
MSL-Equipment Documentation

Release 0.2.0.dev0

Measurement Standards Laboratory of New Zealand

Dec 08, 2023

CONTENTS

1	Contents	3
2	Index	611
	Python Module Index	613
	Index	615

The purpose of MSL-Equipment is to manage information about equipment that are required to perform a measurement and to connect to equipment that support computer control. Three items are used to achieve this purpose

1. *Configuration File*
2. *Equipment-Register Database*
3. *Connections Database*

The following example uses a *configuration file* that specifies an *equipment-register database* (which contains information about all of the equipment that is available), a *connections database* (which contains information on how to connect to equipment that support computer control), and specifies a digital multimeter to use for a measurement (which is defined as an `<equipment>` XML element).

Loading the *configuration file* using the `Config` class is the main entry point

```
>>> from msl.equipment import Config
>>> cfg = Config('config.xml')
```

and loading the *Databases* is done via the `database()` method

```
>>> db = cfg.database()
```

You can access XML elements, attributes and values in the *configuration file*

```
>>> cfg.find('max_voltage')
<Element 'max_voltage' at ...>
>>> cfg.attrib('max_voltage')
{'unit': 'V'}
>>> cfg.value('max_voltage')
3.3
```

and all of the *EquipmentRecord*'s that are contained within the *Equipment-Register Database*¹

```
>>> for record in db.records():
...     print(record)
...
EquipmentRecord<Fluke|8506A|cecf2e0a>
EquipmentRecord<Oriel|66087|169b71e9>
EquipmentRecord<Kepco|JQE|baee83d4>
EquipmentRecord<Hewlett Packard|34401A|15ab8c6c>
EquipmentRecord<Arlunya|Milli Gauss|890a4c02>
EquipmentRecord<Toledo|1000|444213b7>
EquipmentRecord<Stanford Research Systems|SR850 DSP|cec817f5>
EquipmentRecord<HP|3478A|bd92c887>
```

To select the records that are from Hewlett Packard¹, specify the *manufacturer* as a search criteria (supports `regex`, so both *HP* and *Hewlett Packard* will match)

¹ Companies that sell equipment that is used for scientific research are identified in this guide in order to illustrate how to adequately use MSL-Equipment in your own application. Such identification is not intended to imply recommendation or endorsement by the Measurement Standards Laboratory of New Zealand, nor is it intended to imply that the companies identified are necessarily the best for the purpose.

```
>>> for record in db.records(manufacturer='H.*P'):
...     print(record)
...
EquipmentRecord<Hewlett Packard|34401A|15ab8c6c>
EquipmentRecord<HP|3478A|bd92c887>
```

Get the *ConnectionRecord*'s of the equipment that can be computer controlled

```
>>> for conn in db.connections():
...     print(conn)
...
ConnectionRecord<Fluke|8506A|cecf2e0a>
ConnectionRecord<Hewlett Packard|34401A|15ab8c6c>
ConnectionRecord<Stanford Research Systems|SR850 DSP|cec817f5>
ConnectionRecord<HP|3478A|bd92c887>
```

or filter by the equipment that use GPIB as the communication bus and that are from Hewlett Packard¹

```
>>> for conn in db.connections(address='GPIB', manufacturer='H.*P'):
...     print(conn)
...
ConnectionRecord<HP|3478A|bd92c887>
```

Access the *EquipmentRecord* that has the specified alias `dmm` in the configuration file

```
>>> record = db.equipment['dmm']
>>> print(record)
EquipmentRecord<Hewlett Packard|34401A|15ab8c6c>
>>> record.is_calibration_due()
False
>>> print(record.connection)
ConnectionRecord<Hewlett Packard|34401A|15ab8c6c>
```

Establishing a connection to the equipment is achieved by calling the `connect()` method of an *EquipmentRecord*. This will return a specific *Connection* subclass that contains the necessary properties and methods for communicating with the equipment.

```
>>> dmm = record.connect()
>>> dmm.query('*IDN?')
'Hewlett Packard,34401A,15ab8c6c,A.02.14-02.40-02.14-00.49-03-01\n'
```

CHAPTER
ONE

CONTENTS

1.1 Configuration File

A configuration file is used by MSL-Equipment to:

1. Specify which *Databases* to use
2. Specify the equipment that is being used to perform a measurement
3. Specify additional parameters to use in your program

The configuration file uses the eXtensible Markup Language ([XML](#)) format to specify this information.

The following illustrates an example configuration file.

```
<?xml version="1.0" encoding="utf-8"?>
<msl>

    <!-- OPTIONAL: Whether to open all connections in demo mode. -->
    <demo_mode>true</demo_mode>

    <!-- OPTIONAL: Set the path to the GPIB library file.
        Specifying this element is necessary only if the file is not automatically
        found or if you want to use a different file than the default file. -->
    <gpib_library>/opt/gpib/libgpib.so.0</gpib_library>

    <!-- OPTIONAL: Add paths to where external resource files are located. The
        paths get appended to Config.PATH and os.environ['PATH']. If a recursive=
        "true"
        attribute is included, then recursively adds all sub-directories starting
        from
        the root directory (also includes the root directory). The <path> element
        may
        be specified multiple times. -->
    <path>D:\code\SDKs</path>
    <path recursive="true">C:\Program Files\Manufacturer</path>

    <!-- OPTIONAL: Set the PyVISA backend library.
        @ivi (PyVISA >=1.11)
        @ni (PyVISA <1.11)
        @py (PyVISA-py)
        @sim (PyVISA-sim) -->
```

(continues on next page)

(continued from previous page)

```

<pyvisa_library>@py</pyvisa_library>

    <!-- OPTIONAL: Specify the equipment that is being used to perform the
    measurement
        and assign an "alias" that you want to use to associate for each
        equipment. You
            only need to specify enough XML attributes to uniquely identify the
            equipment record
                in an Equipment-Registry database. For example, if there is only 1
                equipment record
                    in the Equipment-Registry databases (see the <registers> tag below) that
                    is from
                        "Company XYZ" then specifying manufacturer="Company XYZ" is enough
                        information to
                            uniquely identify the equipment record. If in the future another
                            equipment record
                                is added to an Equipment-Registry database for "Company XYZ" then an
                                exception will
                                    be raised telling you to specify more information in the configuration
                                    file to
                                        uniquely identify a single equipment record. -->
<equipment alias="dmm" manufacturer="Keysight" model="34465A"/>
<equipment alias="scope" manufacturer="Pico Technologies"/>
<equipment alias="flipper" manufacturer="Thorlabs" model="MFF101/M" serial=
"123456"/>

    <!-- REQUIRED: Specify the Equipment-Register Databases to load equipment
    records from. -->
<registers>
    <!-- The "team" attribute is used to specify which research team the
        Equipment-Register database belongs to. -->
    <register team="P&R">
        <path>Z:\Equipment\Equipment Register.xls</path>
            <!-- If there are multiple Sheets in the Excel database then you must
            specify the
                name of the Sheet that contains the equipment records. This Excel
            database
                also contains connection records (see the <connections> tag below)
            and so
                the <sheet> tag must be specified. -->
        <sheet>Equipment</sheet>
    </register>
    <register team="Electrical">
        <path>H:\Quality\Registers\Equipment.xlsx</path>
            <!-- No need to specify the Sheet name if there is only 1 Sheet in the
            Excel database. -->
    </register>
    <register team="Pressure">
        <!-- Using the XML or JSON format to store equipment records allows

```

(continues on next page)

(continued from previous page)

```

→for including
    MaintenanceRecords, CalibrationRecords, MeasurandRecords, ... -->
    <path>J:\Registers\Equipment.xml</path>
</register>
<!-- For a text-based database (e.g., CSV, TXT files) you can specify
→how the dates are
    formatted and the encoding that is used in the file (UTF-8 is assumed
→if the encoding
    is not specified). A CSV database uses "," as the delimiter and a TXT
→database uses
    "\t" as the delimiter. --&gt;
&lt;register team="Time" date_format="%d.%m.%y" encoding="cp1252"&gt;
    &lt;path&gt;W:\Registers\Equip.csv&lt;/path&gt;
&lt;/register&gt;
&lt;register team="Mass" date_format="%Y-%m-%d"&gt;
    <!-- You can also specify the database path to be a path that is
→relative to the
        location of the configuration file. For example, this "equip-reg.txt"
→file is
        located in the same directory as the configuration file. --&gt;
    &lt;path&gt;equip-reg.txt&lt;/path&gt;
&lt;/register&gt;
&lt;register team="Length" user_defined="apples, pears, oranges"&gt;
    <!-- An EquipmentRecord has standard properties (e.g, manufacturer,
→model, ...) that
        are read from the database. You can also include additional fields
→from the database
        that are not part of the standard properties. Include a "user_defined
→" list
        (comma-separated) of additional properties to include. The field
→names that
        contain the text "apples", "pears" and "oranges" are added to the
→"user_defined"
        dictionary for all EquipmentRecord's in this register. --&gt;
    &lt;path&gt;I:\LS-Equip-Reg\reg.csv&lt;/path&gt;
&lt;/register&gt;
&lt;/registers&gt;

<!-- OPTIONAL: Specify the Connection Databases to load connection records
→from. --&gt;
&lt;connections&gt;
    &lt;connection&gt;
        &lt;path&gt;Z:\Equipment\Equipment Register.xls&lt;/path&gt;
        <!-- Must also specify which Sheet in this Excel database contains the
→connection records.
            This "Equipment Register.xls" file also contains an "Equipment" Sheet,
→see the
            &lt;register team="P&amp;R"&gt; element above. --&gt;
        &lt;sheet&gt;Connections&lt;/sheet&gt;
</pre>

```

(continues on next page)

(continued from previous page)

```

</connection>
<!-- You can set the encoding that is used for a text-based database. --&gt;
&lt;connection encoding="utf-16"&gt;
    <!-- Specify a relative path (relative to the location of the
        configuration file). --&gt;
    &lt;path&gt;data/my_connections.txt&lt;/path&gt;
&lt;/connection&gt;
&lt;/connections&gt;

<!-- OPTIONAL: You may define your own elements. --&gt;
&lt;max_temperature units="C"&gt;60&lt;/max_temperature&gt;

&lt;/msl&gt;
</pre>

```

The `Config` class is used to load a configuration file and it is the main entry point, for example

```

>>> from msl.equipment import Config
>>> cfg = Config('config.xml')

```

1.2 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 `xml`, `json`, `txt` (\t delimited), `csv` (, delimited) and `xls[x]`.

