value pairs:

```
'load_settings': bool,  # optional, default is True (load the default settings when connection is created)
```

Parameters **record** (*EquipmentRecord*) – An equipment record from an **Equipment-Register** *Database*.

can_home()
Can the device perform a *home()*?

Returns bool – Whether the device can be homed.

can_move_without_homing_first()
Does the device need to be *home*'d before a move can be performed?

Returns bool – Whether the device needs to be homed.

check_connection()
Check connection.

Returns bool – Whether the USB is listed by the FTDI controller.

clear_message_queue()
Clears the device message queue.

close()
Disconnect and close the device.

disable_channel()
Disable the channel so that motor can be moved by hand.

When disabled, power is removed from the motor and it can be freely moved.

Raises ConnectionError – If not successful.

enable_channel()
Enable channel for computer control.

When enabled, power is applied to the motor so it is fixed in position.

Raises ConnectionError – If not successful.

enable_last_msg_timer(*enable*, *last_msg_timeout*)
Enables the last message monitoring timer.

This can be used to determine whether communications with the device is still good.

Parameters

- **enable** (bool) – True to enable monitoring otherwise False to disable.
- **last_msg_timeout** (int) – The last message error timeout in ms. Set to 0 to disable.

get_backlash()
Get the backlash distance setting (used to control hysteresis).

See *get_real_value_from_device_unit* for converting from a device_unit to a real_value.

Returns int – The backlash distance in DeviceUnits (see manual).

get_bow_index()
Gets the stepper motor bow index.

Returns int – The bow index.

get_button_params()
Gets the LTS button parameters.

See *get_real_value_from_device_unit* for converting from a device_unit to a real_value.

Returns

- *enums.MOT_ButtonModes* – The button mode.
- int – The Preset position in DeviceUnits for the left button (when in preset mode).
- int – The Preset position in DeviceUnits for the right button (when in preset mode).
- int – The time that buttons need to be pressed in order to go home or to record a preset buttons defined position.

Raises ConnectionError – If not successful.

get_button_params_block()
Get the button parameters.

Returns *structs.MOT_ButtonParameters* – The button parameters.

Raises ConnectionError – If not successful.

get_calibration_file()
Get calibration file for this motor.

Returns str – The filename of the calibration file.

Raises ConnectionError – If not successful.

get_device_unit_from_real_value(*real_value*, *unit_type*)
Converts a real-world value to a device value.

Either *load_settings()* or *set_motor_params_ext()* must be called before calling this function, otherwise the returned value will always be 0.

Parameters

- **real_value** (float) – The real-world value.
- **unit_type** (*UnitType*) – The unit of the real-world value.

Returns int – The device value.

Raises ConnectionError – If not successful.

get_firmware_version()
Gets version number of the device firmware.

Returns str – The firmware version.

get_hardware_info()
Gets the hardware information from the device.

Returns *structs.TLI_HardwareInformation* – The hardware information.

Raises ConnectionError – If not successful.

get_hardware_info_block()
Gets the hardware information in a block.

Returns *structs.TLI_HardwareInformation* – The hardware information.

Raises ConnectionError – If not successful.

get_homing_params_block()
Get the homing parameters.

Returns *structs.MOT_HomingParameters* – The homing parameters.

Raises ConnectionError – If not successful.

get_homing_velocity()
Gets the homing velocity.

See *get_real_value_from_device_unit* for converting from a device_unit to a real_value.

Returns int – The homing velocity in DeviceUnits (see manual).

get_jog_mode()
Gets the jog mode.

Returns

- *enums.MOT_JogModes* – The jog mode.
- *enums.MOT_StopModes* – The stop mode.

Raises ConnectionError – If not successful.

get_jog_params_block()
Get the jog parameters.

Returns *structs.MOT_JogParameters* – The jog parameters.

Raises ConnectionError – If not successful.

get_jog_step_size()
Gets the distance to move when jogging.

See *get_real_value_from_device_unit* for converting from a device_unit to a real_value.

Returns int – The step size in DeviceUnits (see manual).

get_jog_vel_params()
Gets the jog velocity parameters.

See *get_real_value_from_device_unit* for converting from a device_unit to a real_value.

Returns

- int – The maximum velocity in DeviceUnits (see manual).
- int – The acceleration in DeviceUnits (see manual).

Raises ConnectionError – If not successful.

get_led_switches()
Get the LED indicator bits on the device.

Returns int – Sum of: 8 to indicate moving 2 to indicate end of track and 1 to flash on identify command.

get_limit_switch_params()
Gets the limit switch parameters.

See *get_real_value_from_device_unit* for converting from a device_unit to a real_value.

Returns

- *enums.MOT_LimitSwitchModes* – The clockwise hardware limit mode.
- *enums.MOT_LimitSwitchModes* – The anticlockwise hardware limit mode.
- int – The position of the clockwise software limit in DeviceUnits (see manual).
- int – The position of the anticlockwise software limit in DeviceUnits (see manual).
- *enums.MOT_LimitSwitchSWModes* – The soft limit mode.

Raises ConnectionError – If not successful.

get_limit_switch_params_block()
Get the limit switch parameters.

Returns *structs.MOT_LimitSwitchParameters* – The limit switch parameters.

Raises ConnectionError – If not successful.

get_motor_params()
Gets the motor stage parameters.

Deprecated: calls *get_motor_params_ext()*

These parameters, when combined define the stage motion in terms of RealWorldUnits [millimeters or degrees]. The real-world unit is defined from steps_per_rev * gear_box_ratio / pitch.

Returns

- `float` – The steps per revolution.
- `float` – The gear box ratio.
- `float` – The pitch.

Raises `ConnectionError` – If not successful.

get_motor_params_ext()
Gets the motor stage parameters.

These parameters, when combined define the stage motion in terms of `RealWorldUnits` [millimeters or degrees]. The real-world unit is defined from `steps_per_rev * gear_box_ratio / pitch`.

Returns

- `float` – The steps per revolution.
- `float` – The gear box ratio.
- `float` – The pitch.

Raises `ConnectionError` – If not successful.

get_motor_travel_limits()
Gets the motor stage min and max position.

Returns

- `float` – The minimum position in `RealWorldUnits` [millimeters or degrees].
- `float` – The maximum position in `RealWorldUnits` [millimeters or degrees].

Raises `ConnectionError` – If not successful.

get_motor_travel_mode()
Get the motor travel mode.

Returns *enums.MOT_TravelModes* – The travel mode.

get_motor_velocity_limits()
Gets the motor stage maximum velocity and acceleration.

Returns

- `float` – The maximum velocity in `RealWorldUnits` [millimeters or degrees].
- `float` – The maximum acceleration in `RealWorldUnits` [millimeters or degrees].

Raises `ConnectionError` – If not successful.

get_move_absolute_position()
Gets the move absolute position.

See *get_real_value_from_device_unit* for converting from a `device_unit` to a `real_value`.

Returns `int` – The absolute position in `DeviceUnits` (see manual).

get_move_relative_distance()
Gets the move relative distance.

See *get_real_value_from_device_unit* for converting from a device_unit to a real_value.

Returns int – The move relative position in DeviceUnits (see manual).

get_next_message()
Get the next Message Queue item. See *messages*.

Returns

- int – The message type.
- int – The message ID.
- int – The message data.

Raises ConnectionError – If not successful.

get_number_positions()
Get number of positions.

This function will get the maximum position reachable by the device. The motor may need to be set to its *home()* position before this parameter can be used.

Returns int – The number of positions.

get_position()
Get the current position.

See *get_real_value_from_device_unit* for converting from a device_unit to a real_value.

Returns **index** (int) – The position in DeviceUnits (see manual).

get_position_counter()
Get the position counter.

The position counter is identical to the position parameter. The position counter is set to zero when homing is complete.

See *get_real_value_from_device_unit* for converting from a device_unit to a real_value.

Returns int – The position counter in DeviceUnits (see manual).

get_potentiometer_params(*index*)
Gets the potentiometer parameters for the LTS.

See *get_real_value_from_device_unit* for converting from a device_unit to a real_value.

Parameters **index** (int) – The potentiometer index to be read.

Returns

- int – The potentiometer threshold, range 0 to 127.
- int – The velocity in DeviceUnits for the current potentiometer threshold.

Raises ConnectionError – If not successful.

get_potentiometer_params_block()
Get the potentiometer parameters.

Returns *structs.MOT_PotentiometerSteps* – The potentiometer parameters.

Raises ConnectionError – If not successful.

get_power_params()
Gets the power parameters for the stepper motor.

Returns *structs.MOT_PowerParameters* – The power parameters.

Raises ConnectionError – If not successful.

get_real_value_from_device_unit(*device_value*, *unit_type*)
Converts a device value to a real-world value.

Either *load_settings()* or *set_motor_params_ext()* must be called before calling this function, otherwise the returned value will always be 0.0.

Parameters

- **device_value** (int) – The device value.
- **unit_type** (*UnitType*) – The unit of the device value.

Returns float – The real-world value.

Raises ConnectionError – If not successful.

get_soft_limit_mode()
Gets the software limits mode.

Returns *enums.MOT_LimitsSoftwareApproachPolicy* – The software limits mode.

get_software_version()
Gets version number of the device software.

Returns str – The device software version.

get_stage_axis_max_pos()
Gets the LTS Motor maximum stage position.

See *get_real_value_from_device_unit* for converting from a device_unit to a real_value.

Returns int – The maximum position in DeviceUnits (see manual).

get_stage_axis_min_pos()
Gets the LTS Motor minimum stage position.

See *get_real_value_from_device_unit* for converting from a device_unit to a real_value.

Returns int – The minimum position in DeviceUnits (see manual).

get_status_bits()
Get the current status bits.

This returns the latest status bits received from the device. To get new status bits, use *request_status_bits()* or use *request_status()* or use the polling functions, *start_polling()*.

Returns int – The status bits from the device.

get_trigger_switches()
Gets the trigger switch bits.

Returns int – 8 bits indicating action on trigger input and events to trigger electronic output.

get_vel_params()
Gets the move velocity parameters.

See *get_real_value_from_device_unit* for converting from a device_unit to a real_value.

Returns

- **max_velocity** (int) – The maximum velocity in DeviceUnits (see manual).
- **acceleration** (int) – The acceleration in DeviceUnits (see manual).

Raises ConnectionError – If not successful.

get_vel_params_block()
Get the move velocity parameters.

Returns *structs.MOT_VelocityParameters* – The velocity parameters.

Raises ConnectionError – If not successful.

has_last_msg_timer_overrun()
Queries if the time since the last message has exceeded the lastMsgTimeout set by *enable_last_msg_timer()*.

This can be used to determine whether communications with the device is still good.

Returns bool – True if last message timer has elapsed or False if monitoring is not enabled or if time of last message received is less than lastMsgTimeout.

home(*wait=True*)
Home the device.

Homing the device will set the device to a known state and determine the home position.

Parameters **wait** (bool) – Wait until the device has been homed before returning to the calling program.

Raises ConnectionError – If not successful.

identify()
Sends a command to the device to make it identify itself.

is_calibration_active()
Is a calibration file active for this motor?

Returns bool – Whether a calibration file is active.

load_settings()
Update device with stored settings.

The settings are read from `ThorlabsDefaultSettings.xml`, which gets created when the Kinesis software is installed.

Raises `ConnectionError` – If not successful.

message_queue_size()
Gets the size of the message queue.

Returns `int` – The number of messages in the queue.

move_absolute()
Moves the device to the position defined in *set_move_absolute_position()*.

Raises `ConnectionError` – If not successful.

move_at_velocity(*direction*)
Start moving at the current velocity in the specified direction.

Parameters **direction** (*enums.MOT_TravelDirection*) – The required direction of travel as a *enums.MOT_TravelDirection* enum value or member name.

Raises `ConnectionError` – If not successful.

move_jog(*jog_direction*)
Perform a jog.

Parameters **jog_direction** (*enums.MOT_TravelDirection*) – The jog direction as a *enums.MOT_TravelDirection* enum value or member name.

Raises `ConnectionError` – If not successful.

move_relative(*displacement*)
Move the motor by a relative amount.

See *get_device_unit_from_real_value* for converting from a `real_value` to a `device_unit`.

Parameters **displacement** (`int`) – Signed displacement in `DeviceUnits` (see manual).

Raises `ConnectionError` – If not successful.

move_relative_distance()
Moves the device by a relative distance defined by *set_move_relative_distance()*.

Raises `ConnectionError` – If not successful.

move_to_position(*index*, *wait=True*)
Move the device to the specified position (index).

The motor may need to be set to its *home()* position before a position can be set.