The `txt`, `csv` and `xls[x]` formats are simple databases that are composed of *fields* (columns) and *records* (rows). The `xml` and `json` formats allow for storing more complex data structures, such as `MaintenanceRecord`'s `CalibrationRecord`'s and `MeasurandRecord`'s for each `EquipmentRecord`. For example `xml` and `json` formats, see `equipment_register.xml` and `equipment_register.json` respectively.

1.2.1 Equipment-Register Database

Attention: The design of the Equipment-Register database is in active development and it will be unstable until an official release of MSL-Equipment is made.

The information about the equipment that is used to perform a measurement must be known and it must be kept up to date. Keeping a central and official (hence the word *Register*) database of the equipment that is available in the laboratory allows for easily managing this information and for helping to ensure that the equipment that is being used for a measurement meets the calibration requirements needed to obtain the desired measurement uncertainty.

MSL-Equipment does not require that a *single* database is used for all equipment records. However, it is vital that each equipment record can only be uniquely found in one `Equipment-Register Database`. The records in a database must never be copied from one database to another database (*keeping a backup copy of the database is encouraged*). Rather, if you are borrowing equipment from another team you simply specify the path to that teams `Equipment-Register Database` as a `<register>` element in your `Configuration File`. The owner of the equipment is responsible for ensuring that the information about

the equipment is kept up to date in their *Equipment-Register Database* and the user of the equipment defines an <equipment> element in the *Configuration File* to access this information. Therefore, an *Equipment-Register Database* is to be considered as a *global* database that can be accessed (with read permission only) by anyone.

Each record in an *Equipment-Register Database* is converted into an *EquipmentRecord*.

The following is an example of an *Equipment-Register Database* (additional *fields* can also be added to a database, see *Field Names*).

Manufacturer	Model Number	Serial Number	Description
Keysight	34465A	MY5450	6.5 digit digital multimeter
Hewlett Packard	HP8478B	BCD024	Dual element thermistor power sensors
Agilent	53230A	49e39f	Universal counter/timer

Tip: Not all records in the *Equipment-Register Database* need to have the ability to be interfaced with a computer. For example, cables, amplifiers, filters and adaptors can all be important equipment that may be used to perform a measurement and should be included in the *Equipment-Register Database* and specified as <equipment> elements in the *Configuration File*.

Field Names

Some of the supported *fields* for an *Equipment-Register Database* are:

- **Category** – The category (e.g., Laser, DMM) that the equipment belongs to.
- **Description** – A description of the equipment.
- **Location** – The location where the equipment can usually be found.
- **Manufacturer** – The name of the manufacturer of the equipment.
- **Model** – The model number of the equipment.
- **Serial** – The serial number, or engraved unique ID, of the equipment.

The text in the header of each *field* is not too particular for what it must be. The header text is parsed for one of the specific *field* names listed above and if the header contains one of these *field* names then that column is assigned to be that *field*.

For example, the following headers are valid (the blue text is what is important in the header)

- Headers can contain many words. For a *field* to be assigned to the *manufacturer* attribute the header can be written as:

This column is used to specify the Manufacturer of the equipment

- Text is case insensitive. For a *field* to be assigned to the *model* attribute the header can be written as any of the following:

MODEL No. Model # The model number of the equipment MoDeL

Although using the following header will not raise an exception, you should not use the following header because either the `manufacturer` or the `model` attribute will be assigned for this `field` depending on the order in which the `fields` appear in the database:

The model number given by the manufacturer

- Whitespace is replaced by an underscore. For a `field` to be assigned to the `is_operable` attribute the header can be written as:

Is Operable, True or False

- If the header does not contain any of the specific `field` names that are being searched for then the values in that column are silently ignored.

Equipment records should be defined in an *Equipment-Register Database* and accessed via the `database()` method; however, you can also define equipment records in a Python module, for example:

```
from msl.equipment import EquipmentRecord, ConnectionRecord, Backend

equipment = {
    'dmm':
        EquipmentRecord(
            manufacturer='HP',
            model='34401A',
            serial='123456789',
            connection=ConnectionRecord(
                backend=Backend.MSL,
                address='COM3',
                properties=dict(
                    baud_rate=19200,
                )
            )
        ),
    'scope':
        EquipmentRecord(
            manufacturer='Pico Technology',
            model='5244B',
            serial='XY135/001',
            description='Oscilloscope -- 2 Channel, 200 MHz, 1 GSPS, 512 Mpts,
↪ 5.8 ns',
            connection=ConnectionRecord(
                backend=Backend.MSL,
                address='SDK::ps5000a.dll',
                properties=dict(
                    resolution='16bit',
                )
            )
        ),
    '1ohm':
        EquipmentRecord(
```

(continues on next page)

(continued from previous page)

```

    manufacturer='Tinsley',
    model='64750',
    serial='5672413',
    description='1.0 Ohm Resistor 3A',
),
}

```

1.2.2 Connections Database

A *Connections Database* is used to store the information that is required to establish communication with the equipment.

You specify the *Connections Database* that you want to use as a `<connection>` element in your *Configuration File*. Each record in an *Connections Database* is converted into a *ConnectionRecord*.

Field Names

The supported *fields* for a *Connections Database* are:

- **Address** – The address to use for the connection (see *Address Syntax*).
- **Backend** – The *Backend* to use to communicate with the equipment.
- **Manufacturer** – The name of the manufacturer of the equipment.
- **Model** – The model number of the equipment.
- **Properties** – Additional properties that may be required to establish a connection to the equipment as key-value pairs separated by a semi-colon. For example, for a *ConnectionSerial* connection the baud rate and parity might need to be defined – `baud_rate=11920; parity=even`. The value (as in a *key-value* pair) gets cast to the appropriate data type (e.g., `int`, `float`, `str`) so the baud rate value would be `11920` as an `int` and the parity value becomes `Parity.EVEN`.
- **Serial** – The serial number, or unique ID, of the equipment.

A record in a *Connections Database* gets matched with the appropriate record in an *Equipment-Register Database* by the unique combination of the `manufacturer + model + serial` values, which when combined act as the primary key in each database.

The following is an example of a *Connections Database*. The header of each *field* also follows the same *Field Names* format used in an *Equipment-Register Database* and so `MODEL#` would also be an acceptable header for *Model Number*.

Manufacturer	Model Number	Serial Number	Backend	Address	Properties
OMEGA	iTHX-W3	458615	MSL	TCP::192.168.1.100:	termination="\r"; timeout=10
Hewlett Packard	3468A	BCD024	PyVISA	GPIB::7	
Agilent	53230A	49e39f	MSL	COM2	baud_rate=119200; parity=even

Unlike an *Equipment-Register Database* each person can maintain their own *Connections Database*. The reason being that since equipment can be shared between people some Connection *fields*, like the COM address, can vary depending on which computer the equipment is connected to and what other equipment is also connected to that computer. Therefore, everyone could have their own *Connections Database* and connection records can be copied from one *Connections Database* to another. If you are using someone else's equipment and if none of the Connection *fields* need to be changed to be able to communicate with the equipment then it is recommended to add their *Connections Database* as a <connection> element in your *Configuration File*.

Address Syntax

The following are examples of an **Address** syntax (see more examples from [National Instruments](#)).

<i>Interface</i>	Syntax Example	Description
GPIB	GPIB::10	GPIB device at board=0 (default), primary address=10, no secondary address
GPIB	GPIB0::voltmeter	GPIB device at board=0, interface name="voltmeter" (see gpib.conf)
GPIB	GPIB1::6::97::INSTR	GPIB device at board=1, primary address=6, secondary address=97
GPIB	GPIB2::INTFC	GPIB interface at board=2
PRO-LOGIX	Pro-logix::192.168.1.110::1	The GPIB-ETHERNET Controller: host=192.168.1.110, port=1234, primary-GPIB-address=6
PRO-LOGIX	Pro-logix::192.168.1.70::12	The GPIB-ETHERNET Controller: host=192.168.1.70, port=1234, primary-GPIB-address=6, secondary-GPIB-address=112
PRO-LOGIX	Pro-logix::192.168.1.70::12	The GPIB-ETHERNET Controller: host=192.168.1.70, port=1234, primary-GPIB-address=6, secondary-GPIB-address=112
PRO-LOGIX	Prologix::COM3::6	The GPIB-USB Controller: port=COM3, primary-GPIB-address=6
PRO-LOGIX	Pro-logix::/dev/ttyS0::4::96	The GPIB-USB Controller: port=/dev/ttyS0, primary-GPIB-address=4, secondary-GPIB-address=96
SDK	SDK::C:/Program Files/Manufacturer/file	Specify the full path to the SDK
SDK	SDK::filename.dll	Specify only the filename if the path to where the SDK file is located has been added as a <path> element in the Configuration File
SE-RIAL	COM2	A serial port on Windows
SE-RIAL	ASRL/dev/ttyS1	A serial port on Linux
SE-RIAL	ASRL2::INSTR	Compatible with National Instruments syntax
SE-RIAL	ASRLCOM2	Compatible with PyVISA-py syntax
SOCKET	TCP::192.168.1.100::5	Creates the connection as a <code>socket.SOCK_STREAM</code> to the IP address 192.168.1.100 at port 5000
SOCKET	UDP::192.168.1.100::5	Creates the connection as a <code>socket.SOCK_DGRAM</code>
SOCKET	TCPIP::192.168.1.100:	Compatible with National Instruments syntax
SOCKET	SOCKET::192.168.1.100	Generic socket type. You can specify the connection type in the Properties field (i.e., type=RAW)
TCPIP	TCPIP::dev.company.com	A HiSLIP LAN instrument at the hostname dev.company.com .
HiS-LIP		
TCPIP	TCPIP::10.12.114.50::1	A HiSLIP LAN instrument whose IP address is 10.12.114.50 with the server listening at port 5000
TCPIP	TCPIP::dev.company.com	A VXI-11.3 LAN instrument at the hostname dev.company.com . This uses the default LAN Device Name inst0
VXI-11		
TCPIP	TCPIP::10.6.56.21::gpi	A VXI-11.2 GPIB device whose IP address is 10.6.56.21
VXI-11		
TCPIP	TCPIP::192.168.1.100	A VXI-11.3 LAN instrument at IP address 192.168.1.100 . Note that default values for board 0 and LAN device name inst0 will be used
ZMQ	ZMQ::192.168.20.90::1	Use the ZeroMQ messaging library to connect to a device at IP 192.168.20.90 on port 1 (5555)

1.3 Install MSL-Equipment

To install MSL-Equipment run

```
pip install https://github.com/MSLNZ/msl-equipment/releases/download/v0.1.0/  
→msl_equipment-0.1.0-py2.py3-none-any.whl
```

Alternatively, using the [MSL Package Manager](#) run

```
msl install equipment
```

1.3.1 Dependencies

- Python 3.8+
- msl-loadlib
- msl-io
- numpy
- pyserial
- pyzmq

1.3.2 Optional Dependencies

- PyVISA
- PyVISA-py
- NI-DAQmx

Some of the [MSL Resources](#) might not work in your application because the resource might depend on an external dependency (e.g., the SDK provided by a manufacturer) and this external dependency might not be available for your operating system.