See *get_device_unit_from_real_value* for converting from a `real_value` to a `device_unit`.

Parameters

- **index** (int) – The position in DeviceUnits (see manual).
- **wait** (bool) –

Raises ConnectionError – If not successful.

needs_homing()
Does the device need to be *home*'d before a move can be performed?

Deprecated: calls *can_move_without_homing_first()* instead.

Returns bool – Whether the device needs to be homed.

open()
Open the device for communication.

Raises ConnectionError – If not successful.

persist_settings()
Persist the devices current settings.

Raises ConnectionError – If not successful.

polling_duration()
Gets the polling loop duration.

Returns int – The time between polls in milliseconds or 0 if polling is not active.

register_message_callback(*callback*)
Registers a callback on the message queue.

Parameters **callback** (*callbacks.MotionControlCallback*) – A function to be called whenever messages are received.

request_backlash()
Requests the backlash.

Raises ConnectionError – If not successful.

request_bow_index()
Requests the stepper motor bow index.

Raises ConnectionError – If not successful.

request_button_params()
Requests the LTS button parameters.

Raises ConnectionError – If not successful.

request_homing_params()
Requests the homing parameters.

Raises ConnectionError – If not successful.

request_jog_params()
Requests the jog parameters.

Raises ConnectionError – If not successful.

request_limit_switch_params()
Requests the limit switch parameters.

Raises ConnectionError – If not successful.

request_move_absolute_position()
Requests the position of next absolute move.

Raises ConnectionError – If not successful.

request_move_relative_distance()
Requests the relative move distance.

Raises ConnectionError – If not successful.

request_position()
Requests the current position.

This needs to be called to get the device to send it's current position. Note, this is called automatically if Polling is enabled for the device using *start_polling()*.

Raises ConnectionError – If not successful.

request_potentiometer_params()
Requests the potentiometer parameters.

Raises ConnectionError – If not successful.

request_power_params()
Requests the power parameters.

Raises ConnectionError – If not successful.

request_settings()
Requests that all settings are downloaded from the device.

This function requests that the device upload all it's settings to the DLL.

Raises ConnectionError – If not successful.

request_status()
Request position and status bits.

This needs to be called to get the device to send it's current status. Note, this is called automatically if Polling is enabled for the device using *start_polling()*.

Raises ConnectionError – If not successful.

request_status_bits()
Request the status bits which identify the current motor state.

This needs to be called to get the device to send it's current status bits. Note, this is called automatically if Polling is enabled for the device using *start_polling()*.

Raises ConnectionError – If not successful.

request_trigger_switches()
Requests the trigger switch bits.

Raises ConnectionError – If not successful.

request_vel_params()
Requests the velocity parameters.

Raises ConnectionError – If not successful.

reset_stage_to_defaults()
Reset the stage settings to defaults.

Raises ConnectionError – If not successful.

set_backlash(*distance*)
Sets the backlash distance (used to control hysteresis).

See *get_device_unit_from_real_value* for converting from a real_value to a device_unit.

Parameters **distance** (int) – The backlash distance in DeviceUnits (see manual).

Raises ConnectionError – If not successful.

set_bow_index(*bow_index*)
Sets the stepper motor bow index.

Parameters **bow_index** (int) – The bow index.

Raises ConnectionError – If not successful.

set_button_params(*button_mode*, *left_button_position*, *right_button_position*)
Sets the LTS button parameters.

See *get_device_unit_from_real_value* for converting from a real_value to a device_unit.

Parameters

- **button_mode** (*enums.MOT_ButtonModes*) – The button mode as a *enums.MOT_ButtonModes* enum value or member name.
- **left_button_position** (int) – The Preset position in DeviceUnits for the left button (when in preset mode).
- **right_button_position** (int) – The Preset position in DeviceUnits for the right button (when in preset mode).

Raises ConnectionError – If not successful.

set_button_params_block(*mode*, *left_button*, *right_button*, *timeout*)
Set the button parameters.

Parameters

- **mode** (*enums.MOT_ButtonModes*) – The mode of operation of the device buttons as a *enums.MOT_ButtonModes* enum value or member name.
- **left_button** (int) – Position in encoder counts to go to when left button is pressed.
- **right_button** (int) – Position in encoder counts to go to when right button is pressed.
- **timeout** (int) – The Time a button needs to be held down for to record the position as a preset.

Raises ConnectionError – If not successful.

set_calibration_file(*path*, *enabled*)
Set the calibration file for this motor.

Parameters

- **path** (str) – The path to a calibration file to load.
- **enabled** (bool) – True to enable, False to disable.

Raises FileNotFoundError – If the *path* does not exist.

set_direction(*reverse*)
Sets the motor direction sense.

This function is used because some actuators use have directions of motion reversed. This parameter will tell the system to reverse the direction sense when moving, jogging etc.

Parameters **reverse** (bool) – If True then directions will be swapped on these moves.

set_homing_params_block(*direction*, *limit*, *velocity*, *offset*)
Set the homing parameters.

Parameters

- **direction** (*enums.MOT_TravelDirection*) – The Homing direction sense as a *enums.MOT_TravelDirection* enum value or member name.
- **limit** (*enums.MOT_HomeLimitSwitchDirection*) – The limit switch direction as a *enums.MOT_HomeLimitSwitchDirection* enum value or member name.
- **velocity** (int) – The velocity in small indivisible units.
- **offset** (int) – Distance of home from limit in small indivisible units.

Raises ConnectionError – If not successful.

set_homing_velocity(*velocity*)
Sets the homing velocity.

See *get_device_unit_from_real_value* for converting from a real_value to a device_unit.

Parameters **velocity** (int) – The homing velocity in DeviceUnits (see manual).

Raises ConnectionError – If not successful.

set_jog_mode(*mode*, *stop_mode*)
Sets the jog mode.

Parameters

- **mode** (*enums.MOT_JogModes*) – The jog mode, as a *enums.MOT_JogModes* enum value or member name.
- **stop_mode** (*enums.MOT_StopModes*) – The stop mode, as a *enums.MOT_StopModes* enum value or member name.

Raises ConnectionError – If not successful.

set_jog_params_block(*jog_params*)
Set the jog parameters.

Parameters **jog_params** (*structs.MOT_JogParameters*) – The jog parameters.

Raises

- ConnectionError – If not successful.
- TypeError – If the data type of *jog_params* is not *structs.MOT_JogParameters*

set_jog_step_size(*step_size*)
Sets the distance to move on jogging.

See *get_device_unit_from_real_value* for converting from a real_value to a device_unit.

Parameters **step_size** (int) – The step size in DeviceUnits (see manual).

Raises ConnectionError – If not successful.

set_jog_vel_params(*max_velocity*, *acceleration*)
Sets jog velocity parameters.

See *get_device_unit_from_real_value* for converting from a real_value to a device_unit.

Parameters

- **max_velocity** (int) – The maximum velocity in DeviceUnits (see manual).
- **acceleration** (int) – The acceleration in DeviceUnits (see manual).

Raises ConnectionError – If not successful.

set_led_switches(*led_switches*)
Set the LED indicator bits on the device.

Parameters **led_switches** (int) – Sum of: 8 to indicate moving 2 to indicate end of track and 1 to flash on identify command.

Raises ConnectionError – If not successful.

set_limit_switch_params(*cw_lim*, *ccw_lim*, *cw_pos*, *ccw_pos*, *soft_limit_mode*)
Sets the limit switch parameters.

See *get_device_unit_from_real_value* for converting from a real_value to a device_unit.

Parameters

- **cw_lim** (*enums.MOT_LimitSwitchModes*) – The clockwise hardware limit mode as a *enums.MOT_LimitSwitchModes* enum value or member name.
- **ccw_lim** (*enums.MOT_LimitSwitchModes*) – The anticlockwise hardware limit mode as a *enums.MOT_LimitSwitchModes* enum value or member name.
- **cw_pos** (int) – The position of the clockwise software limit in DeviceUnits (see manual).
- **ccw_pos** (int) – The position of the anticlockwise software limit in DeviceUnits (see manual).

- **soft_limit_mode** (*enums.MOT_LimitSwitchSWModes*) – The soft limit mode as a *enums.MOT_LimitSwitchSWModes* enum value or member name.

Raises ConnectionError – If not successful.

set_limit_switch_params_block(*cw_limit*, *ccw_limit*, *cw_pos*, *ccw_pos*, *mode*)
Set the limit switch parameters.

See *get_device_unit_from_real_value* for converting from a real_value to a device_unit.

Parameters

- **cw_limit** (*enums.MOT_LimitSwitchModes*) – The clockwise hardware limit as a *enums.MOT_LimitSwitchModes* enum value or member name.
- **ccw_limit** (int) – The anticlockwise hardware limit as a *enums.MOT_LimitSwitchModes* enum value or member name.
- **cw_pos** (int) – The position of clockwise software limit in DeviceUnits.
- **ccw_pos** (int) – The position of anticlockwise software limit in DeviceUnits.
- **mode** (*enums.MOT_LimitSwitchSWModes*) – Actions to take when software limit is detected as a *enums.MOT_LimitSwitchSWModes* enum value or member name.

Raises ConnectionError – If not successful.

set_limits_software_approach_policy(*policy*)
Sets the software limits mode.

Parameters **policy** (*enums.MOT_LimitsSoftwareApproachPolicy*) – The soft limit mode as a *enums.MOT_LimitsSoftwareApproachPolicy* enum value or member name.

set_motor_params(*steps_per_rev*, *gear_box_ratio*, *pitch*)
Sets the motor stage parameters.

Deprecated: calls *set_motor_params_ext()*

These parameters, when combined, define the stage motion in terms of RealWorldUnits [millimeters or degrees]. The real-world unit is defined from steps_per_rev * gear_box_ratio / pitch.

See *get_real_value_from_device_unit* for converting from a device_unit to a real_value.

Parameters

- **steps_per_rev** (float) – The steps per revolution.
- **gear_box_ratio** (float) – The gear box ratio.
- **pitch** (float) – The pitch.

Raises ConnectionError – If not successful.

set_motor_params_ext(*steps_per_rev*, *gear_box_ratio*, *pitch*)
Sets the motor stage parameters.

These parameters, when combined, define the stage motion in terms of `RealWorldUnits` [millimeters or degrees]. The real-world unit is defined from `steps_per_rev * gear_box_ratio / pitch`.

See *get_real_value_from_device_unit* for converting from a `device_unit` to a `real_value`.

Parameters

- **steps_per_rev** (`float`) – The steps per revolution.
- **gear_box_ratio** (`float`) – The gear box ratio.
- **pitch** (`float`) – The pitch.

Raises `ConnectionError` – If not successful.

set_motor_travel_limits(*min_position*, *max_position*)
Sets the motor stage min and max position.

These define the range of travel for the stage.

See *get_real_value_from_device_unit* for converting from a `device_unit` to a `real_value`.

Parameters

- **min_position** (`float`) – The minimum position in `RealWorldUnits` [millimeters or degrees].
- **max_position** (`float`) – The maximum position in `RealWorldUnits` [millimeters or degrees].

Raises `ConnectionError` – If not successful.

set_motor_travel_mode(*travel_mode*)
Set the motor travel mode.

Parameters **travel_mode** (*enums.MOT_TravelModes*) – The travel mode as a *enums.MOT_TravelModes* enum value or member name.

Raises `ConnectionError` – If not successful.

set_motor_velocity_limits(*max_velocity*, *max_acceleration*)
Sets the motor stage maximum velocity and acceleration.

See *get_real_value_from_device_unit* for converting from a `device_unit` to a `real_value`.

Parameters

- **max_velocity** (`float`) – The maximum velocity in `RealWorldUnits` [millimeters or degrees].
- **max_acceleration** (`float`) – The maximum acceleration in `RealWorldUnits` [millimeters or degrees].

Raises `ConnectionError` – If not successful.

set_move_absolute_position(*position*)
Sets the move absolute position.

See *get_device_unit_from_real_value* for converting from a `real_value` to a `device_unit`.

Parameters **position** (`int`) – The absolute position in `DeviceUnits` (see manual).

Raises `ConnectionError` – If not successful.

set_move_relative_distance(*distance*)
Sets the move relative distance.

See *get_device_unit_from_real_value* for converting from a `real_value` to a `device_unit`.

Parameters **distance** (`int`) – The relative position in `DeviceUnits` (see manual).

Raises `ConnectionError` – If not successful.

set_position_counter(*count*)
Set the position counter.

Setting the position counter will locate the current position. Setting the position counter will effectively define the home position of a motor.

See *get_device_unit_from_real_value* for converting from a `real_value` to a `device_unit`.