1.4 Examples

Some example scripts that illustrate how to use MSL-Equipment to communicate with equipment can be found in the [repository](#).

1.5 MSL Resources

MSL Resources are specific classes that are used to communicate with the equipment.

- Aim and Thurlby Thandar Instruments
 - [MXSeries](#) – MX100QP, MX100TP, MX180TP
- Avantes
 - [Avantes](#) – Wrapper around the [AvaSpec SDK](#)

- [Bentham Instruments Ltd](#)
 - [*Bentham*](#) – Wrapper around the [Bentham SDK](#)
- [CMI](#) (Czech Metrology Institute)
 - [*SIA3*](#) – Switched Integrator Amplifier
- [DataRay](#)
 - [*DataRayOCX64*](#) – Wrapper around the [DATARAYOCX](#) library
- [Electron Dynamics Ltd](#)
 - [*TCSeries*](#) – Temperature Controller (TC LV, TC M, TC Lite)
- [Energetiq](#)
 - [*EQ99*](#) – EQ-99 Manager
- [Greisinger](#)
 - [*GMH3000*](#) – GMH 3000 Series thermometer
- [MKS Instruments](#)
 - [*PR4000B*](#) – Flow and Pressure controller
- [NKT Photonics](#)
 - [*NKT*](#) – Wrapper around the [NKT Photonics SDK](#)
- [OMEGA](#)
 - [*iTHX*](#) – iTHX-W3, iTHX-D3, iTHX-SD, iTHX-M, iTHX-W, iTHX-2
- [OptoSigma](#)
 - [*SHOT702*](#) – Two-axis Stage Controller
- [Optronic Laboratories](#)
 - [*OL756*](#) – UV-VIS spectroradiometer
 - [*OLCurrentSource*](#) – DC Current Source (16A, 65A, 83A)
- [Pico Technology](#) – Wrapper around the [Pico Technology SDK](#)
 - [*PicoScope*](#)
 - * [*PicoScope2000*](#) - PicoScope 2000 Series
 - * [*PicoScope2000A*](#) - PicoScope 2000 Series A
 - * [*PicoScope3000*](#) - PicoScope 3000 Series
 - * [*PicoScope3000A*](#) - PicoScope 3000 Series A
 - * [*PicoScope4000*](#) - PicoScope 4000 Series
 - * [*PicoScope4000A*](#) - PicoScope 4000 Series A
 - * [*PicoScope5000*](#) - PicoScope 5000 Series
 - * [*PicoScope5000A*](#) - PicoScope 5000 Series A
 - * [*PicoScope6000*](#) - PicoScope 6000 Series

- PT-104 Platinum Resistance Data Logger
 - * *PT104* – PT-104
- Princeton Instruments
 - *PrincetonInstruments* – Wrapper around the `ARC_Instrument.dll` library
- Raicol Crystals
 - *RaicolTEC* – TEC (Peltier-based) oven
- Thorlabs
 - Wrapper around the `Kinesis` SDK.
 - * *FilterFlipper* – MFF101, MFF102
 - * *IntegratedStepperMotors* – LTS150, LTS300, MLJ050, MLJ150, K10CR1
 - * *KCubeSolenoid* – KSC101
 - * *KCubeStepperMotor* – KST101
 - * *KCubeDCServo* – KDC101
 - * *BenchtopStepperMotor* – BSC101, BSC102, BSC103, BSC201, BSC202, BSC203
 - *FilterWheelXX2C* – FW102C, FW212C

1.5.1 Creating a new MSL Resource

When adding a new MSL Resource class to the repository the following steps should be performed. Please follow the [style guide](#).

Note: If you do not want to upload the new MSL Resource class to the [repository](#) then you only need to write the code found in Step 5 to use your class in your own programs.

1. Create a [fork](#) of the [repository](#).
2. If you are adding a new MSL Resource for equipment from a manufacturer that does not already exist in the `msl/equipment/resources` directory then create a new Python package in `msl/equipment/resources` using the name of the *manufacturer* as the package name.
3. Create a new Python module, in the package from Step 2, using the *model number* of the equipment as the name of the module.
4. If a `msl.equipment.exceptions` class has not been created for this manufacture then create a new exception handler class using the name of the *manufacturer* in the class name.
5. Create a new connection class within the module that you created in Step 3. The class must be a subclass of one of the [Connection Classes](#).

```
# msl/equipment/resources/<manufacturer>/<model_number>.py
#
from msl.equipment.resources import register
from msl.equipment.exceptions import TheErrorClassFromStep4 # this is
    ↴optional
```

(continues on next page)

(continued from previous page)

```

from msl.equipment.connection_xxx import ConnectionXxx # replace xxx
    ↪with the Connection subclass

# Register your class so that MSL-Equipment knows that it exists
@register(manufacturer='a regex pattern', model='a regex pattern') # can
    ↪include a `flags` kwarg
class ModelNumber(ConnectionXxx): # change ModelNumber and ConnectionXxx

    def __init__(self, record):
        """Edit the docstring...

        Do not instantiate this class directly. Use the :meth:`~.
        ↪EquipmentRecord.connect`
            method to connect to the equipment.

    Parameters
    -----
    record : :class:`~.EquipmentRecord`
        A record from an :ref:`equipment-database`.
    """
    super(ModelNumber, self).__init__(record) # change ModelNumber

    # the following is optional, the default exception handler is
    ↪MSLConnectionError
    self.set_exception_class(TheErrorClassFromStep4) # change
    ↪TheErrorClassFromStep4

```

6. Add at least one example for how to use the new MSL Resource in `msl/examples/equipment`. Follow the template of the other examples in this package for naming conventions and for showing how to use the new MSL Resource.
7. Create tests for the new MSL Resource. The tests cannot be dependent on whether the equipment is physically connected to the computer running the test (ideally the examples that you write in Step 6 will demonstrate that communicating with the equipment works). The very minimal test to create is to add a test case to the `def test_find_resource_class()` function for ensuring that your class is returned for various values of `manufacturer` and `model`. Run the tests using `python setup.py test` (ideally you would run the tests for all *currently-supported versions* of Python, see also `condatests.py`).
8. Add `.rst` documentation files for the new MSL Resource to the `docs/_api` folder. You can either run `python setup.py apidoc` to automatically generate the `.rst` documentation files or you can create the necessary `.rst` files manually. Running `python setup.py apidoc` will generate `.rst` files for *all* modules in **MSL-Equipment** in the `docs/_autosummary` folder. Only copy the `.rst` files that are associated with your new MSL Resource to the `docs/_api` folder. After copying the files you can delete the `docs/_autosummary` folder before running `python setup.py docs` to build the documentation, otherwise you will get numerous warnings. If you want to manually create the `.rst` files then look in the `docs/_api` folder for examples from other MSL Resources.
9. If you created a new package in Step 2 then you need to add the new package to the `toctree` of the `Subpackages` section in `docs/_api/msl.equipment.resources.rst`. Insert the name of the new MSL Resource package in the file alphabetically. If you forget to do this step then a warning will appear when building the documentation to help remind you to do it. If you did not create a new

package in Step 2 then add the `.rst` file from Step 8 to the Subpackages section in the appropriate `msl.equipment.resources.*.rst` file.

10. Add the new MSL Resource class, alphabetically, to the list of MSL Resources in `docs/resources.rst`. Follow the template that is used for the other MSL Resources listed in this file.
11. Add yourself to `AUTHORS.rst` and add a note in `CHANGES.rst` that you created this new Resource. These files are located in the root directory of the **MSL-Equipment** package.
12. If running the tests pass and building the docs show no errors/warnings then create a [pull request](#).

1.6 API Documentation

Although this package contains many classes and functions, the only object that you must initialize in your application is `Config` and perhaps `EquipmentRecord`'s (depending on the format that is chosen to store the `Databases`).

1.6.1 Find Equipment

To find equipment that can be connected to, you may either call the `find_equipment()` function or run the `find-equipment` executable from a terminal. To see the help for the executable, run

```
find-equipment --help
```

Running either the function or the executable will return/display a description about the equipment and the address(es) that may be used to connect to the equipment.

1.6.2 Connection Classes

The following `Connection` classes are available to communicate with the equipment (*although you should never need to instantiate these classes directly*):

<code>ConnectionDemo</code>	Simulate a connection to the equipment
<code>ConnectionGPIB</code>	Equipment that use the IEEE-488 bus (GPIB)
<code>ConnectionMessageBased</code>	Equipment that use message-based communication
<code>ConnectionPrologix</code>	Equipment that is connected through a <code>Prologix</code> Controller
<code>ConnectionSDK</code>	Equipment that use the manufacturer's SDK for the connection
<code>ConnectionSerial</code>	Equipment that is connected through a Serial port
<code>ConnectionSocket</code>	Equipment that is connected through a Socket
<code>ConnectionTCPIPWXI11</code>	Equipment that use the <code>VXI-11</code> protocol
<code>ConnectionTCPiPhiSLIP</code>	Equipment that use the <code>HiSLIP</code> protocol
<code>ConnectionZeroMQ</code>	Equipment that use the <code>ZeroMQ</code> protocol

and the `Connection` classes that are available from external Python libraries are:

<code>ConnectionPyVISA</code>	Uses <code>PyVISA</code> to establish a connection to the equipment
<code>ConnectionNIDAQ</code>	Uses <code>NI-DAQ</code> to establish a connection to the equipment

1.6.3 Package Structure

msl.equipment package

Manage and connect to equipment in the laboratory.

`msl.equipment.version_info = (0, 2, 0, 'dev0')`

Contains the version information as a (major, minor, micro, releaselevel) tuple.

Type

`namedtuple`

`msl.equipment.find_equipment(*, ip: list[str] | None = None, timeout: float = 2, gpib_library: str = "", include_sad: bool = True) → ValuesView`

Returns information about equipment that are available.

Parameters

- **ip** – The IP address(es) on the local computer to use to search for network devices. If not specified, uses all network interfaces.
- **timeout** – The maximum number of seconds to wait for a reply from a network device.
- **gpib_library** – The path to a GPIB library file. The default file that is used is platform dependent. If a GPIB library cannot be found, GPIB devices will not be searched for.
- **include_sad** – Whether to scan all secondary GPIB addresses.

Returns

The information about the devices that were found.

msl.equipment.config module

Load an XML *Configuration File*.

`msl.equipment.config.XMLType`

An XML-document type that can be parsed.

alias of `Union[str, bytes, PathLike, BinaryIO, TextIO]`

`class msl.equipment.config.Config(source: str | bytes | PathLike | BinaryIO | TextIO)`

Bases: `object`

Load an XML *Configuration File*.

The purpose of the *Configuration File* is to define parameters that may be required during data acquisition and to access *EquipmentRecord*'s from an *Equipment-Register Database* and *ConnectionRecord*'s from a *Connections Database*.

The following table summarizes the XML elements that are used by MSL-Equipment which may be defined in a *Configuration File*:

XML Tag	Example Values	Description
demo_	true, false, True, False	Whether to open connections in demo mode. The value will set <code>DEMO_MODE</code> .
gpib_1	/opt/gpib/lib	The path to a GPIB library file. Required only if you want to use a specific file. The value will set <code>GPIB_LIBRARY</code> .
path	C:\Program Files\Compa	A path that contains additional resources. Accepts a <code>recursive="true"</code> attribute. The path(s) are appended to <code>PATH</code> and to <code>os.environ['PATH']</code> . A <code><path></code> element may be specified multiple times.
pyvisa	@ivi, @py, /opt/ni/libvis	The PyVISA library to use. The value will set <code>PyVISA_LIBRARY</code> .

You are also encouraged to define your own application-specific elements within your *Configuration File*.

Parameters

source – A filename or file object containing XML data.

`GPIB_LIBRARY: str = ''`

The path to a GPIB library file.

Setting this attribute is necessary only if you want to communicate with a GPIB device and the file is not automatically found or you want to use a different file than the default file.

`PyVISA_LIBRARY: str = '@ivi'`

The PyVISA backend `library` to use.

`DEMO_MODE: bool = False`

Whether to open connections in demo mode.

If enabled then the equipment does not need to be physically connected to a computer and the connection is simulated.

`PATH: list[str] = []`

Paths are also appended to `os.environ['PATH']`.

`attrib(tag_or_path: str) → dict[str, Any]`

Get the attributes of the first matching element by tag name or path.

The values are converted to the appropriate data type if possible. For example, if the text of the element is `true` it will be converted to `True`, otherwise the value will be kept as a `str`.

Parameters

tag_or_path – Either an element tag name or an XPath.

Returns

The attributes of the matching element.

`database() → Database`

A reference to the equipment and connection records in the database(s).

`find(tag_or_path: str) → Element | None`

Find the first matching element by tag name or path.

Parameters

tag_or_path – Either an element tag name or an XPath.

Returns

The element or `None` if no element was found.

findall(*tag_or_path*: `str`) → list[`Element`]

Find all matching sub-elements by tag name or path.

Parameters

tag_or_path – Either an element tag name or an XPath.

Returns

All matching elements in document order.

property path: `str`

The path to the configuration file.

property root: `Element`

The root element (the first node) in the XML document.

value(*tag_or_path*: `str`, *default*: `Any` = `None`) → `Any`

Gets the value (text) associated with the first matching element.

The value is converted to the appropriate data type if possible. For example, if the text of the element is `true` it will be converted to `True`.

Parameters

- **tag_or_path** – Either an element tag name or an XPath.
- **default** – The default value if an element cannot be found.

Returns

The value of the element or *default* if no element was found.

msl.equipment.connection module

Base class for establishing a connection to the equipment.

class `msl.equipment.connection.Connection`(*record*)

Bases: `object`

Base class for establishing a connection to the equipment.

Do not instantiate this class directly. Use the `connect()` method to connect to the equipment.

Parameters

record (`EquipmentRecord`) – A record from an *Equipment-Register Database*.

property equipment_record

The information about the equipment.

Type

`EquipmentRecord`

`disconnect()`

Disconnect from the equipment.

This method should be overridden in the 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 garbage collected, which happens when the reference count is 0.

`raise_exception(message)`

Raise an `MSLConnectionError` and log the error message.

Parameters

`message` (`str` or `Exception`) – The message to display in the exception class that was set by `set_exception_class()`. If an `Exception` object then its string representation is used as the message.

`static convert_to_enum(obj, enum, prefix=None, to_upper=False, strict=True)`

Convert `obj` to an `Enum`.

Parameters

- `obj` – Any object to be converted to the specified `enum`. Can be a value of member of the specified `enum`.
- `enum` – The `Enum` object that `obj` should be converted to.
- `prefix` (`str`, optional) – If `obj` is a `str`, then ensures that `prefix` is included at the beginning of `obj` before converting `obj` to the `enum`.
- `to_upper` (`bool`, optional) – If `obj` is a `str`, then whether to change `obj` to be upper case before converting `obj` to the `enum`.
- `strict` (`bool`, optional) – Whether errors should be raised. If `False` and `obj` cannot be converted to `enum` then `obj` is returned and the error is logged.

Returns

`Enum` – The `enum`.

Raises

`ValueError` – If `obj` is not in `enum` and `strict` is `True`.

`static log_debug(msg, *args, **kwargs)`

Log a debug message.

All input parameters are passed to `debug()`.

`static log_info(msg, *args, **kwargs)`

Log an info message.

All input parameters are passed to `info()`.

```
static log_warning(msg, *args, **kwargs)
    Log a warning message.

    All input parameters are passed to warning().

static log_error(msg, *args, **kwargs)
    Log an error message.

    All input parameters are passed to error().

static log_critical(msg, *args, **kwargs)
    Log a critical message.

    All input parameters are passed to critical().
```

set_exception_class(*handler*)

Set the exception-handler class for this `Connection`.

Parameters**handler** (`MSLConnectionError`) – A subclass of `MSLConnectionError`**Raises****TypeError** – If the *handler* is not a subclass of `MSLConnectionError`**static parse_address**(*address*)

Determine whether a subclass should be used to connect to the equipment.

Attention: The subclass should override this method.**Parameters****address** (`str`) – The address of a `ConnectionRecord`.**Returns**`dict` or `None` – If the *address* is in a valid format for the subclass to be able to connect to the equipment then a `dict` is returned containing the information necessary to connect to the equipment. Otherwise, if the *address* is not valid for the subclass to be able to connect to the equipment then `None` is returned.**msl.equipment.connection_demo module**

Simulate a connection to the equipment.

class msl.equipment.connection_demo.ConnectionDemo(*record, cls*)Bases: `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 a 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 method. For example, if `:rtype: int` then an `int` is returned or if `:rtype: int, float` then an `int` and a `float`

are 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 `record.connect(demo=True)` method to connect to the equipment in demo mode or set `DEMO_MODE` to be `True` in the *Configuration File* to open all connections in demo mode.

Parameters

- `record` (*EquipmentRecord*) – A record from an *Equipment-Register Database*.
- `cls` (*Connection*) – A *Connection* subclass (that has NOT been instantiated).

`disconnect()`

Log a disconnection from the equipment.

`msl.equipment.connection_gpib module`

Base class for equipment that is connected through GPIB.

`msl.equipment.connection_gpib.find_listeners(include_sad: bool = True) → list[str]`

Find GPIB listeners.

Parameters

`include_sad` – Whether to scan all secondary GPIB addresses.

Returns

The GPIB addresses that were found.

`class msl.equipment.connection_gpib.ConnectionGPIB(record: EquipmentRecord)`

Bases: *ConnectionMessageBased*

Base class for equipment that is connected through GPIB.

The *properties* for a GPIB connection supports the following key-value pairs in the *Connections Database*:

```
'encoding': str, the encoding to use [default: 'utf-8']
'encoding_errors': str, encoding error handling scheme, e.g. 'strict',
    ↳'ignore' [default: 'strict']
'eos_mode': int, the end-of-string mode [default: 0]
'max_read_size': int, the maximum number of bytes that can be read,
    ↳[default: 1 MB]
'read_termination': str or None, read until this termination sequence is
    ↳found [default: None]
'rstrip': bool, whether to remove trailing whitespace from "read"
    ↳messages [default: False]
'send_eoi': bool, enables or disables the assertion of the EOI signal
    ↳[default: True]
'termination': shortcut for setting both 'read_termination' and 'write_
    ↳termination' to this value
'timeout': float or None, the timeout (in seconds) for read and write
    ↳operations [default: None]
```

(continues on next page)

(continued from previous page)

```
'write_termination': str or None, termination sequence appended to write_
→messages [default: '\r\n']
```

The `backend` value must be equal to `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 `connect()` method to connect to the equipment.

Parameters

`record` – A record from an *Equipment-Register Database*.

`ask(option: int, *, handle: int | None = None) → int`

Get a configuration setting (board or device).

This method is the `ibask` function, it should not be confused with the `query()` method.

Parameters

- `option` – A configuration setting to get the value of.
- `handle` – Board or device descriptor. Default is the handle of the instantiated class.

Returns

The value of the configuration setting.

`property board: int`

Returns the board index.

`clear(*, handle: int | None = None) → int`

Send the clear command (device).

This method is the `ibclr` function.

Parameters

`handle` – Board or device descriptor. Default is the handle of the instantiated class.

Returns

The status value (ibsta).