Parameters **count** (`int`) – The position counter in `DeviceUnits` (see manual).

Raises `ConnectionError` – If not successful.

set_potentiometer_params(*index*, *threshold*, *velocity*)
Sets the potentiometer parameters for the LTS.

See *get_device_unit_from_real_value* for converting from a `real_value` to a `device_unit`.

Parameters

- **index** (`int`) – The potentiometer index to be stored.
- **threshold** (`int`) – The potentiometer threshold, range 0 to 127.
- **velocity** (`int`) – The velocity in `DeviceUnits` for the current potentiometer threshold.

Raises

- `ConnectionError` – If not successful.
- `ValueError` – If the value of *threshold* is out of range.

set_potentiometer_params_block(*params*)
Set the potentiometer parameters.

Parameters **params** (*structs.MOT_PotentiometerSteps*) – The potentiometer parameters.

Raises ConnectionError – If not successful.

set_power_params(*rest*, *move*)
Sets the power parameters for the stepper motor.

Parameters

- **rest** (int) – Percentage of full power to give while not moving (0 - 100).
- **move** (int) – Percentage of full power to give while moving (0 - 100).

Raises ConnectionError – If not successful.

set_stage_axis_limits(*min_position*, *max_position*)
Sets the stage axis position limits.

See *get_device_unit_from_real_value* for converting from a real_value to a device_unit.

Parameters

- **min_position** (int) – The minimum position in DeviceUnits (see manual).
- **max_position** (int) – The maximum position in DeviceUnits (see manual).

Raises ConnectionError – If not successful.

set_trigger_switches(*indicator_bits*)
Sets the trigger switch bits.

Parameters **indicator_bits** (int) – Sets the 8 bits indicating action on trigger input and events to trigger electronic output.

Raises ConnectionError – If not successful.

set_vel_params(*max_velocity*, *acceleration*)
Sets the move velocity parameters.

See *get_device_unit_from_real_value* for converting from a real_value to a device_unit.

Parameters

- **max_velocity** (int) – The maximum velocity in DeviceUnits (see manual).
- **acceleration** (int) – The acceleration in DeviceUnits (see manual).

Raises ConnectionError – If not successful.

set_vel_params_block(*min_velocity*, *max_velocity*, *acceleration*)
Set the move velocity parameters.

Parameters

- **min_velocity** (int) – The minimum velocity.
- **max_velocity** (int) – The maximum velocity.
- **acceleration** (int) – The acceleration.

Raises ConnectionError – If not successful.

start_polling(*milliseconds*)
Starts the internal polling loop.

This function continuously requests position and status messages.

Parameters **milliseconds** (int) – The polling rate, in milliseconds.

Raises ConnectionError – If not successful.

stop_immediate()
Stop the current move immediately (with risk of losing track of position).

Raises ConnectionError – If not successful.

stop_polling()
Stops the internal polling loop.

stop_profiled()
Stop the current move using the current velocity profile.

Raises ConnectionError – If not successful.

time_since_last_msg_received()
Gets the time, in milliseconds, since tha last message was received.

This can be used to determine whether communications with the device is still good.

Returns int – The time, in milliseconds, since the last message was received.

wait_for_message()
Wait for next Message Queue item. See *messages*.

Returns

- int – The message type.
- int – The message ID.
- int – The message data.

Raises ConnectionError – If not successful.

msl.equipment.resources.thorlabs.kinesis.kcube_solenoid module

This module provides all the functionality required to control a KCube Solenoid (KSC101).

class msl.equipment.resources.thorlabs.kinesis.kcube_solenoid.**KCubeSolenoid**(*record*)
Bases: *msl.equipment.resources.thorlabs.kinesis.motion_control.MotionControl*

A wrapper around Thorlabs.MotionControl.KCube.Solenoid

Parameters **record** (*EquipmentRecord*) – An equipment record from an **Equipment-Register** *Database*.

check_connection()
Check connection.

Returns bool – Whether the USB is listed by the FTDI controller.

clear_message_queue()
Clears the device message queue.

close()
Disconnect and close the device.

enable_last_msg_timer(*enable*, *msg_timeout=0*)
Enables the last message monitoring timer.

This can be used to determine whether communications with the device is still good.

Parameters

- **enable** (bool) – True to enable monitoring otherwise False to disable.
- **msg_timeout** (int) – The last message error timeout in ms. Set to 0 to disable.

get_cycle_params()
Gets the cycle parameters.

Returns

- int – The *On Time* parameter. Range 250 to 100,000,000 in steps of 1 milliseconds (0.250s to 10,000s).
- int – The *Off Time* parameter. Range 250 to 100,000,000 in steps of 1 milliseconds (0.250s to 10,000s).
- int – The *Number of Cycles* parameter. Range 0 to 1000,000 where 0 represents unlimited.

Raises ConnectionError – If not successful.

get_cycle_params_block()
Get the cycle parameters.

Returns *structs.SC_CycleParameters* – The cycle parameters.

Raises ConnectionError – If not successful.

get_digital_outputs()
Gets the digital output bits.

Returns int – Bit mask of states of the 4 digital output pins.

get_hardware_info()
Gets the hardware information from the device.

Returns *structs.TLI_HardwareInformation* – The hardware information.

Raises ConnectionError – If not successful.

get_hardware_info_block()
Gets the hardware information in a block.

Returns *structs.TLI_HardwareInformation* – The hardware information.

Raises ConnectionError – If not successful.

get_hub_bay()
Gets the hub bay number this device is fitted to.

Returns int – Either the number, or 0x00 if unknown, or 0xff if not on a hub.

get_led_switches()
Get the LED indicator bits on cube.

Returns int – Sum of: 8 to indicate moving 2 to indicate end of track and 1 to flash on identify command.

get_mmi_params()
Get the MMI Parameters for the KCube Display Interface.

Deprecated: calls *get_mmi_params_ext()*

Returns

- int – The display intensity, range 0 to 100%.
- int – The display timeout, range 0 to 480 in minutes (0 is off, otherwise the inactivity period before dimming the display).
- int – The display dimmed intensity, range 0 to 10 (after the timeout period the device display will dim).

Raises ConnectionError – If not successful.

get_mmi_params_ext()
Get the MMI Parameters for the KCube Display Interface.

Returns

- int – The display intensity, range 0 to 100%.
- int – The display timeout, range 0 to 480 in minutes (0 is off, otherwise the inactivity period before dimming the display).
- int – The display dimmed intensity, range 0 to 10 (after the timeout period the device display will dim).

Raises ConnectionError – If not successful.

get_next_message()
Get the next Message Queue item. See *messages*.

Returns

- int – The message type.
- int – The message ID.
- int – The message data.

Raises ConnectionError – If not successful.

get_operating_mode()
Gets the Operating Mode.

Returns *enums.SC_OperatingModes* – The current operating mode.

get_operating_state()
Gets the current operating state.

Returns *enums.SC_OperatingStates* – The current operating state.

get_software_version()
Gets version number of the device software.

Returns str – The device software version.

get_solenoid_state()
Gets the current solenoid state.

Returns *enums.SC_SolenoidStates* – The current solenoid state.

get_status_bits()
Get the current status bits.

This returns the latest status bits received from the device. To get new status bits, use *request_status()* or use the polling function, *start_polling()*

Returns int – The status bits from the device.

get_trigger_config_params()
Get the Trigger Configuration Parameters.

Returns

- *enums.KSC_TriggerPortMode* – The trigger 1 mode.
- *enums.KSC_TriggerPortPolarity* – The trigger 1 polarity.
- *enums.KSC_TriggerPortMode* – The trigger 2 mode.
- *enums.KSC_TriggerPortPolarity* – The trigger 2 polarity.

Raises ConnectionError – If not successful.

get_trigger_config_params_block()
Gets the trigger configuration parameters block.

Returns *structs.KSC_TriggerConfig* – Options for controlling the trigger configuration.

Raises ConnectionError – If not successful.

has_last_msg_timer_overrun()
Queries if the time since the last message has exceeded the lastMsgTimeout set by *enable_last_msg_timer()*.

This can be used to determine whether communications with the device is still good.

Returns bool – True if last message timer has elapsed or False if monitoring is not enabled or if time of last message received is less than lastMsgTimeout.

identify()
Sends a command to the device to make it identify itself.

load_settings()
Update device with stored settings.

The settings are read from ThorlabsDefaultSettings.xml, which gets created when the Kinesis software is installed.

Raises ConnectionError – If not successful.

message_queue_size()
Gets the size of the message queue.

Returns int – The number of messages in the queue.

open()
Open the device for communication.

Raises ConnectionError – If not successful.

persist_settings()
Persist the devices current settings.

Raises ConnectionError – If not successful.

polling_duration()
Gets the polling loop duration.

Returns int – The time between polls in milliseconds or 0 if polling is not active.

register_message_callback(*callback*)
Registers a callback on the message queue.

Parameters **callback** (*callbacks.MotionControlCallback*) – A function to be called whenever messages are received.

request_cycle_params()
Requests the cycle parameters.

Raises ConnectionError – If not successful.

request_digital_outputs()
Requests the digital output bits.

Raises ConnectionError – If not successful.

request_hub_bay()
Requests the hub bay number this device is fitted to.

Raises ConnectionError – If not successful.

request_led_switches()
Requests the LED indicator bits on the cube.

Raises ConnectionError – If not successful.

request_mmi_params()
Requests the MMI Parameters for the KCube Display Interface.

Raises ConnectionError – If not successful.

request_operating_mode()
Requests the operating mode.

Raises ConnectionError – If not successful.

request_operating_state()
Requests the operating state.

Raises ConnectionError – If not successful.

request_settings()
Requests that all settings are download from device.

This function requests that the device upload all it's settings to the DLL.

Raises ConnectionError – If not successful.

request_status()
Requests the status from the device.

This needs to be called to get the device to send it's current status bits. Note, this is called automatically if Polling is enabled for the device using *start_polling()*.

Raises ConnectionError – If not successful.

request_status_bits()
Request the status bits which identify the current motor state.

This needs to be called to get the device to send it's current status bits. Note, this is called automatically if Polling is enabled for the device using *start_polling()*.

Raises ConnectionError – If not successful.

request_trigger_config_params()
Requests the Trigger Configuration Parameters.

Raises ConnectionError – If not successful.

set_cycle_params(*on_time*, *off_time*, *num_cycles*)
Sets the cycle parameters.

Parameters

- **on_time** (int) – The On Time parameter. Range 250 to 100,000,000 in steps of 1 milliseconds (0.250s to 10,000s).
- **off_time** (int) – The Off Time parameter. Range 250 to 100,000,000 in steps of 1 milliseconds (0.250s to 10,000s).
- **num_cycles** (int) – The Number of Cycles parameter Range 0 to 1,000,000 where 0 represent unlimited.

Raises

- ValueError – If any of the input parameters are out of range.
- ConnectionError – If not successful.

set_cycle_params_block(*cycle_params*)
Sets the cycle parameters.

Parameters **cycle_params** (*structs.SC_CycleParameters*) – The new cycle parameters.

Raises ConnectionError – If not successful.

set_digital_outputs(*outputs_bits*)
Sets the digital output bits.

Parameters **outputs_bits** (int) – Bit mask to set the states of the 4 digital output pins.

Raises ConnectionError – If not successful.

set_led_switches(*led_switches*)
Set the LED indicator bits on the cube.

Parameters **led_switches** (int) – Sum of: 8 to indicate moving 2 to indicate end of track and 1 to flash on identify command.

Raises ConnectionError – If not successful.

set_mmi_params(*display_intensity*)
Set the MMI Parameters for the KCube Display Interface.

Deprecated: superceded by *set_mmi_params_ext()*

Parameters **display_intensity** (int) – The display intensity, range 0 to 100%.

Raises

- ValueError – If the value of *display_intensity* is out of range.
- ConnectionError – If not successful.

set_mmi_params_ext(*intensity*, *timeout*, *dim_intensity*)
Set the MMI Parameters for the KCube Display Interface.

Parameters

- **intensity** (int) – The display intensity, range 0 to 100%.
- **timeout** (int) – The display timeout, range 0 to 480 in minutes (0 is off, otherwise the inactivity period before dimming the display).
- **dim_intensity** (int) – The display dimmed intensity, range 0 to 10 (after the timeout period the device display will dim).

Raises

- ValueError – If any of the input parameters are out of range.
- ConnectionError – If not successful.

set_operating_mode(*mode*)
Sets the operating mode.

Parameters **mode** (*enums.SC_OperatingModes*) – The required operating mode as a *SC_OperatingModes* enum value or member name.