`command(data: bytes, *, handle: int | None = None) → int`

Write command bytes (board).

This method is the `ibcmd` function.

Parameters

- `data` – The `commands` to write to the bus.
- `handle` – Board or device descriptor. Default is the handle of the instantiated class.

Returns

The status value (ibsta).

config(*option*: *int*, *value*: *int*, *, *handle*: *int* | *None* = *None*) → *int*

Change configuration settings (board or device).

This method is the [ibconfig](#) function.

Parameters

- **option** – A configuration setting to change the value of.
- **value** – The new configuration setting value.
- **handle** – Board or device descriptor. Default is the handle of the instantiated class.

Returns

The status value (ibsta).

control_atn(*state*: *int*, *, *handle*: *int* | *None* = *None*) → *int*

Set the state of the ATN line (board).

This method mimics the PyVISA-py implementation.

Parameters

- **state** – The state of the ATN line or the active controller.
Allowed values are:
 - 0: ATN_DEASSERT
 - 1: ATN_ASSERT
 - 2: ATN_DEASSERT_HANDSHAKE
 - 3: ATN_ASSERT_IMMEDIATE
- **handle** – Board or device descriptor. Default is the handle of the instantiated class.

Returns

The status value (ibsta).

control_ren(*state*: *int*, *, *handle*: *int* | *None* = *None*) → *int*

Controls the state of the GPIB Remote Enable (REN) interface line.

Optionally the remote/local state of the device is also controlled.

This method mimics the PyVISA-py implementation.

Parameters

- **state** – Specifies the state of the REN line and optionally the device remote/local state.
Allowed values are:

- 0: REN_DEASSERT
- 1: REN_ASSERT
- 2: REN_DEASSERT_GTL
- 3: REN_ASSERT_ADDRESS
- 4: REN_ASSERT_LLO

- 5: REN_ASSERT_ADDRESS_LLO
 - 6: REN_ADDRESS_GTL
- **handle** – Board or device descriptor. Default is the handle of the instantiated class.

Returns

The status value (ibsta).

count() → **int**

Get the number of bytes sent or received.

This method is the `ibcntl` function.

disconnect() → **None**

Close the GPIB connection.

property handle: int

Returns the handle of the instantiated board or device.

interface_clear(*, handle: int | None = None) → int

Perform interface clear (board).

Resets the GPIB bus by asserting the *interface clear* (IFC) bus line for a duration of at least 100 microseconds.

This method is the `ibsic` function.

Parameters

handle – Board or device descriptor. Default is the handle of the instantiated class.

Returns

The status value (ibsta).

property library_path: str

Returns the path to the GPIB library.

lines(*, handle: int | None = None) → int

Returns the status of the control and handshaking bus lines (board).

This method is the `iblines` function.

Parameters

handle – Board or device descriptor. Default is the handle of the instantiated class.

listener(pad: int, sad: int = 0, *, handle: int | None = None) → bool

Check if a listener is present (board or device).

This method is the `ibln` function.

Parameters

- **pad** – Primary address of the GPIB device.
- **sad** – Secondary address of the GPIB device.
- **handle** – Board or device descriptor. Default is the handle of the instantiated class.

Returns

Whether a listener is present.

local(**, handle: int | None = None*) → int

Go to local mode (board or device).

This method is the `ibloc` function.

Parameters

handle – Board or device descriptor. Default is the handle of the instantiated class.

Returns

The status value (ibsta).

online(*value: bool*, **, handle: int | None = None*) → int

Close or reinitialize descriptor (board or device).

This method is the `ibonl` function.

If you want to close the connection for the GPIB board or device that was instantiated, use `disconnect()`.

Parameters

- **value** – If `False`, closes the connection. If `True`, then all settings associated with the descriptor (GPIB address, end-of-string mode, timeout, etc.) are reset to their *default* values. The *default* values are the settings the descriptor had when it was first obtained.
- **handle** – Board or device descriptor. Default is the handle of the instantiated class.

Returns

The status value (ibsta).

property name: str | None

Returns the name of the board or device or `None` if a name was not specified in the `address`.

static parse_address(*address: str*) → dict | None

Get the board, interface name, primary address and secondary address.

Parameters

address – The address of a `ConnectionRecord`

Returns

The information about the GPIB connection or `None` if `address` is not valid for a GPIB interface.

pass_control(**, handle: int | None = None, name: str | None = None, board: int | None = None, pad: int = 0, sad: int = 65535*) → int

Set a GPIB board or device to become the controller-in-charge (CIC).

This method is the `ibpct` function.

If no arguments are specified, the instantiated class becomes the CIC.

Parameters

- **handle** – Board or device descriptor. If specified, *name*, *board*, *pad* and *sad* are ignored.
- **name** – The name of a GPIB board or device. If specified, *board*, *pad* and *sad* are ignored.
- **board** – Index of the GPIB interface board.
- **pad** – Primary address of the GPIB device.
- **sad** – Secondary address of the GPIB device.

Returns

The handle of the board or device that became CIC.

`property primary_address: int | None`

Returns the primary address of the GPIB device or `None` if a primary address was not specified in the *address*.

`property read_termination: bytes | None`

The termination character sequence that is used for the `read()` method.

By default, reading stops when the EOI line is asserted.

`remote_enable(value: bool, *, handle: int | None = None) → int`

Set remote enable (board).

This method is the `ibsre` function.

Parameters

- **value** – If `True`, the board asserts the REN line. Otherwise, the REN line is unasserted. The board must be the system controller.
- **handle** – Board or device descriptor. Default is the handle of the instantiated class.

Returns

The status value (`ibsta`).

`property secondary_address: int | None`

Returns the secondary address of the GPIB device.

`serial_poll(*, handle: int | None = None) → int`

Read status byte / serial poll (device).

This method is the `ibrsp` function.

Parameters

handle – Board or device descriptor. Default is the handle of the instantiated class.

Returns

The status byte.

`spoll_bytes(*, handle: int | None = None) → int`

Get the length of the serial poll bytes queue (device).

This method is the `ibspb` function.

Parameters

handle – Board or device descriptor. Default is the handle of the instantiated class.

status() → int

Returns the status value (ibsta).

trigger(*, handle: int | None = None) → int

Trigger device.

This method is the `ibtrg` function.

Parameters

handle – Board or device descriptor. Default is the handle of the instantiated class.

Returns

The status value (ibsta).

version() → str

Returns the version of the GPIB library (linux).

wait(mask: int, *, handle: int | None = None) → int

Wait for an event (board or device).

This method is the `ibwait` function.

Parameters

- **mask** – Wait until one of the conditions specified in `mask` is true.
- **handle** – Board or device descriptor. Default is the handle of the instantiated class.

Returns

The status value (ibsta).

wait_for_srq(*, handle: int | None = None) → int

Wait for the SRQ line to be asserted (board or device).

Parameters

handle – Board or device descriptor. Default is the handle of the instantiated class.

Returns

The status value (ibsta).

write_async(message: bytes, *, handle: int | None = None) → int

Write a message asynchronously (board or device).

This method is the `ibwrta` function.

Parameters

- **message** – The data to send.
- **handle** – Board or device descriptor. Default is the handle of the instantiated class.

Returns

The status value (ibsta).

msl.equipment.connection_message_based module

Base class for equipment that use message-based communication.

class `msl.equipment.connection_message_based.ConnectionMessageBased(record)`

Bases: `Connection`

Base class for equipment that use message-based communication.

The `backend` value must be equal to `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 `connect()` method to connect to the equipment.

Parameters

`record (EquipmentRecord)` – A record from an `Equipment-Register Database`.

`CR = b'\r'`

The carriage-return character (hex: 0x0D, decimal: 13).

Type

`bytes`

`LF = b'\n'`

The line-feed character (hex: 0x0A, decimal: 10).

Type

`bytes`

property encoding

The encoding that is used for `read()` and `write()` operations.

Type

`str`

property encoding_errors

The error handling scheme to use when encoding and decoding messages.

For example: `strict, ignore, replace, xmlcharrefreplace, backslashreplace`

Type

`str`

property read_termination

The termination character sequence that is used for the `read()` method.

Reading stops when the equipment stops sending data or the `read_termination` character sequence is detected. If you set the `read_termination` to be equal to a variable of type `str` it will automatically be encoded.

Type

`bytes or None`

property write_termination

The termination character sequence that is appended to `write()` messages.

If you set the `write_termination` to be equal to a variable of type `str` it will automatically be encoded.

Type
`bytes` or `None`

property `max_read_size`

The maximum number of bytes that can be `read()`.

Type
`int`

property `timeout`

The timeout, in seconds, for `read()` and `write()` operations.

A value `0` will set the timeout to be `None` (blocking mode).

Type
`float` or `None`

property `rstrip`

Whether to remove trailing whitespace from `read()` messages.

Type
`bool`

`raise_timeout(append_msg='')`

Raise a `MSLTimeoutError`.

Parameters

`append_msg` (`str`, optional) – A message to append to the generic timeout message.

`read(size=None, fmt='ascii', dtype=None, decode=True)`

Read a message from the equipment.

This method will block until one of the following conditions is fulfilled:

1. the `read_termination` byte(s) is(are) received – only if `read_termination` is not `None`.
2. `size` bytes have been received – only if `size` is not `None`.
3. a timeout occurs – only if `timeout` is not `None`. An `MSLTimeoutError` is raised.
4. `max_read_size` bytes have been received. An `MSLConnectionError` is raised.

Parameters

- `size` (`int`, optional) – The number of bytes to read. Ignored if it is `None`.
- `fmt` (`str` or `None`, optional) – The format that the message data is in. Ignored if `dtype` is not specified. See `from_bytes()` for more details.
- `dtype` – The data type of the elements in the message data. Can be any object that `numpy.dtype` supports. See `from_bytes()` for more details. For messages that are of scalar type (i.e., a single number) it is more efficient to not specify `dtype` but to pass the message to the `int` or `float` class to convert the message to the appropriate numeric type.
- `decode` (`bool`, optional) – Whether to decode the message (i.e., convert the message to a `str`) or keep the message as `bytes`. Ignored if `dtype` is specified.

Returns

`str`, `bytes` or `ndarray` – The message from the equipment. If `dtype` is specified, then the message is returned as an `ndarray`, if `decode` is `True` then the message is returned as a `str`, otherwise the message is returned as `bytes`.