Raises ConnectionError – If not successful.

set_operating_state(*state*)
Sets the operating state.

Parameters **state** (*enums.SC_OperatingStates*) – The required operating state as a *SC_OperatingStates* enum value or member name.

Raises ConnectionError – If not successful.

set_trigger_config_params(*mode1*, *polarity1*, *mode2*, *polarity2*)

Parameters

- **mode1** (*enums.KSC_TriggerPortMode*) – The trigger 1 mode as a *KSC_TriggerPortMode* enum value or member name.
- **polarity1** (*enums.KSC_TriggerPortPolarity*) – The trigger 1 polarity as a *KSC_TriggerPortPolarity* enum value or member name.

- **mode2** (*enums.KSC_TriggerPortMode*) – The trigger 2 mode as a *KSC_TriggerPortMode* enum value or member name.
- **polarity2** (*enums.KSC_TriggerPortPolarity*) – The trigger 2 polarity as a *KSC_TriggerPortPolarity* enum value or member name.

Raises ConnectionError – If not successful.

set_trigger_config_params_block(*trigger_config_params*)
Sets the trigger configuration parameters block.

Parameters **trigger_config_params** (*structs.KSC_TriggerConfig*) – Options for controlling the trigger configuration.

Raises ConnectionError – If not successful.

start_polling(*milliseconds*)
Starts the internal polling loop.

This function continuously requests position and status messages.

Parameters **milliseconds** (int) – The polling rate, in milliseconds.

Raises ConnectionError – If not successful.

stop_polling()
Stops the internal polling loop.

time_since_last_msg_received()
Gets the time, in milliseconds, since tha last message was received.

This can be used to determine whether communications with the device is still good.

Returns int – The time, in milliseconds, since the last message was received.

wait_for_message()
Wait for next Message Queue item. See *messages*.

Returns

- int – The message type.
- int – The message ID.
- int – The message data.

Raises ConnectionError – If not successful.

msl.equipment.resources.thorlabs.kinesis.messages module

Device Message Queue defined in Thorlabs.MotionControl.C_API.

The device message queue allows the internal events raised by the device to be monitored by the DLLs owner.

The device raises many different events, usually associated with a change of state.

These messages are temporarily stored in the DLL and can be accessed using the appropriate message functions.

msl.equipment.resources.thorlabs.kinesis.motion_control module

Base **Thorlabs.MotionControl** class.

msl.equipment.resources.thorlabs.kinesis.motion_control.**device_manager**()
Returns a reference to the DeviceManager library.

The `Thorlabs.MotionControl.DeviceManager.dll` library must be available on `os.environ['PATH']`.

Returns `ctypes.CDLL` – A reference to the library.

class msl.equipment.resources.thorlabs.kinesis.motion_control.**MotionControl**(*record*, *api_funct*

Bases: *msl.equipment.connection_msl.ConnectionSDK*

Base **Thorlabs.MotionControl** class.

Parameters

- **record** (*EquipmentRecord*) – An equipment record from an **Equipment-Register** *Database*.
- **api_function** (*api_functions*) – An API function list from *api_functions* that the subclass is a wrapper around.

Raises `ConnectionError` – If a connection to the device cannot be established.

Benchtop_Brushless_Motor = 73
Benchtop Brushless Motor device ID

Benchtop_NanoTrak = 22
Benchtop NanoTrak device ID

Benchtop_Piezo_1_Channel = 41
Benchtop Piezo 1-Channel device ID

Benchtop_Piezo_3_Channel = 71
Benchtop Piezo 3-Channel device ID

Benchtop_Stepper_Motor_1_Channel = 40
Benchtop Stepper Motor 1-Channel device ID

Benchtop_Stepper_Motor_3_Channel = 70
Benchtop Stepper Motor 3-Channel device ID

Filter_Flipper = 37
Filter Flipper device ID

Filter_Wheel = 47
Filter Wheel device ID

KCube_Brushless_Motor = 28
KCube Brushless Motor device ID

KCube_DC_Servo = 27
KCube DC Servo device ID

KCube_LaserSource = 56
KCube Laser Source device ID

KCube_Piezo = 29
KCube Piezo device ID

KCube_Solenoid = 68
KCube Solenoid device ID

KCube_Stepper_Motor = 26
KCube Stepper Motor device ID

Long_Travel_Stage = 45
Long Travel Stage device ID

Cage_Rotator = 55
Cage Rotator device ID

LabJack_490 = 46
LabJack 490 device ID

LabJack_050 = 49
LabJack 050 device ID

Modular_NanoTrak = 52
Modular NanoTrak device ID

Modular_Piezo = 51
Modular Piezo device ID

Modular_Stepper_Motor = 50
Modular Stepper Motor device ID

TCube_Brushless_Motor = 67
TCube Brushless Motor device ID

TCube_DC_Servo = 83
TCube DC Servo device ID

TCube_Inertial_Motor = 65
TCube Inertial Motor device ID

TCube_LaserSource = 86
TCube Laser Source device ID

TCube_LaserDiode = 64
TCube Laser Diode device ID

TCube_NanoTrak = 82
TCube NanoTrak device ID

TCube_Quad = 89
TCube Quad device ID

TCube_Solenoid = 85
TCube Solenoid device ID

TCube_Stepper_Motor = 80
TCube Stepper_Motor device ID

TCube_Strain_Gauge = 84
TCube Strain Gauge device ID

TCube_TEC = 87
TCube TEC device ID

SERIAL_NUMBER_BUFFER_SIZE = 500

errcheck_api(*result*, *func*, *args*)
The API function returns OK if the function call was successful.

errcheck_true(*result*, *func*, *args*)
The API function returns `True` if the function call was successful.

disconnect()
Disconnect and close the device.

static **build_device_list**()
Build the device list.

This function builds an internal collection of all devices found on a USB port that are not currently open.

Note: If a device is open, it will not appear in the list until the device has been closed.

Raises `ConnectionError` – If the device list cannot be built.

static **get_device_list_size**()
`int`: The number of devices in the device list.

static **get_device_list**(**device_ids*)
Get the contents of the device list which match the supplied device IDs.

Parameters **device_ids** (`int`) – A sequence of device ID's.

Returns `list` of `str` – A list of device serial numbers for the specified device ID(s).

Raises `ConnectionError` – If there was an error getting the device list.

static **get_device_info**(*serial_number*)
Get the device information from a USB port.

The device info is read from the USB port not from the device itself.

Parameters **serial_number** (`str`) – The serial number of the device.

Returns *structs.TLI_DeviceInfo* – A DeviceInfo structure.

Raises `ConnectionError` – If there was an error getting the device information.

static **to_version**(*dword*)
Convert the firmware or software number to a string.

The number is made up of 4-byte parts.

See the *get_firmware_version()* or the *get_software_version()* method of the appropriate Thorlabs MotionControl subclass.

Parameters **dword** (`int`) – The firmware or software number.

Returns `str` – The string representation of the version number.

static convert_message(*msg_type*, *msg_id*, *msg_data*)
Converts the message into a string representation.

See the *get_next_message()* or the *wait_for_message()* method of the appropriate Thorlabs MotionControl subclass.

Parameters

- **msg_type** (int) – The message type defines the device type which raised the message.
- **msg_id** (int) – The message ID for the *msg_type*.
- **msg_data** (int) – The message data.

Returns str – The message as a string, with the type, id and data separated by a semicolon.

msl.equipment.resources.thorlabs.kinesis.structs module

Structs defined in the Thorlabs Kinesis software v1.11.0

class msl.equipment.resources.thorlabs.kinesis.structs.**TLI_DeviceInfo**
Bases: _ctypes.Structure

PID
Structure/Union member

description
Structure/Union member

isCustomType
Structure/Union member

isKnownType
Structure/Union member

isLaser
Structure/Union member

isPiezoDevice
Structure/Union member

isRack
Structure/Union member

maxChannels
Structure/Union member

motorType
Structure/Union member

serialNo
Structure/Union member

typeID
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**TLI_HardwareInformation**
Bases: _ctypes.Structure

deviceDependantData
Structure/Union member

firmwareVersion
Structure/Union member

hardwareVersion
Structure/Union member

modelNumber
Structure/Union member

modificationState
Structure/Union member

notes
Structure/Union member

numChannels
Structure/Union member

serialNumber
Structure/Union member

type
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**MOT_VelocityParameters**
Bases: _ctypes.Structure

acceleration
Structure/Union member

maxVelocity
Structure/Union member

minVelocity
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**MOT_JogParameters**
Bases: _ctypes.Structure

mode
Structure/Union member

stepSize
Structure/Union member

stopMode
Structure/Union member

velParams
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**MOT_HomingParameters**
Bases: _ctypes.Structure

direction
Structure/Union member

limitSwitch
Structure/Union member

offsetDistance
Structure/Union member

velocity
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**MOT_VelocityProfileParamete**
Bases: _ctypes.Structure

jerk
Structure/Union member

lastNotUsed
Structure/Union member

mode
Structure/Union member

notUsed
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**MOT_StageAxisParameters**
Bases: _ctypes.Structure

axisID
Structure/Union member

countsPerUnit
Structure/Union member

maxAcceleration
Structure/Union member

maxDecceleration
Structure/Union member

maxPosition
Structure/Union member

maxVelocity
Structure/Union member

minPosition
Structure/Union member

partNumber
Structure/Union member

reserved1
Structure/Union member

reserved2
Structure/Union member

reserved3
Structure/Union member

reserved4
Structure/Union member

reserved5
Structure/Union member

reserved6
Structure/Union member

reserved7
Structure/Union member

reserved8
Structure/Union member

serialNumber
Structure/Union member

stageID
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**MOT_JoystickParameters**
Bases: _ctypes.Structure

directionSense
Structure/Union member

highGearAcceleration
Structure/Union member

highGearMaxVelocity
Structure/Union member

lowGearAcceleration
Structure/Union member

lowGearMaxVelocity
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**MOT_BrushlessPositionLoopPa**
Bases: _ctypes.Structure

accelerationFeedForward
Structure/Union member

derivativeRecalculationTime
Structure/Union member

differentialGain
Structure/Union member

factorForOutput
Structure/Union member

integralGain
Structure/Union member

integralLimit
Structure/Union member

lastNotUsed
Structure/Union member

notUsed
Structure/Union member

positionErrorLimit
Structure/Union member

proportionalGain
Structure/Union member

velocityFeedForward
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**MOT_BrushlessTrackSettlePara**
Bases: _ctypes.Structure

lastNotUsed
Structure/Union member

maxTrackingError
Structure/Union member

notUsed
Structure/Union member

settledError
Structure/Union member

time
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**MOT_BrushlessCurrentLoopPara**
Bases: _ctypes.Structure

deadErrorBand
Structure/Union member

feedForward
Structure/Union member

integralGain
Structure/Union member

integralLimit
Structure/Union member

lastNotUsed
Structure/Union member

notUsed
Structure/Union member

phase
Structure/Union member

proportionalGain
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**MOT_BrushlessElectricOutputI**
Bases: _ctypes.Structure

continuousCurrentLimit
Structure/Union member

excessEnergyLimit
Structure/Union member

lastNotUsed
Structure/Union member

motorSignalBias
Structure/Union member

motorSignalLimit
Structure/Union member

notUsed
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**MOT_LimitSwitchParameters**
Bases: _ctypes.Structure

anticlockwiseHardwareLimit
Structure/Union member

anticlockwisePosition
Structure/Union member

clockwiseHardwareLimit
Structure/Union member

clockwisePosition
Structure/Union member

softLimitMode
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**MOT_PowerParameters**
Bases: _ctypes.Structure

movePercentage
Structure/Union member

restPercentage
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**MOT_DC_PIDParameters**
Bases: _ctypes.Structure

differentialGain
Structure/Union member

integralGain
Structure/Union member

integralLimit
Structure/Union member

parameterFilter
Structure/Union member

proportionalGain
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**BNT_IO_Settings**
Bases: _ctypes.Structure

BNCtriggerOrLowVoltageOut
Structure/Union member

amplifierCurrentLimit
Structure/Union member

amplifierLowPassFilter
Structure/Union member

channel
Structure/Union member

feedbackSignal
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**NT_HVComponent**
Bases: _ctypes.Structure

horizontalComponent
Structure/Union member

verticalComponent
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**NT_CircleParameters**
Bases: _ctypes.Structure

algorithmAdjustment
Structure/Union member

diameter
Structure/Union member

maxDiameter
Structure/Union member

minDiameter
Structure/Union member

mode
Structure/Union member

samplesPerRevolution
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**NT_CircleDiameterLUT**
Bases: _ctypes.Structure