See also:

`rstrip`

`write(message, data=None, fmt='ieee', dtype='<f'`)

Write a message to the equipment.

Parameters

- **message** (`str` or `bytes`) – The message to write to the equipment.
- **data** (`list`, `tuple` or `numpy.ndarray`, optional) – The data to append to `message`. See `to_bytes()` for more details.
- **fmt** (`str` or `None`, optional) – The format to use to convert `data` to bytes. Ignored if `data` is `None`. See `to_bytes()` for more details.
- **dtype** – The data type to use to convert each element in `data` to bytes. Ignored if `data` is `None`. See `to_bytes()` for more details.

Returns

`int` – The number of bytes written.

`query(message, delay=0.0, **kwargs)`

Convenience method for performing a `write()` followed by a `read()`.

Parameters

- **message** (`str` or `bytes`) – The message to write to the equipment.
- **delay** (`float`, optional) – The time delay, in seconds, to wait between `write()` and `read()` operations.
- ****kwargs** – All additional keyword arguments are passed to `read()`

Returns

`str`, `bytes` or `ndarray` – The message from the equipment. If `dtype` is specified, then the message is returned as an `ndarray`, if `decode` is `True` then the message is returned as a `str`, otherwise the message is returned as `bytes`.

msl.equipment.connection_nidaq module

Uses NI-DAQ as the backend to communicate with the equipment.

`class msl.equipment.connection_nidaq.ConnectionNIDAQ(record)`

Bases: `Connection`

Uses NI-DAQ to establish a connection to the equipment.

See the `nidaqmx` examples for how to use NI-DAQ.

The returned object from the `connect()` method is equivalent to importing the NI-DAQ package.

For example:

```
nidaqmx = record.connect()
with nidaqmx.Task() as task:
    task.ai_channels.add_ai_voltage_chan('Dev1/ai0')
    voltage = task.read()
```

is equivalent to:

```
import nidaqmx
with nidaqmx.Task() as task:
    task.ai_channels.add_ai_voltage_chan('Dev1/ai0')
    voltage = task.read()
```

The *backend* value must be equal to *NIDAQ* 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 NIDAQ.

Do not instantiate this class directly. Use the `connect()` method to connect to the equipment.

Parameters

`record (EquipmentRecord)` – A record from an *Equipment-Register Database*.

property constants

Returns the `nidaqmx.constants` module.

property CtrFreq

Returns the `CtrFreq` class.

property CtrTick

Returns the `CtrTick` class.

property CtrTime

Returns the `CtrTime` class.

property DaqError

Returns the `DaqError` class.

property DaqResourceWarning

Returns the `DaqResourceWarning` class.

property DaqWarning

Returns the `DaqWarning` class.

property errors

Returns the `nidaqmx.errors` module.

property Scale

Returns the `Scale` class.

property stream_readers

Returns the `nidaqmx.stream_readers` module.

property stream_writers

Returns the `nidaqmx.stream_writers` module.

property system

Returns the `nidaqmx.system` module.

property Task

Returns the `Task` class.

property types

Returns the `nidaqmx.types` module.

property utils

Returns the `nidaqmx.utils` module.

property version

The NI-DAQmx driver version number.

Type

`str`

msl.equipment.connection_prologix module

Uses `Prologix` hardware to establish a connection to the equipment.

class `msl.equipment.connection_prologix.ConnectionPrologix(record)`

Bases: `Connection`

Uses `Prologix` hardware to establish a connection to the equipment.

For the GPIB-ETHERNET Controller, the format of the `address` is `Prologix::HOST::1234::PAD[::SAD]`, where PAD (Primary Address) is a decimal value between 0 and 30 and SAD (Secondary Address) is a decimal value between 96 and 126. SAD is optional. For example, `Prologix::192.168.1.110::1234::6` or `Prologix::192.168.1.110::1234::6::96`.

For the GPIB-USB Controller, the format of the `address` is `Prologix::PORT::PAD[::SAD]`, where PAD (Primary Address) is a decimal value between 0 and 30 and SAD (Secondary Address) is a decimal value between 96 and 126. SAD is optional. For example, `Prologix::COM3::6` or `Prologix::/dev/ttyUSB0::6::112`.

The `properties` for a `Prologix` connection supports the following key-value pairs in the `Connections Database` and any of the key-value pairs supported by `ConnectionSerial` or `ConnectionSocket` (depending on whether a GPIB-USB or a GPIB-ETHERNET Controller is used):

```
'eoi': int, 0 or 1
'eos': int, 0, 1, 2 or 3
'eot_char': int, an ASCII value less than 256
'eot_enable': int, 0 or 1
'mode': int, 0 or 1 [default: 1]
'read_tmo_ms': int, a timeout value between 1 and 3000 milliseconds
```

The `backend` value must be equal to `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 `connect()` method to connect to the equipment.

Parameters

record (*EquipmentRecord*) – A record from an *Equipment-Register Database*.

controllers = {}

A *dict* of all *Prologix* Controllers that are being used to communicate with GPIB devices.

selected_addresses = {}

A *dict* of the currently-selected GPIB address for all *Prologix* Controllers.

property encoding

The encoding that is used for *read()* and *write()* operations.

Type

str

property encoding_errors

The error handling scheme to use when encoding and decoding messages.

For example: *strict*, *ignore*, *replace*, *xmlcharrefreplace*, *backslashreplace*

Type

str

property read_termination

The termination character sequence that is used for the *read()* method.

Reading stops when the equipment stops sending data or the *read_termination* character sequence is detected. If you set the *read_termination* to be equal to a variable of type *str* it will automatically be encoded.

Type

bytes or *None*

property write_termination

The termination character sequence that is appended to *write()* messages.

If you set the *write_termination* to be equal to a variable of type *str* it will automatically be encoded.

Type

bytes or *None*

property max_read_size

The maximum number of bytes that can be *read()*.

Type

int

property timeout

The timeout, in seconds, for *read()* and *write()* operations.

Type

float or *None*

property rstrip

Whether to remove trailing whitespace from *read()* messages.

Type

bool

property controller

`ConnectionSerial` or `ConnectionSocket`: The connection to the Prologix Controller for this equipment.

Depends on whether a GPIB-USB or a GPIB-ETHERNET Controller is being used to communicate with the equipment.

disconnect()

Calling this method does not close the underlying `ConnectionSerial` or `ConnectionSocket` connection to the Prologix Controller since the connection to the Prologix Controller may still be required to send messages to other devices via GPIB.

Calling this method sets the `controller` to be `None`.

group_execute_trigger(*addresses)

Send the Group Execute Trigger command to equipment at the specified addresses.

Up to 15 addresses may be specified. If no address is specified then the Group Execute Trigger command is issued to the currently-addressed equipment.

Parameters

addresses – The primary (and optional secondary) GPIB addresses. If a secondary address is specified then it must follow its corresponding primary address. For example:

- `group_execute_trigger(1, 11, 17)` → primary, primary, primary
- `group_execute_trigger(3, 96, 12, 21)` → primary, secondary, primary, primary

Returns

`int` – The number of bytes written.

read(kwargs)**

Read a message from the equipment.

Parameters

****kwargs** – All keyword arguments are passed to `read()`.

Returns

`str`, `bytes` or `ndarray` – The message from the equipment. If `dtype` is specified, then the message is returned as an `ndarray`, if `decode` is `True` then the message is returned as a `str`, otherwise the message is returned as `bytes`.

write(message, **kwargs)

Write a message to the equipment.

Parameters

- **message** (`str` or `bytes`) – The message to write to the equipment.
- ****kwargs** – All keyword arguments are passed to `write()`.

Returns

`int` – The number of bytes written.

query(message, **kwargs)

Convenience method for performing a `write()` followed by a `read()`.

Parameters

- **message** (`str` or `bytes`) – The message to write to the equipment.
- ****kwargs** – All keyword arguments are passed to `query()`.

Returns

`str`, `bytes` or `ndarray` – The message from the equipment. If `dtype` is specified, then the message is returned as an `ndarray`, if `decode` is `True` then the message is returned as a `str`, otherwise the message is returned as `bytes`.

property query_auto

Whether to send `++auto 1` before and `++auto 0` after a `query()` to the Prologix Controller.

Type

`bool`

version()

Get the version of the Prologix Controller.

Returns

`str` – The type of the Controller (GPIB-USB or GPIB-ETHERNET) and the version of the firmware.

static parse_address(address)

Parse the address to determine the connection class and the GPIB address.

Parameters

`address (str)` – The address of a `ConnectionRecord`.

Returns

`dict` or `None` – If `address` is valid for a Prologix connection then the key-value pairs are:

- **class**, `ConnectionSocket` or `ConnectionSerial`
The underlying connection class to use (not instantiated).
- **name**, `str`
The name of the connection class.
- **pad**, `int`
The primary GPIB address.
- **sad**, `int` or `None`
The secondary GPIB address.

otherwise `None` is returned.

`msl.equipment.connection_prologix.find_prologix(*, ip: list[str] | None = None, timeout: float = 1) → dict[str, str | list[str]]`

Find all Prologix ENET-GPIB devices that are on the network.

To resolve the MAC address of a Prologix device, the `arp` program must be installed. On Linux, install `net-tools`. On Windows and macOS, `arp` should already be installed.

Parameters

- **ip** – The IP address(es) on the local computer to use to search for Prologix ENET-GPIB devices. If not specified, uses all network interfaces.
- **timeout** – The maximum number of seconds to wait for a reply.

Returns

The information about the Prologix ENET-GPIB devices that were found.

msl.equipment.connection_pyvisa module

Uses PyVISA as the backend to communicate with the equipment.

```
class msl.equipment.connection_pyvisa.ConnectionPyVISA(record)
```

Bases: *Connection*

Uses PyVISA to establish a connection to the equipment.

The *backend* value must be equal to *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.

Do not instantiate this class directly. Use the *connect()* method to connect to the equipment.

Parameters

record (*EquipmentRecord*) – A record from an *Equipment-Register Database*.

property resource

The PyVISA resource that is used for the connection.

This is the **Resource** that would have been returned if you did the following in a script:

```
import pyvisa
rm = pyvisa.ResourceManager()
resource = rm.open_resource('ASRL3::INSTR')
```

Type

Resource

disconnect()

Calls *close()*.

static resource_manager(visa_library=None)

Return the PyVISA ResourceManager.