LUTdiameter
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**NT_TIARangeParameters**
Bases: _ctypes.Structure

changeToOddOrEven
Structure/Union member

downLimit
Structure/Union member

mode
Structure/Union member

newRange
Structure/Union member

settleSamples
Structure/Union member

upLimit
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**NT_LowPassFilterParameters**
Bases: _ctypes.Structure

param1
Structure/Union member

param2
Structure/Union member

param3
Structure/Union member

param4
Structure/Union member

param5
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**NT_TIAReading**
Bases: _ctypes.Structure

absoluteReading
Structure/Union member

relativeReading
Structure/Union member

selectedRange
Structure/Union member

underOrOverRead
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**NT_IOSettings**
Bases: _ctypes.Structure

lowVoltageOutRange
Structure/Union member

lowVoltageOutputRoute
Structure/Union member

notYetInUse
Structure/Union member

unused
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**NT_GainParameters**
Bases: _ctypes.Structure

controlMode
Structure/Union member

gain
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**PZ_FeedbackLoopConstants**
Bases: _ctypes.Structure

integralTerm
Structure/Union member

proportionalTerm
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**PZ_LUTWaveParameters**
Bases: _ctypes.Structure

LUTValueDelay
Structure/Union member

cycleLength
Structure/Union member

mode
Structure/Union member

numCycles
Structure/Union member

numOutTriggerRepeat
Structure/Union member

outTriggerDuration
Structure/Union member

outTriggerStart
Structure/Union member

postCycleDelay
Structure/Union member

preCycleDelay
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**PPC_PIDConsts**
Bases: _ctypes.Structure

PIDConstsD
Structure/Union member

PIDConstsDFc
Structure/Union member

PIDConstsI
Structure/Union member

PIDConstsP
Structure/Union member

PIDDerivFilterOn
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**PPC_NotchParams**
Bases: _ctypes.Structure

filter1Fc
Structure/Union member

filter1Q
Structure/Union member

filter2Fc
Structure/Union member

filter2Q
Structure/Union member

filterNo
Structure/Union member

notchFilter1On
Structure/Union member

notchFilter2On
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**PPC_IOSettings**
Bases: _ctypes.Structure

FPBrightness
Structure/Union member

controlSrc
Structure/Union member

feedbackSrc
Structure/Union member

monitorOPBandwidth
Structure/Union member

monitorOPSig
Structure/Union member

reserved1
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**MOT_PIDLoopEncoderParams**
Bases: _ctypes.Structure

PIDOutputLimit
Structure/Union member

PIDTolerance
Structure/Union member

differentialGain
Structure/Union member

integralGain
Structure/Union member

loopMode
Structure/Union member

proportionalGain
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**FF_IOSettings**
Bases: _ctypes.Structure

ADCspeedValue
Structure/Union member

digIO1OperMode
Structure/Union member

digIO1PulseWidth
Structure/Union member

digIO1SignalMode
Structure/Union member

digIO2OperMode
Structure/Union member

digIO2PulseWidth
Structure/Union member

digIO2SignalMode
Structure/Union member

reserved1
Structure/Union member

reserved2
Structure/Union member

transitTime
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**MOT_ButtonParameters**
Bases: _ctypes.Structure

buttonMode
Structure/Union member

leftButtonPosition
Structure/Union member

rightButtonPosition
Structure/Union member

timeout
Structure/Union member

unused
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**MOT_PotentiometerStep**
Bases: _ctypes.Structure

thresholdDeflection
Structure/Union member

velocity
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**MOT_PotentiometerSteps**
Bases: _ctypes.Structure

potentiometerStepParameters
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**KMOT_MMIParams**
Bases: _ctypes.Structure

DisplayDimIntensity
Structure/Union member

DisplayIntensity
Structure/Union member

DisplayTimeout
Structure/Union member

JoystickAcceleration
Structure/Union member

JoystickDirectionSense
Structure/Union member

JoystickMaxVelocity
Structure/Union member

JoystickMode
Structure/Union member

PresetPos1
Structure/Union member

PresetPos2
Structure/Union member

reserved
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**KMOT_TriggerConfig**
Bases: _ctypes.Structure

Trigger1Mode
Structure/Union member

Trigger1Polarity
Structure/Union member

Trigger2Mode
Structure/Union member

Trigger2Polarity
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**KMOT_TriggerParams**
Bases: _ctypes.Structure

CycleCount
Structure/Union member

TriggerIntervalFwd
Structure/Union member

TriggerIntervalRev
Structure/Union member

TriggerPulseCountFwd
Structure/Union member

TriggerPulseCountRev
Structure/Union member

TriggerPulseWidth
Structure/Union member

TriggerStartPositionFwd
Structure/Union member

TriggerStartPositionRev
Structure/Union member

reserved
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**KLS_MMIParams**
Bases: _ctypes.Structure

displayIntensity
Structure/Union member

reserved
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**KLS_TrigIOParams**
Bases: _ctypes.Structure

mode1
Structure/Union member

mode2
Structure/Union member

polarity1
Structure/Union member

polarity2
Structure/Union member

power1
Structure/Union member

power2
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**TPZ_IOSettings**
Bases: _ctypes.Structure

class msl.equipment.resources.thorlabs.kinesis.structs.**KPZ_MMIParams**
Bases: _ctypes.Structure

DisplayDimIntensity
Structure/Union member

DisplayIntensity
Structure/Union member

DisplayTimeout
Structure/Union member

JoystickDirectionSense
Structure/Union member

JoystickMode
Structure/Union member

PresetPos1
Structure/Union member

PresetPos2
Structure/Union member

VoltageAdjustRate
Structure/Union member

VoltageStep
Structure/Union member

reserved
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**KPZ_TriggerConfig**
Bases: _ctypes.Structure

Trigger1Mode
Structure/Union member

Trigger1Polarity
Structure/Union member

Trigger2Mode
Structure/Union member

Trigger2Polarity
Structure/Union member

reserved
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**QD_LoopParameters**
Bases: _ctypes.Structure

differentialGain
Structure/Union member

integralGain
Structure/Union member

lowPassFilterCutOffFreq
Structure/Union member

lowPassFilterEnabled
Structure/Union member

notchFilterCenterFrequency
Structure/Union member

notchFilterEnabled
Structure/Union member

notchFilterQ
Structure/Union member

proportionalGain
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**QD_PIDParameters**
Bases: _ctypes.Structure

differentialGain
Structure/Union member

integralGain
Structure/Union member

proportionalGain
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**QD_LowPassFilterParameters**
Bases: _ctypes.Structure

lowPassFilterCutOffFreq
Structure/Union member

lowPassFilterEnabled
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**QD_NotchFilterParameters**
Bases: _ctypes.Structure

notchFilterCenterFrequency
Structure/Union member

notchFilterEnabled
Structure/Union member

notchFilterQ
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**QD_PositionDemandParameters**
Bases: _ctypes.Structure

lowVoltageOutputRoute
Structure/Union member

maxXdemand
Structure/Union member

maxYdemand
Structure/Union member

minXdemand
Structure/Union member

minYdemand
Structure/Union member

openLoopOption
Structure/Union member

xFeedbackSignedGain
Structure/Union member

yFeedbackSignedGain
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**QD_Position**
Bases: _ctypes.Structure

x
Structure/Union member

y
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**QD_Readings**
Bases: _ctypes.Structure

demandedPos
Structure/Union member

posDifference
Structure/Union member

sum
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**QD_KPA_TrigIOConfig**
Bases: _ctypes.Structure

trig1DiffThreshold
Structure/Union member

trig1Mode
Structure/Union member

trig1Polarity
Structure/Union member

trig1SumMax
Structure/Union member

trig1SumMin
Structure/Union member

trig2DiffThreshold
Structure/Union member

trig2Mode
Structure/Union member

trig2Polarity
Structure/Union member

trig2SumMax
Structure/Union member

trig2SumMin
Structure/Union member

wReserved
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**QD_KPA_DigitalIO**
Bases: _ctypes.Structure

wDigOPs
Structure/Union member

wReserved
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**SC_CycleParameters**
Bases: _ctypes.Structure

closedTime
Structure/Union member

numCycles
Structure/Union member

openTime
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**KSC_MMIParams**
Bases: _ctypes.Structure

DisplayDimIntensity
Structure/Union member

DisplayIntensity
Structure/Union member

DisplayTimeout
Structure/Union member

reserved
Structure/Union member

unused
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**KSC_TriggerConfig**
Bases: _ctypes.Structure

Trigger1Mode
Structure/Union member

Trigger1Polarity
Structure/Union member

Trigger2Mode
Structure/Union member

Trigger2Polarity
Structure/Union member

reserved
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**TIM_DriveOPParameters**
Bases: _ctypes.Structure

class msl.equipment.resources.thorlabs.kinesis.structs.**TIM_JogParameters**
Bases: _ctypes.Structure

class msl.equipment.resources.thorlabs.kinesis.structs.**TIM_ButtonParameters**
Bases: _ctypes.Structure

class msl.equipment.resources.thorlabs.kinesis.structs.**TIM_Status**
Bases: _ctypes.Structure

class msl.equipment.resources.thorlabs.kinesis.structs.**TSG_IOSettings**
Bases: _ctypes.Structure

displayMode
Structure/Union member

forceCalibration
Structure/Union member

futureUse
Structure/Union member

hubAnalogOutput
Structure/Union member

notYetInUse
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.**TC_LoopParameters**
Bases: _ctypes.Structure

differentialGain
Structure/Union member

integralGain
Structure/Union member

proportionalGain
Structure/Union member

Submodules

msl.equipment.resources.thorlabs.fw102c module

Wrapper around Thorlabs **FilterWheel102.dll**, v4.0.0.

Thorlabs FW102C Series and FW212C Series Motorized Filter Wheels.

class msl.equipment.resources.thorlabs.fw102c.**FilterCount**
Bases: enum.IntEnum

The number of filter positions that the filter wheel has.

SIX = 6

TWELVE = 12

class msl.equipment.resources.thorlabs.fw102c.**SensorMode**
Bases: enum.IntEnum

Sensor modes of the filter wheel.

ON = 0

OFF = 1

class msl.equipment.resources.thorlabs.fw102c.**SpeedMode**
Bases: enum.IntEnum

Speed modes of the filter wheel.

SLOW = 0

FAST = 1

class msl.equipment.resources.thorlabs.fw102c.**TriggerMode**
Bases: enum.IntEnum

Trigger modes of the filter wheel.

INPUT = 0
Respond to an active-low pulse by advancing the position by 1

OUTPUT = 1
Generate an active-high pulse when the position changes

class msl.equipment.resources.thorlabs.fw102c.**FilterWheel102C**(*record*)
Bases: *msl.equipment.connection_msl.ConnectionSDK*

Wrapper around Thorlabs **FilterWheel102.dll**, v4.0.0.

A 64-bit version of the library can be download from here and it is located in **App-Notes_FW102C/LabVIEW/Thorlabs_FW102C/Library/FilterWheel102_win64.dll**.

The *record.connection.properties* dictionary for a FilterWheel102C device supports the following key-value pairs:

```
'port': str,  # mandatory, example 'COM3'
'baud_rate': int,  # optional, default is 115200
'timeout': int,  # optional, default is 10
```

Parameters **record** (*EquipmentRecord*) – An equipment record from an **Equipment-Register** *Database*.

Raises ConnectionError – If a connection to the filter wheel cannot be established.

close()
Close the opened COM port.

disconnect()
Close the opened COM port.

errcheck_code(*result*, *func*, *arguments*)
The SDK function returns OK if the function call was successful.

errcheck_negative(*result*, *func*, *arguments*)
The SDK function returns a positive number if the call was successful.

errcheck_non_zero(*result*, *func*, *arguments*)
The SDK function returns 0 if the call was successful.

get_acceleration()

Returns int – The current acceleration value of the filter wheel.

get_id()

Returns str – The id of the filter wheel.

get_max_velocity()

Returns int – The current maximum velocity value of the filter wheel.

get_min_velocity()

Returns int – The current minimum velocity value of the filter wheel.

get_ports()
List all the COM ports on the computer.

Returns dict of str – A dictionary where the keys are the port numbers, e.g. COM1, COM3, and the values are a description about each device connected to the port.

get_position()

Returns int – The current position of the filter wheel.

get_position_count()

Returns *FilterCount* – The number of filter positions that the filter wheel has.

get_sensor_mode()

Returns *SensorMode* – The current sensor mode of the filter wheel.

get_speed_mode()

Returns *SpeedMode* – The current speed mode of the filter wheel.

get_time_to_current_pos()

Returns int – The time from last position to current position.

get_trigger_mode()

Returns *TriggerMode* – The current trigger mode of the filter wheel.

is_open(*port*)
Check if the COM port is open.