Parameters

visa_library (*VisaLibraryBase* or *str*, optional) – The library to use for PyVISA. For example:

- @ivi to use IVI
- @ni to use NI-VISA (only supported in PyVISA <1.11)
- @py to use PyVISA-py
- @sim to use PyVISA-sim

If *None* then *PyVISA_LIBRARY* will be used.

Returns

ResourceManager – The PyVISA Resource Manager.

Raises

- **ValueError** – If the PyVISA backend wrapper cannot be found.
- **OSError** – If an IVI library cannot be found.

static resource_class(record)

Get the PyVISA Resource class.

Parameters

record (*EquipmentRecord* or *ConnectionRecord*) – An equipment or connection record from a *Database*.

Returns

A *Resource* subclass – The PyVISA Resource class that can open the *record*.

msl.equipment.connection_sdk module

Base class for equipment that use the SDK provided by the manufacturer for the connection.

class msl.equipment.connection_sdk.ConnectionSDK(record, libtype, path=None)

Bases: *Connection*

Base class for equipment that use the SDK provided by the manufacturer for the connection.

The *backend* value must be equal to *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 *connect()* method to connect to the equipment.

Parameters

- **record** (*EquipmentRecord*) – A record from an *Equipment-Register Database*.
- **libtype** (*str*) – The library type. See *LoadLibrary* for more information.
- **path** (*str*, optional) – The path to the SDK (if *record.connection.address* does not contain this information).

Raises

- **OSError** – If the shared library cannot be loaded.
- **TypeError** – If either *record* or *libtype* is invalid.

property assembly

The reference to the .NET assembly.

Type

assembly

property gateway

The reference to the JAVA gateway.

Type

gateway

property path

The path to the SDK file.

Type

`str`

property sdk

The reference to the SDK object.

Type

`lib`

log_errcheck(result, func, arguments)

Convenience method for logging an `errcheck`

static parse_address(address)

Get the file path from an address.

Parameters

`address (str)` – The address of a `ConnectionRecord`.

Returns

`dict` or `None` – The file path or `None` if `address` is not valid for an SDK.

msl.equipment.connection_serial module

Base class for equipment that is connected through a serial port.

class msl.equipment.connection_serial.ConnectionSerial(record)

Bases: `ConnectionMessageBased`

Base class for equipment that is connected through a serial port.

The `properties` for a serial connection supports the following key-value pairs in the `Connections Database` (see also `serial.Serial` for more details about each parameter):

```
'baud_rate': int, the baud rate [default: 9600]
'data_bits': int, the number of data bits, e.g. 5, 6, 7, 8 [default: 8]
'dsr_dtr': bool, enable hardware (DSR/DTR) flow control [default: False]
'encoding': str, the encoding to use [default: 'utf-8']
'encoding_errors': str, encoding error handling scheme, e.g. 'strict',
    ↪'ignore' [default: 'strict']
'inter_byte_timeout': float or None, the inter-character timeout,
    ↪[default: None]
'max_read_size': int, the maximum number of bytes that can be read,
    ↪[default: 1 MB]
'parity': str or None, parity checking, e.g. 'even', 'odd' [default: None]
'read_termination': str or None, read until this termination sequence is
    ↪found [default: '\n']
'rstrip': bool, whether to remove trailing whitespace from "read"
    ↪messages [default: False]
'rts_cts': bool, enable hardware (RTS/CTS) flow control [default: False]
'stop_bits': int or float, the number of stop bits, e.g. 1, 1.5, 2
    ↪[default: 1]
```

(continues on next page)

(continued from previous page)

```
'termination': shortcut for setting both 'read_termination' and 'write_termination' to this value
'timeout': float or None, the timeout (in seconds) for read and write operations [default: None]
'write_termination': str or None, termination sequence appended to write messages [default: '\r\n']
'xon_xoff': bool, enable software flow control [default: False]
```

The *backend* value must be equal to *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 `connect()` method to connect to the equipment.

Parameters

record (*EquipmentRecord*) – A record from an *Equipment-Register Database*.

Raises

MSLConnectionError – If the serial port cannot be opened.

property serial

The reference to the serial object.

Type

serial.Serial

property baud_rate

The baud rate setting.

Type

int

property data_bits

The number of data bits.

Type

DataBits

property stop_bits

The stop bit setting.

Type

StopBits

property parity

The parity setting.

Type

Parity

disconnect()

Close the serial port.

static parse_address(address)

Get the serial port from an address.

Parameters

address (`str`) – The address of a `ConnectionRecord`

Returns

`dict` or `None` – The serial port in a format that is valid for PySerial (i.e., ASRL3 becomes COM3) or `None` if the port cannot be determined from `address`.

msl.equipment.connection_socket module

Base classes for equipment that is connected through a socket.

`class msl.equipment.connection_socket.ConnectionSocket(record)`

Bases: `ConnectionMessageBased`

Base class for equipment that is connected through a socket.

The `properties` for a socket connection supports the following key-value pairs in the *Connections Database* (see also `socket` for more details):

```
'buffer_size': int, the maximum number of bytes to read at a time
↳ [default: 4096]
'encoding': str, the encoding to use [default: 'utf-8']
'encoding_errors': str, encoding error handling scheme, e.g. 'strict',
↳ 'ignore' [default: 'strict']
'family': str, the address family, e.g., 'INET', 'INET6', 'IPX' [default:
↳ 'INET']
'max_read_size': int, the maximum number of bytes that can be read
↳ [default: 1 MB]
'proto': int, the socket protocol number [default: 0]
'read_termination': str or None, read until this termination sequence is
↳ found [default: '\n']
'rstrip': bool, whether to remove trailing whitespace from "read"
↳ messages [default: False]
'socket_type': str, the socket type, e.g. 'STREAM', 'DGRAM' [default:
↳ 'STREAM']
'termination': shortcut for setting both 'read_termination' and 'write_
↳ termination' to this value
'timeout': float or None, the timeout (in seconds) for read and write
↳ operations [default: None]
'write_termination': str or None, termination sequence appended to write
↳ messages [default: '\r\n']
```

The `backend` value must be equal to `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 `connect()` method to connect to the equipment.

Parameters

record (`EquipmentRecord`) – A record from an *Equipment-Register Database*.

Raises

`MSLConnectionError` – If the socket cannot be opened.

property byte_buffer

Returns the reference to the byte buffer.

Type

`bytarray`

property host

The host (IP address).

Type

`str`

property port

The port number.

Type

`int`

property socket

The reference to the socket.

Type

`socket`

static parse_address(address)

Get the host and port from an address.

Parameters

`address (str)` – The address of a *ConnectionRecord*.

Returns

`dict` or `None` – The value of the host and the port or `None` if `address` is not valid for a socket.

disconnect()

Close the socket.

reconnect(max_attempts=1)

Reconnect to the equipment.

Parameters

`max_attempts (int, optional)` – The maximum number of attempts to try to reconnect with the equipment. If < 1 or `None` then keep trying until a connection is successful. If the maximum number of attempts has been reached then an exception is raise.

msl.equipment.connection_tcpip_hislip module

Base class for equipment that use the HiSLIP communication protocol.

class msl.equipment.connection_tcpip_hislip.ConnectionTCPiPHiSliP(record)

Bases: `ConnectionMessageBased`

Base class for equipment that use the HiSLIP communication protocol.

The `properties` for a HiSLIP connection supports the following key-value pairs in the *Connections Database*:

```
'buffer_size': int, the maximum number of bytes to read at a time
↳ [default: 4096]
'encoding': str, the encoding to use [default: 'utf-8']
'encoding_errors': str, encoding error handling scheme, e.g. 'strict',
↳ 'ignore' [default: 'strict']
'lock_timeout': float or None, the timeout (in seconds) to wait for a
↳ lock [default: 0]
'max_read_size': int, the maximum number of bytes that can be read
↳ [default: 1 MB]
'rstrip': bool, whether to remove trailing whitespace from "read"
↳ messages [default: False]
'timeout': float or None, the timeout (in seconds) for read and write
↳ operations [default: None]
```

The `backend` value must be equal to `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 `connect()` method to connect to the equipment.

Parameters

`record (EquipmentRecord)` – A record from an *Equipment-Register Database*.

property host

The host (IP address).

Type

`str`

property port

The port number.

Type

`int`

property asynchronous

The reference to the asynchronous client.

Type

`AsyncClient`

property synchronous

The reference to the synchronous client.

Type

`SyncClient`

static parse_address(`address`)

Parse the address for valid TCPIP HiSLIP fields.

Parameters

`address (str)` – The address of a *ConnectionRecord*.

Returns

`dict` or `None` – The board number, hostname, LAN device name, and HiSLIP

port number of the device or `None` if *address* is not valid for a TCPIP HiSLIP connection.

property max_read_size

The maximum number of bytes that can be `read()`.

Type

`int`

property lock_timeout

The time, in seconds, to wait to acquire a lock.

Type

`float`

disconnect()

Close the connection to the HiSLIP server.

reconnect(max_attempts=1)

Reconnect to the equipment.

Parameters

max_attempts (`int`, optional) – The maximum number of attempts to try to reconnect with the equipment. If < 1 or `None` then keep trying until a connection is successful. If the maximum number of attempts has been reached then an exception is raise.

read_stb()

Read the status byte from the device.

Returns

`int` – The status byte.

trigger()

Send the trigger message (emulates a GPIB Group Execute Trigger event).

clear()

Send the *clear* command to the device.

lock(lock_string="")

Acquire the device's lock.

Parameters

lock_string (`str`, optional) – An ASCII string that identifies this lock. If not specified, then an exclusive lock is requested, otherwise the string indicates an identification of a shared-lock request.

Returns

`bool` – Whether acquiring the lock was successful.

unlock()

Release the lock acquired by `lock()`.

Returns

`bool` – Whether releasing the lock was successful.

lock_status()

Request the lock status from the HiSLIP server.

Returns

- `bool` – Whether the HiSLIP server has an exclusive lock with a client.
- `int` – The number of HiSLIP clients that have a lock with the HiSLIP server.

remote_local_control(*request*)

Send a GPIB-like remote/local control request.

Parameters

`request (int)` – The request to perform.