Parameters **port** (str) – The port to be checked, e.g. 'COM3'.

Returns bool – True if the port is opened; False if the port is closed.

open(*port*, *baud_rate*, *timeout*)
Open a COM port for communication.

Parameters

- **port** (str) – The port to be opened, use the *get_ports()* function to get list of available ports.
- **baud_rate** (int) – The number of bits per second to use for the communication protocol.
- **timeout** (int) – Set the timeout value, in seconds.

save()
Save the current settings as the default settings on power up.

set_acceleration(*acceleration*)
Set the filter wheel's acceleration.

Parameters **acceleration** (int) – The filter wheel's acceleration value.

set_max_velocity(*maximum*)
Set the filter wheel's maximum velocity.

Parameters **maximum** (int) – The filter wheel's maximum velocity value.

set_min_velocity(*minimum*)
Set the filter wheel's minimum velocity.

Parameters **minimum** (int) – The filter wheel's minimum velocity value.

set_position(*position*)
Set the filter wheel's position.

Parameters **position** (int) – The position number to set the filter wheel to.

Raises ValueError – If the value of *position* is invalid.

set_position_count(*count*)
Set the filter wheel's position count.

This is the number of filter positions that the filter wheel has.

Parameters **count** (*FilterCount*) – The number of filters in the filter wheel as a *FilterCount* enum value or member name.

Raises ValueError – If the value of *count* is invalid.

set_sensor_mode(*mode*)
Set the filter wheel's sensor mode.

Parameters **mode** (*SensorMode*) – The filter wheel's sensor mode as a *SensorMode* enum value or member name.

Raises ValueError – If the value of *mode* is invalid.

set_speed_mode(*mode*)
Set the filter wheel's speed mode.

Parameters **mode** (*SpeedMode*) – The speed mode of the filter wheel as a *SpeedMode* enum value or member name.

Raises ValueError – If the value of *mode* is invalid.

set_trigger_mode(*mode*)
Set the filter wheel's trigger mode.

Parameters **mode** (*TriggerMode*) – The filter wheel's trigger mode as a *TriggerMode* enum value or member name.

Raises ValueError – If the value of *mode* is invalid.

Submodules

msl.equipment.resources.utils module

Utility functions/classes to help create modules in the **resources** package.

msl.equipment.resources.utils.**camelcase_to_underscore**(*text*)
Converts **CamelCaseText** to **camel_case_text**.

Parameters **text** (str) – The camel-case text to be converted.

Returns str – The *text* converted to lowercase and separated by underscores.

msl.equipment.resources.utils.**get_lines**(*header_path*, *remove_comments=True*)
Returns the lines in a C/C++ header file that are not empty.

Also strips the whitespace from each line and can optionally remove the all comments from the header file.

Parameters

- **header_path** (str) – The path to a C/C++ header file.
- **remove_comments** (bool) – Whether to remove the comments from each line.

Returns list of str – The list of lines in the header file.

class msl.equipment.resources.utils.**CHeader**(*header_path*, *remove_comments=True*)

Bases: object

Parses a C/C++ header file to determine the constants, enums, structs, callbacks and the function signatures.

Parameters

- **header_path** (str) – The path to the header file.
- **remove_comments** (bool) – Whether to remove the comments from the header file.

constants(*ignore_ifdef=True*)

Finds the **#define** statements that in a C/C++ header file.

Parameters **ignore_ifdef** (bool, optional) – Whether to ignore the #define statements in between the #ifdef and #endif statements.

Returns dict of str – A dictionary of all the #define constants (as strings).

enums()

Returns dict – The enums that are defined in the C/C++ header file. The value for each dictionary key is a tuple of *(the naming convention, the enum data type that is defined, a dict of name-value pairs)*.

structs()

Returns dict – The structs that are defined in the C/C++ header file.

callbacks()

Returns dict – The callbacks that are defined in the C/C++ header file.

functions(*regex*)

Returns the function signatures.

Parameters **regex** (str) – The regex must create 2 groups, *(return type, function name)*, and it must match the function declaration up until, but excluding, the '(' which begins the argument declarations.

For example,

If the function declarations are similar to `FILTERFLIPPERDLL_API unsigned int __cdecl FF_GetTransitTime(const char * serialNo);` then the value of *regex* could be `r'_API\s+([\w\s]+?)__cdecl\s+(\w+)'`

If the function declarations are similar to `PREF0 PREF1 PICO_STATUS PREF2 PREF3 (ps6000OpenUnit)(int16_t *handle, int8_t *serial);` then the value of *regex* could be `r'PREF0\s+PREF1\s+(\w+)\s+PREF2\s+PREF3\s+\((\w+)\)'`

Returns dict – The function signature. The key is the function name and the value is a list of [return type, [(argument data type, argument name), ...]].

get_lines()

Returns list of str – The lines in the C/C++ header file.

get_struct_imports()

Returns list of str – The list of structs that must be imported for the C/C++ functions.

Note: Must call *functions()* first.

---

static get_text_between_brackets(*lines*, *index*, *bracket1*, *bracket2*)
Get all the text (excluding comments) between two brackets.

Parameters

- **lines** (list of str) – A list of lines. Comes from *get_lines()*.
- **index** (int) – The current index in *lines*.
- **bracket1** (str) – One of {, (, or [
- **bracket2** (str) – One of },) or]

Returns

- str – The text between *bracket1* and *bracket2*.
- int – The current index in *lines*.

Submodules

msl.equipment.config module

Load a XML configuration file.

class msl.equipment.config.**Config**(*path*)
Bases: object

Load a XML configuration file.

This function is used to set the configuration constants to use for the Python runtime and it creates *EquipmentRecord*'s from **Equipment-Register** databases and *ConnectionRecord*'s from **Connection** databases.

MSL-Equipment configuration constants that can be defined in a configuration file:

Name	Example Values	Description
PyVISA_LIBRARY	@ni, @py, @sim, /path/to/lib@ni	The PyVISA backend library to use.
DEMO_MODE	true, false	Whether to open **all** connections in demo mode.
SDK_PATH	/path/to/SDKs, D:/data/SDKs	A path that contains SDK libraries. Accepts a *recursive="true"* attribute. Appends the path(s) to os.environ['PATH']

Also, the user is encouraged to define their own application-specific constants within the configuration file.

Example configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<msl>

    <!-- Application-specific constant to use as a warning if the␣
↪temperature of a device gets too hot -->
    <max_temperature units="C">60</max_temperature>

    <!-- Use PyVISA-py as the PyVISA library -->
    <PyVISA_LIBRARY>@py</PyVISA_LIBRARY>

    <!-- Open all connections in demo mode -->
    <DEMO_MODE>true</DEMO_MODE>

    <!-- Add a path to os.environ['PATH'] where SDK files are located␣
↪-->
    <SDK_PATH>I:\Photometry\SDKs</SDK_PATH>

    <!-- Recursively add SDK paths starting from a root path -->
    <SDK_PATH recursive="true">C:\Program Files\Thorlabs</SDK_PATH>

    <!-- The equipment that is being used to perform the measurement -
↪->
    <equipment alias="ref" manufacturer="Keysight" model="34465A"␣
↪serial="MY54506462"/>
    <equipment alias="scope" manufacturer="Pico Technologies" serial=
↪"DY135/055"/>
    <equipment alias="flipper" manufacturer="Thorlabs" model="MFF101/M
↪" serial="37871232"/>

    <!-- The database that contains the information required to␣
↪connect to the equipment -->
    <equipment_connections>
        <path>Z:\QUAL\Equipment\Equipment Register.xls</path>
        <sheet>Connections</sheet>
    </equipment_connections>

    <!-- The equipment-register database(s) -->
    <equipment_registers>
        <register section="P&amp;R">
            <path>Z:\QUAL\Equipment\Equipment Register.xls</path>
            <sheet>Equipment</sheet>
        </register>
        <register section="Electrical">
            <path>H:\Quality\Registers\Equipment.xls</path>
            <sheet>REG</sheet>
        </register>
        <register section="Time">
            <path>W:\Registers\Equip.csv</path>
        </register>
        <register section="Mass">
            <path>Y:\databases\equipment\equip-reg.txt</path>
        </register>
    </equipment_registers>

</msl>
```

Parameters **path** (str) – The path to a XML configuration file.

Raises

- FileNotFoundError – If *path* does not exist.
- ParseError – If the configuration file is invalid.

PyVISA_LIBRARY = '@ni'
str – The PyVISA backend library to use

DEMO_MODE = False
bool – Whether to open **all** connections in demo mode

SDK_PATH = []
list of str – Paths that were appended to os.environ['PATH']

path
str – The path to the configuration file.

root
Returns the root element (the first node) of the XML tree.

Returns Element – The root element.

database()

Returns *Database* – A reference to the equipment and connection records in the database(s) that are specified in the configuration file.

value(*tag*)
Gets the value associated with the specified *tag*.

Parameters **tag** (str) – The name of a XML tag in the configuration file.

Returns str or None – The value associated with the *tag* or None if the tag cannot be found.

msl.equipment.connection module

Base class for establishing a connection to the equipment.

class msl.equipment.connection.**Connection**(*record*)
Bases: object

All *Backend* Connection classes must have this class as the base class.

Do not instantiate this class directly. Use the factory method, *msl.equipment.factory.connect*, or the *record* object itself, *record.connect()*, to connect to the equipment.

Parameters **record** (*EquipmentRecord*) – An equipment record from an **Equipment-Register** *Database*.

equipment_record
EquipmentRecord – The equipment record, from an **Equipment-Register** *Database*, that is being used to establish the connection.

disconnect()
Disconnect from the equipment.

This method should be overridden in the *Connection* subclass if the subclass must implement tasks that need to be performed in order to safely disconnect from the equipment.

For example:

- to clean up system resources from memory (e.g., if using a manufacturer's SDK)
- to configure the equipment to be in a state that is safe for people working in the lab when the equipment is not in use

---

Note: This method gets called automatically when the *Connection* object gets destroyed.

---

raise_exception(*msg*)
Raise a ConnectionError exception.

Parameters **msg** (str) – The message to display when the exception is raised.

static **convert_to_enum**(*item*, *enum*, *prefix=''*, *to_upper=False*)
Convert *item* to an *enum* value.

Parameters

- **item** (int or str) – If str then the name of a *enum* member. If int then the value of a *enum* member.
- **enum** (IntEnum) – An integer-enum object.
- **prefix** (str, optional) – If *item* is a str, then *prefix* is included at the beginning of *item* before converting *item* to an *enum* value.
- **to_upper** (bool, optional) – If *item* is a str, then whether to change *item* to be upper case before converting *item* to an *enum* value.

Returns IntEnum – The *enum* value.

Raises

- TypeError – If *item* is not an int or str.
- ValueError – If *item* is not in *enum*.

static **log_debug**(*msg*, **args*, ***kwargs*)
Log a debug message.

Parameters **msg** (str) – The debug message to log.

static **log_info**(*msg*, **args*, ***kwargs*)
Log an info message.

Parameters **msg** (str) – The info message to log.

static **log_warning**(*msg*, **args*, ***kwargs*)
Log a warning message.

Parameters **msg** (str) – The warning message to log.

static **log_error**(*msg*, **args*, ***kwargs*)
Log an error message.

Parameters **msg** (str) – The error message to log.

static log_critical(*msg*, **args*, ***kwargs*)
Log a critical message.

Parameters **msg** (str) – The critical message to log.

msl.equipment.connection_demo module

Simulate a connection to the equipment.

class msl.equipment.connection_demo.**ConnectionDemo**(*record*, *cls*)
Bases: *msl.equipment.connection.Connection*

Simulate a connection to the equipment.

Establishing a connection in demo mode is useful when developing a program and the equipment is not physically connected to the computer.

A custom logging level is used for logging messages with a connection in demo mode. The logging.DEMO logging level is set to be between logging.INFO and logging.WARNING.

The returned data type is determined from the docstring of the called function. For example, if :rtype: int then an integer value is returned or if :rtype: int, float then an integer and a float value is returned. Although the expected data type is returned the value(s) of the returned object is randomly generated. The docstring must be in either the reStructuredText or NumPy format.

Do not instantiate this class directly. Use the factory method, *msl.equipment.factory.connect*, or the *record* object itself, *record.connect()*, to connect to the equipment in demo mode.

Parameters

- **record** (*EquipmentRecord*) – An equipment record from an **Equipment-Register** *Database*.
- **cls** (*Connection*) – A Connection class (that has **NOT** been instantiated).

disconnect()
Log a disconnection from the equipment.

msl.equipment.connection_msl module

Use MSL resources to establish a connection to the equipment.

class msl.equipment.connection_msl.**ConnectionSDK**(*record*, *libtype*)
Bases: *msl.equipment.connection.Connection*

Base class for equipment that use the SDK provided by the manufacturer for the connection.

The *record.connection.backend* value must be equal to *Backend.MSL* to use this class for the communication system. This is achieved by setting the value in the **Backend** field for a connection record in the **Connections** database to be **MSL**.

Do not instantiate this class directly. Use the factory method, *msl.equipment.factory.connect*, or the *record* object itself, *record.connect()*, to connect to the equipment.

Parameters

- **record** (*EquipmentRecord*) – An equipment record from an **Equipment-Register** *Database*.
- **libtype** (str or None) – The library type to use for the calling convention.

  The following values are allowed:

  - **'cdll'** – for a **__cdecl** library
  - **'windll'** or **'oledll'** – for a **__stdcall** library (Windows only)
  - **'net'** – for a **.NET** library

Raises IOError – If the Python wrapper class around the SDK cannot be found or if the shared library cannot be found.

sdk_path
str – The path to the shared library file.

sdk
The reference to the sdk library.

log_errcheck(*result*, *func*, *arguments*)
Convenience method for logging an errcheck when calling an SDK function.

class msl.equipment.connection_msl.**ConnectionMessageBased**(*record*)
Bases: *msl.equipment.connection.Connection*

Base class for equipment that use message based communication.

The *record.connection.backend* value must be equal to *Backend.MSL* to use this class for the communication system. This is achieved by setting the value in the **Backend** field for a connection record in the **Connections** database to be **MSL**.

Do not instantiate this class directly. Use the factory method, *msl.equipment.factory.connect*, or the *record* object itself, *record.connect()*, to connect to the equipment.

Parameters **record** (*EquipmentRecord*) – An equipment record from an **Equipment-Register** *Database*.

CR = '\r'
str – The carriage-return character

LF = '\n'
str – The line-feed character

encoding
str – The encoding that is used for *read()* and *write()* operations.

read_termination
str or None – The termination character sequence that is used for *read()* operations.

Reading stops when the equipment stops sending data (e.g., by setting appropriate bus lines) or the *read_termination* character sequence is detected.

write_termination
str – The termination character sequence that is appended to *write()* messages.

chunk_size
int – The number of bytes to be *read()*.

read_size
int – The number of bytes to be *read()*.

read(*size=None*)
Read a response from the equipment.

Returns str – The response from the equipment.

write(*message*)
Write (send) a message to the equipment.

Parameters **message** (str) – The message to write (send) to the equipment.

Returns int – The number of bytes written.

send(*message*)
Write (send) a message to the equipment.

Parameters **message** (str) – The message to write (send) to the equipment.

Returns int – The number of bytes written.

query(*message*, *delay=0.0*)
Convenience method for performing a *write()* followed by a *read()*.

Parameters

- **message** (str) – The message to write (send) to the equipment.
- **delay** (float) – The time delay, in seconds, to wait between *write()* and *read()* operations.

Returns str – The response from the equipment.

ask(*message*, *delay=0.0*)
Convenience method for performing a *write()* followed by a *read()*.

Parameters

- **message** (str) – The message to write (send) to the equipment.
- **delay** (float) – The time delay, in seconds, to wait between *write()* and *read()* operations.

Returns str – The response from the equipment.

class msl.equipment.connection_msl.**ConnectionSerial**(*record*)
Bases: *msl.equipment.connection_msl.ConnectionMessageBased*

Establish a connection through a Serial port.

The *record.connection.properties* dictionary for a Serial connection supports the following key-value pairs:

```
'read_termination': str or None
'write_termination': str or None
'read_size': int (the number of bytes to be read, must be > 0)
'encoding': str (e.g., 'ascii')
'baud_rate': int (e.g., 9600, 115200)
'data_bits': int (e.g., 5, 6, 7, 8)
'stop_bits': int or float (e.g., 1, 1.5, 2)
'parity': str (e.g., none, even, odd, mark, space)
'timeout': float or None (the read timeout value)
```

```
'write_timeout': float or None (the write timeout value)
'inter_byte_timeout': float or None (the inter-character timeout)
'exclusive': bool (set exclusive access mode, for POSIX only)
'xon_xoff': bool (enable software flow control)
'rts_cts': bool (enable hardware (RTS/CTS) flow control)
'dsr_dtr': bool (enable hardware (DSR/DTR) flow control)
```

serial
serial.Serial – The reference to the Serial object.

baud_rate
int – The baud rate setting.

data_bits
DataBits – The number of data bits.

stop_bits
StopBits – The stop bit setting.

parity
Parity – The parity setting.

write(*message*)
Write the given message over the serial port.

Parameters **message** (str) – The message to write.

Returns int – The number of bytes written to the serial port.

read(*size=None*)
Read *size* bytes from the serial port.

If a *timeout* is set it may return less characters as requested. With no *timeout* it will block until the requested number of bytes is read.

Parameters **size** (int) – The number of bytes to read. If None then read *read_size* bytes.

Returns bytes – The bytes read from the serial port.

disconnect()

msl.equipment.connection_pyvisa module

Use PyVISA as the backend to communicate with the equipment.

class msl.equipment.connection_pyvisa.**ConnectionPyVISA**(*record*)
Bases: *msl.equipment.connection.Connection*

Use PyVISA to establish a connection to the equipment.

The *record.connection.backend* value must be equal to *Backend.PyVISA* to use this class for the communication system. This is achieved by setting the value in the **Backend** field for a connection record in the **Connections** database to be **PyVISA**.

If you want to change the read_termination, write_termination and/or the encoding value for communication with the equipment then you can define, for exam-

ple, `read_termination=\n; write_termination=\n; encoding=utf-8` in the **Properties** field for a connection record in the **Connections** database.

Do not instantiate this class directly. Use the factory method, *msl.equipment.factory.connect*, or the *record* object itself, *record.connect()*, to connect to the equipment.

Parameters **record** (*EquipmentRecord*) – An equipment record from an **Equipment-Register** *Database*.

disconnect()
Calls `close()`.

static **resource_manager**(*visa_library=None*)
Return the PyVISA `ResourceManager`.

Only **one** Resource Manager session is created per Python runtime and therefore multiple calls to this function will return the same `ResourceManager` object.

Parameters **visa_library** (`VisaLibraryBase`, `str` or `None`) – The library to use for PyVISA. For example:

- `@ni` to use NI-VISA
- `@py` to use PyVISA-py
- `@sim` to use PyVISA-sim

If `None` then the *visa_library* value is read from a `CONFIG` variable. See `msl.equipment.config.load` for more details.

Returns `ResourceManager` – The PyVISA Resource Manager.

Raises

- `ValueError` – If the PyVISA backend wrapper cannot be found.
- `OSError` – If the VISA library cannot be found.

static **resource_pyclass**(*record*)
Find the PyVISA `Resource` that can open the *record*.

Parameters **record** (*EquipmentRecord* or *ConnectionRecord*) – An equipment or connection record from the *Database*.

Returns `Resource` – The appropriate PyVISA Resource class that can open the *record*.

msl.equipment.constants module

MSL-Equipment package constants.

class `msl.equipment.constants.`**Backend**
Bases: `enum.IntEnum`

The software backend to use for the communication system.

UNKNOWN = 0

MSL = 1

PyVISA = 2

class msl.equipment.constants.**MSLInterface**
Bases: enum.IntEnum

The interface to use for the communication system that transfers data between a computer and the equipment. Only used if *Backend.MSL* is chosen as the communication system.

NONE = 0

ASRL = 1

GPIB = 2

USB = 3

PXI = 4

VXI = 5

TCPIP = 6

SDK = 7

TCPIP_ASRL = 8

TCPIP_GPIB = 9

USB_ASRL = 10

USB_GPIB = 11

PROLOGIX_ENET = 12

PROLOGIX_USB = 13

class msl.equipment.constants.**Parity**
Bases: enum.Enum

The parity type to use for Serial communication.

NONE = 'N'

ODD = 'O'

EVEN = 'E'

MARK = 'M'

SPACE = 'S'

class msl.equipment.constants.**StopBits**
Bases: enum.Enum

The number of stop bits to use for Serial communication.

ONE = 1

ONE_POINT_FIVE = 1.5

TWO = 2

class msl.equipment.constants.**DataBits**
Bases: enum.IntEnum

The number of data bits to use for Serial communication.

FIVE = 5

SIX = 6

SEVEN = 7

EIGHT = 8

msl.equipment.database module

Load equipment and connection records (rows) from databases.

class msl.equipment.database.**Database**(*path*)
Bases: object

Create *EquipmentRecord*'s and *ConnectionRecord*'s from databases that are specified in a configuration file.

See msl.equipment.config.load for an example of a XML configuration file.

Parameters **path** (str) – The path to a XML configuration file.

Raises

- FileNotFoundError – If *path* does not exist.
- ParseError – If the configuration file is invalid.
- IOError – If an XML element content is invalid.
- AttributeError – If an <equipment> XML element is specified in the configuration file and it does not uniquely identify an equipment record in an **Equipment-Register** database.
- ValueError – If any of the values in the **Connections** database are invalid.

equipment
dict of *EquipmentRecord* – Equipment records that were listed as <equipment> XML elements in the configuration.

path
str – The path to the configuration file that contains the information about the **Equipment-Register** and **Connections** databases and the environment variables.

connections(***kwargs*)
Search the **Connections** database to find all connection records that match the specified criteria.

Parameters ****kwargs** – The argument names can be any of the *ConnectionRecord* property names. The comparison for the value is performed by regex, see re.

Returns list of *ConnectionRecord* – Connection records that match the search criteria.

Examples

```
>>> connections()
a list of all ConnectionRecords
```

```
>>> connections(manufacturer='H*P')
a list of all ConnectionRecords that have Hewlett Packard as the␣
↪manufacturer
```

```
>>> connections(address='GPIB*')
a list of all ConnectionRecords that use GPIB for the connection␣
↪bus
```

records(***kwargs*)

Search the **Equipment-Register** databases to find all equipment records that match the specified criteria.

Parameters ****kwargs** – The argument names can be any of the *EquipmentRecord* property names. The comparison for the value is performed by regex, see `re`.

Returns `list` of *EquipmentRecord* – Equipment records that match the search criteria.

Examples

```
>>> records()
a list of all EquipmentRecords
```

```
>>> records(manufacturer='H*P')
a list of all EquipmentRecords that have Hewlett Packard as the␣
↪manufacturer
```

```
>>> records(manufacturer='Agilent', model='3458A')
a list of all EquipmentRecords that are from Agilent and that␣
↪have the model number 3458A
```

```
>>> records(manufacturer='Agilent', model='3458A', serial=
↪'MY45046470')
a list of only one EquipmentRecord (if the equipment record␣
↪exists, otherwise an empty list)
```

```
>>> records(description='I-V Converter')
a list of all EquipmentRecords that contain 'I-V Converter' in␣
↪the description field
```

msl.equipment.factory module

Establish a connection to the equipment to send and receive messages.

msl.equipment.factory.**connect**(*record*, *demo=None*)

Factory function to establish a connection to the equipment.

Parameters

- **record** (*EquipmentRecord*) – An equipment record from an **Equipment-Register** *Database*.

- **demo** (bool or None) – Whether to simulate a connection to the equipment by opening a connection in demo mode. This allows you run your code if the equipment is not physically connected to the computer.

  If None then the *demo* value is read from a configuration variable. See *msl.equipment.config.Config* for more details.

Returns *Connection* – A *Connection*-type object.

Raises ValueError – If any of the property values in *record.connection.properties* are invalid.

msl.equipment.record_types module

A record from an **Equipment-Register** database or a **Connections** database.

class msl.equipment.record_types.**EquipmentRecord**(***kwargs*)

Bases: object

Contains the information about an equipment record in an **Equipment-Register** database.

Parameters ****kwargs** – The argument names can be any of the *EquipmentRecord* attribute names.

Raises

- ValueError – If an argument name is *calibration_period*, *connection* or *date_calibrated* and the value is invalid.
- AttributeError – If a named argument is not an *EquipmentRecord* attribute name.

Examples

Equipment records **should** be defined in a database and instantiated by calling msl.equipment.config.load; however, you can still manually create an equipment record.

```
>>> from msl.equipment import EquipmentRecord, ConnectionRecord,␣
↪Backend
>>> record = EquipmentRecord(
...                 manufacturer='Pico Technology',
...                 model='5244B',
...                 serial='DY135/055',
...                 connection=ConnectionRecord(
...                     backend=Backend.MSL,
...                     address='SDK::PicoScope5000A::ps5000a',
...                     properties={
...                         'resolution': '14bit',  # only used for␣
↪ps5000a series PicoScope's
...                         'auto_select_power': True,  # for PicoScopes␣
↪that can be powered by an AC adaptor or by a USB cable
...                     },
...                 )
...             )
```

alias
str – An alias to use to associate with this equipment.

asset_number
str – The IRL/CI asset number of the equipment.

calibration_period
int – The number of years that can pass before the equipment must be recalibrated.

category
str – The category (e.g., Laser, DMM) that the equipment belongs to.

connection
ConnectionRecord – The information necessary to establish a connection to the equipment.

date_calibrated
datetime.date – The date that the equipment was last calibrated.

description
str – A description of the equipment.

location
str – The location where the equipment can usually be found.

manufacturer
str – The name of the manufacturer of the equipment.

model
str – The model number of the equipment.

register
str – The value assigned, as in MSL Policy and Procedures, for any equipment that requires calibration or maintenance for projects.

section
str – The MSL section (e.g., P&R) that the equipment belongs to.

serial
str – The serial number, or engraved unique ID, of the equipment.

static **attributes**()
list of str: A list of all the attribute names for an *EquipmentRecord* object.

connect(*demo=None*)
Establish a connection to the equipment.

Parameters **demo** (bool or None) – Whether to simulate a connection to the equipment by opening a connection in demo mode. This allows you run your code if the equipment is not physically connected to the computer.

If None then the *demo* value is read from a CONFIG variable. See msl.equipment.config.load for more details.

Returns *Connection* – A *Connection*-type object.

Raises ValueError – If any of the property values in *record.connection.properties* are invalid.

class msl.equipment.record_types.**ConnectionRecord**(***kwargs*)
Bases: object

Contains the information about a connection record in a **Connections** database.

Parameters ****kwargs** – The argument names can be any of the *ConnectionRecord* attribute names.

Raises

- ValueError – If an argument name is *backend*, *interface* or *properties* and the value is invalid.
- AttributeError – If a named argument is not an *ConnectionRecord* attribute name.

Examples

Connection records **should** be defined in a database and instantiated by calling `msl.equipment.config.load`; however, you can still manually create a connection record.

```
>>> from msl.equipment import ConnectionRecord, Backend
>>> record = ConnectionRecord(
...                 manufacturer='Pico Technology',
...                 model='5244B',
...                 serial='DY135/055',
...                 backend=Backend.MSL,
...                 address='SDK::PicoScope5000A::ps5000a',
...                 properties={
...                     'resolution': '14bit',  # only used for ps5000a␣
↪series PicoScope's
...                     'auto_select_power': True,  # for PicoScopes␣
↪that can be powered by an AC adaptor or by a USB cable
...                 }
...             )
```

address
: str – The address to use for the connection (see here for examples from National Instruments).

backend
: *Backend* – The backend to use to communicate with the equipment.

interface
: *MSLInterface* – The interface to use for the communication system that transfers data between a computer and the equipment (only used if the *backend* is *Backend.MSL*).

manufacturer
: str – The name of the manufacturer of the equipment.

model
: str – The model number of the equipment.

properties
: dict – Additional properties that may be used to establish a connection to the equipment, e.g., for a Serial connection `{'baud_rate': 11920, 'data_bits': 8}`.

serial
: str – The serial number, or engraved unique ID, of the equipment.

static attributes()

list of str: A list of all the attribute names for a *ConnectionRecord* object.

CHAPTER 4

MSL-Equipment Database Formats

Databases are used by **MSL-Equipment** to store `EquipmentRecord`'s in an *Equipment-Register Database* and `ConnectionRecord`'s in a *Connections Database*. The database file formats that are currently supported are **.txt**, **.csv** and **.xls(x)**. A database is made up of **fields** (columns) and **records** (rows).

Equipment-Register Database

To adhere to international ISO/IEC 17025 standards, the information about the equipment that is used for a calibration or for a comparison must be known and it must be up to date. Keeping a central database of the equipment that is available in the lab allows for managing this information and for helping to ensure that the equipment that is being used for a calibration/comparison meets the requirements needed to achieve the final measurement uncertainty.

MSL-Equipment does not require that a *single* database is used for **ALL** equipment from **ALL** MSL sections. However, it is vital that each equipment record can only be uniquely found in **ONE Equipment-Register** database. These databases are never to be copied from one MSL section to another *(although keeping backups are required)*. Rather if you are borrowing equipment from another team you simply specify the path to that teams **Equipment-Register** database in your configuration file. See `msl.equipment.config.Config` for more details on how to specify multiple **Equipment-Register** databases in a configuration file. The owner of the equipment is responsible for ensuring that the information about the equipment is updated in their **Equipment-Register** database. The user of the equipment has access to this information through an `EquipmentRecord` object.

Each **record** in an **Equipment-Register** database is converted into an `EquipmentRecord` object when a configuration file is loaded. Each **field** is an property name for an `EquipmentRecord`.

Example **Equipment-Register** database:

The Register number used for Policies and Procedures	A column header can be any text. The name of an *EquipmentRecord* property must appear somewhere in the header text to associate the values in this column with the *EquipmentRecord* property value. This column is used to specify the Manufacturer of the equip-	Model #

Connections Database

Describe

CHAPTER 5

MSL Resources

MSL resources that are available for establishing a connection to equipment.

SDK Wrapper Classes

The following classes, which are a wrapper around the Software Development Kit (SDK) provided by the manufacturer, are available to use when defining the **Address** field in a **Connections** database.

The format of an **Address** is `SDK::PythonClassName::PathToLibrary` where `PathToLibrary` is the full path to the SDK file (or only the filename if the library path is available on `os.environ['PATH']`, see *Config* for more details) and `PythonClassName` can be one of the following classes.

For example, the **Address** field for equipment from **Bentham Instruments Ltd** can be `SDK::Bentham::benhw32_cdecl.dll`.

- Bentham Instruments Ltd
  - *Bentham* - The benhw32 library
- Pico Technology
  - PicoScope
    - *PicoScope2000* - ps2000
    - *PicoScope2000A* - ps2000a
    - *PicoScope3000* - ps3000
    - *PicoScope3000A* - ps3000a
    - *PicoScope4000* - ps4000
    - *PicoScope4000A* - ps4000a
    - *PicoScope5000* - ps5000

        * *PicoScope5000A* - ps5000a
        * *PicoScope6000* - ps6000
- Thorlabs
    - Kinesis - Wrapper package around `Thorlabs.MotionControl.C_API`.
        * *FilterFlipper* - MFF101, MFF102
        * *IntegratedStepperMotors* - LTS150, LTS300, MLJ050, K10CR1
        * *KCubeSolenoid* - KSC101
    - *FilterWheel102C* - FW102C, FW212C

CHAPTER 6

License

```
MIT License

Copyright (c) 2017, Joseph Borbely

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

CHAPTER 7

Developers

- Joseph Borbely <joseph.borbely@measurement.govt.nz>

CHAPTER 8

Release Notes

Version 0.1.0

- Initial release

CHAPTER 9

Index

- modindex

Python Module Index

m

Index

A

B

C

attribute), 20
CHANNEL_24 (msl.equipment.resources.picotech.picoscope.enums.PS3000ADigitalChannel attribute), 29
CHANNEL_25 (msl.equipment.resources.picotech.picoscope.enums.PS2000ADigitalChannel attribute), 20
CHANNEL_25 (msl.equipment.resources.picotech.picoscope.enums.PS3000ADigitalChannel attribute), 30
CHANNEL_26 (msl.equipment.resources.picotech.picoscope.enums.PS2000ADigitalChannel attribute), 20
CHANNEL_26 (msl.equipment.resources.picotech.picoscope.enums.PS3000ADigitalChannel attribute), 30
CHANNEL_27 (msl.equipment.resources.picotech.picoscope.enums.PS2000ADigitalChannel attribute), 20
CHANNEL_27 (msl.equipment.resources.picotech.picoscope.enums.PS3000ADigitalChannel attribute), 30
CHANNEL_28 (msl.equipment.resources.picotech.picoscope.enums.PS2000ADigitalChannel attribute), 20
CHANNEL_28 (msl.equipment.resources.picotech.picoscope.enums.PS3000ADigitalChannel attribute), 30
CHANNEL_29 (msl.equipment.resources.picotech.picoscope.enums.PS2000ADigitalChannel attribute), 20
CHANNEL_29 (msl.equipment.resources.picotech.picoscope.enums.PS3000ADigitalChannel attribute), 30
CHANNEL_3 (msl.equipment.resources.picotech.picoscope.enums.PS2000ADigitalChannel attribute), 19
CHANNEL_3 (msl.equipment.resources.picotech.picoscope.enums.PS3000ADigitalChannel attribute), 29
CHANNEL_30 (msl.equipment.resources.picotech.picoscope.enums.PS2000ADigitalChannel attribute), 20
CHANNEL_30 (msl.equipment.resources.picotech.picoscope.enums.PS3000ADigitalChannel attribute), 30
CHANNEL_31 (msl.equipment.resources.picotech.picoscope.enums.PS2000ADigitalChannel attribute), 20
CHANNEL_31 (msl.equipment.resources.picotech.picoscope.enums.PS3000ADigitalChannel attribute), 30
CHANNEL_4 (msl.equipment.resources.picotech.picoscope.enums.PS2000ADigitalChannel attribute), 19
CHANNEL_4 (msl.equipment.resources.picotech.picoscope.enums.PS3000ADigitalChannel attribute), 29
CHANNEL_5 (msl.equipment.resources.picotech.picoscope.enums.PS2000ADigitalChannel attribute), 19
CHANNEL_5 (msl.equipment.resources.picotech.picoscope.enums.PS3000ADigitalChannel attribute), 29
CHANNEL_6 (msl.equipment.resources.picotech.picoscope.enums.PS2000ADigitalChannel attribute), 19
CHANNEL_6 (msl.equipment.resources.picotech.picoscope.enums.PS3000ADigitalChannel attribute), 29
CHANNEL_7 (msl.equipment.resources.picotech.picoscope.enums.PS2000ADigitalChannel attribute), 19
CHANNEL_7 (msl.equipment.resources.picotech.picoscope.enums.PS3000ADigitalChannel attribute), 29
CHANNEL_8 (msl.equipment.resources.picotech.picoscope.enums attribute), 19
CHANNEL_8 (msl.equipment.resources.picotech.picoscope.enums attribute), 29
CHANNEL_9 (msl.equipment.resources.picotech.picoscope.enums attribute), 19
CHANNEL_9 (msl.equipment.resources.picotech.picoscope.enums attribute), 29
CHANNEL_ANALOGUE_OFFSET (msl.equipment.resources.picotech.picoscope.enums.PS40 attribute), 47
CHANNEL_BANDWIDTH (msl.equipment.resources.picotech.picoscope.enums.PS40 attribute), 47
CHANNEL_COUPLING (msl.equipment.resources.picotech.picoscope.enums.PS40 attribute), 47
CHANNEL_ENABLED (msl.equipment.resources.picotech.picoscope.enums.PS40 attribute), 47
CHANNEL_MAX (msl.equipment.resources.picotech.picoscope.enums.PS20 attribute), 20
CHANNEL_MAX (msl.equipment.resources.picotech.picoscope.enums.PS30 attribute), 30
CHANNEL_NAME (msl.equipment.resources.picotech.picoscope.enums.PS40 attribute), 47
CHANNEL_RANGE (msl.equipment.resources.picotech.picoscope.enums.PS40 attribute), 47
channelA (msl.equipment.resources.picotech.picoscope.structs.PS2 attribute), 96
channelA (msl.equipment.resources.picotech.picoscope.structs.PS2 attribute), 95
channelA (msl.equipment.resources.picotech.picoscope.structs.PS2 attribute), 95
channelA (msl.equipment.resources.picotech.picoscope.structs.PS2 attribute), 95
channelA (msl.equipment.resources.picotech.picoscope.structs.PS3 attribute), 99
channelA (msl.equipment.resources.picotech.picoscope.structs.PS3 attribute), 99
channelA (msl.equipment.resources.picotech.picoscope.structs.PS3 attribute), 98
channelA (msl.equipment.resources.picotech.picoscope.structs.PS3 attribute), 98
channelA (msl.equipment.resources.picotech.picoscope.structs.PS3 attribute), 97
channelA (msl.equipment.resources.picotech.picoscope.structs.PS3

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T