- 0 – Disable remote, `VI_GPIB_REN_DEASSERT`
- 1 – Enable remote, `VI_GPIB_REN_ASSERT`
- 2 – Disable remote and go to local, `VI_GPIB_REN_DEASSERT_GTL`
- 3 – Enable Remote and go to remote, `VI_GPIB_REN_ASSERT_ADDRESS`
- 4 – Enable remote and lock out local, `VI_GPIB_REN_ASSERT_LLO`
- 5 – Enable remote, go to remote, and set local lockout, `VI_GPIB_REN_ASSERT_ADDRESS_LLO`
- 6 – go to local without changing REN or lockout state, `VI_GPIB_REN_ADDRESS_GTL`

msl.equipment.connection_tcpip_vxi11 module

Base class for equipment that use the VXI-11 communication protocol.

`class msl.equipment.connection_tcpip_vxi11.ConnectionTCPVXI11(record)`

Bases: `ConnectionMessageBased`

Base class for equipment that use the VXI-11 communication protocol.

The `properties` for a VXI-11 connection supports the following key-value pairs in the `Connections Database`:

```
'buffer_size': int, the maximum number of bytes to read at a time
↳[default: 4096]
'encoding': str, the encoding to use [default: 'utf-8']
'encoding_errors': str, encoding error handling scheme, e.g. 'strict',
↳'ignore' [default: 'strict']
'lock_timeout': float or None, the timeout (in seconds) to wait for a
↳lock [default: 0]
'max_read_size': int, the maximum number of bytes that can be read
↳[default: 1 MB]
'port': int, the port to use instead of calling the RPC Port Mapper
↳function [default: None]
'read_termination': str or None, read until this termination character is
↳found [default: None]
'rstrip': bool, whether to remove trailing whitespace from "read"
```

(continues on next page)

(continued from previous page)

```

→messages [default: False]
'termination': alias for 'read_termination'
'timeout': float or None, the timeout (in seconds) for read and write
→operations [default: None]

```

The `backend` value must be equal to `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 `connect()` method to connect to the equipment.

Parameters

`record (EquipmentRecord)` – A record from an *Equipment-Register Database*.

property byte_buffer

Returns the reference to the byte buffer.

Type

`bytearray`

property host

The host (IP address).

Type

`str`

property port

The port number.

Type

`int`

property socket

The reference to the socket.

Type

`socket`

static parse_address(address)

Parse the address for valid TCPIP VXI-11 fields.

Parameters

`address (str)` – The address of a *ConnectionRecord*.

Returns

`dict` or `None` – The board number, hostname, and LAN device name of the device or `None` if `address` is not valid for a TCPIP VXI-11 connection.

disconnect()

Unlink and close the sockets.

reconnect(max_attempts=1)

Reconnect to the equipment.

Parameters

`max_attempts (int, optional)` – The maximum number of attempts to try to reconnect with the equipment. If < 1 or `None` then keep trying until a connection

is successful. If the maximum number of attempts has been reached then an exception is raised.

property lock_timeout

The time, in seconds, to wait to acquire a lock.

Type

`float`

abort()

Stop an in-progress request.

read_stb()

Read the status byte from the device.

Returns

`int` – The status byte.

trigger()

Send a trigger to the device.

clear()

Send the *clear* command to the device.

remote()

Place the device in a remote state wherein all programmable local controls are disabled.

local()

Place the device in a local state wherein all programmable local controls are enabled.

lock()

Acquire the device's lock.

unlock()

Release the lock acquired by *lock()*

enable_sqr(enable, handle)

Enable or disable the sending of *device_intr_sq* RPCs by the network instrument server.

Parameters

- **enable (bool)** – Whether to enable or disable interrupts.
- **handle (bytes)** – Host specific data (maximum length is 40 characters).

docmd(cmd, value, fmt)

Allows for a variety of commands to be executed.

Parameters

- **cmd (int)** – An IEEE 488 command messages, (e.g., to send a group execute trigger, GET, command the value of *cmd* would be 0x08).
- **value (bool, int or float)** – The value to use with *cmd*.
- **fmt (str)** – How to format *value*. See [Format Characters](#) for more details. Do not include the byte-order character. Network (big-endian) order will always be used.

Returns

`bytes` – The results defined by `cmd`.

destroy_link()

Destroy the link with the device.

create_intr_chan(*host_addr*, *host_port*, *prog_num*, *prog_ver*, *prog_family*)

Inform the network instrument server to establish an interrupt channel.

Parameters

- **host_addr** (`int`) – Host servicing the interrupt.
- **host_port** (`int`) – Valid port number on the client.
- **prog_num** (`int`) – Program number.
- **prog_ver** (`int`) – Program version number.
- **prog_family** (`int`) – The underlying socket protocol family type (IPPROTO_TCP or IPPROTO_UDP).

destroy_intr_chan()

Inform the network instrument server to close its interrupt channel.

msl.equipment.connection_zeromq module

Base class for equipment that use the [ZeroMQ](#) communication protocol.

class msl.equipment.connection_zeromq.ConnectionZeroMQ(*record*)

Bases: [ConnectionMessageBased](#)

Base class for equipment that use the [ZeroMQ](#) communication protocol.

The [properties](#) for a ZeroMQ connection supports the following key-value pairs in the [Connections Database](#):

```
'encoding': str, the encoding to use [default: 'utf-8']
'encoding_errors': str, encoding error handling scheme, e.g. 'strict',
    ↳ 'ignore' [default: 'strict']
'max_read_size': int, the maximum number of bytes that can be read,
    ↳ [default: 1 MB]
'protocol': str, the ZeroMQ protocol [default: 'tcp']
'rstrip': bool, whether to remove trailing whitespace from "read"
    ↳ messages [default: False]
'socket_type': str, the ZeroMQ socket type [default: 'REQ']
'timeout': float or None, the timeout (in seconds) for read and write
    ↳ operations [default: None]
```

The [backend](#) value must be equal to [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 `connect()` method to connect to the equipment.

Parameters

record ([EquipmentRecord](#)) – A record from an [Equipment-Register Database](#).

Raises

MSLConnectionError – If the socket cannot be opened.

property context

Reference to the ZeroMQ context.

Type

`Context`

disconnect()

Close the connection.

property host

The host (IP address).

Type

`str`

property max_read_size

The maximum number of bytes that can be `read()`.

Type

`int`

static parse_address(address)

Parse the address for valid ZeroMQ fields.

Parameters

address (`str`) – The address of a `ConnectionRecord`.

Returns

`dict` or `None` – The host and port number of the device or `None` if `address` is not valid for a ZeroMQ connection.

property port

The port number.

Type

`int`

reconnect(max_attempts=1)

Reconnect to the equipment.

Parameters

max_attempts (`int`, optional) – The maximum number of attempts to try to reconnect with the equipment. If < 1 or `None` then keep trying until a connection is successful. If the maximum number of attempts has been reached then an exception is raise.

property socket

Reference to the ZeroMQ socket.

Type

`Socket`

msl.equipment.constants module

MSL-Equipment constants.

```
class msl.equipment.constants.Backend(value, names=None, *values, module=None,
                                         qualname=None, type=None, start=1,
                                         boundary=None)
```

Bases: [IntEnum](#)

The software backend to use for the communication system.

UNKNOWN = 0

MSL = 1

PyVISA = 2

NIDAQ = 3

```
class msl.equipment.constants.Interface(value, names=None, *values, module=None,
                                         qualname=None, type=None, start=1,
                                         boundary=None)
```

Bases: [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 backend.

NONE = 0

SDK = 1

SERIAL = 2

SOCKET = 3

PROLOGIX = 4

TCPIP_VXI11 = 5

TCPIP_HISLIP = 6

ZMQ = 7

GPIB = 8

```
class msl.equipment.constants.Parity(value, names=None, *values, module=None,
                                         qualname=None, type=None, start=1,
                                         boundary=None)
```

Bases: [Enum](#)

The parity type to use for Serial communication.

NONE = 'N'

ODD = 'O'

EVEN = 'E'

```
MARK = 'M'
```

```
SPACE = 'S'
```

```
class msl.equipment.constants.StopBits(value, names=None, *values, module=None,
                                         qualname=None, type=None, start=1,
                                         boundary=None)
```

Bases: [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(value, names=None, *values, module=None,
                                         qualname=None, type=None, start=1,
                                         boundary=None)
```

Bases: [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 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*.

This class should be accessed through the `database()` method after a *Config* object has been created.

Parameters

`path (str)` – The path to an XML *Configuration File*.

Raises

- **OSError** – If `path` does not exist or if the *Configuration File* is invalid.
- **AttributeError** – If an `<equipment>` XML tag is specified in the *Configuration File* and it does not uniquely identify an equipment record in an *Equipment-Register Database*.

- **ValueError** – If an *alias* has been specified multiple times for the same *EquipmentRecord* or if the name of the Sheet in an Excel spreadsheet is invalid.

property equipment

EquipmentRecord's that were listed as <equipment> XML tags in the *Configuration File*.

Type

`dict`

property path

The path to the *Configuration File*.

Type

`str`

connections(**kwargs)

Search the *Connections Database* to find all *ConnectionRecord*'s that match the specified criteria.

Parameters

****kwargs** – The argument names can be any of the *ConnectionRecord* property names or a *flags* argument of type `int` for performing the search, see `re.search()`. For testing regex expressions online you can use [this website](#).

If a *kwarg* is *properties* then the value must be a `dict`. See the examples below.

Examples

- *connections()* → a list of all ConnectionRecord's
- *connections(manufacturer='Keysight')* → a list of all ConnectionRecord's that have Keysight as the manufacturer
- *connections(manufacturer='Agilent|Keysight')* → a list of all ConnectionRecord's that are from Agilent or Keysight
- *connections(manufacturer='H.*P')* → a list of all ConnectionRecord's that have Hewlett Packard (or HP) as the manufacturer
- *connections(manufacturer='Agilent', model='^34')* → a list of all ConnectionRecord's that have Agilent as the manufacturer and a model number beginning with '34'
- *connections(interface=Interface.SERIAL)* → a list of all ConnectionRecord's that use SERIAL for the connection bus
- *connections(interface='SDK')* → a list of all ConnectionRecord's that use the manufacturers SDK to control the equipment
- *connections(backend=Backend.PyVISA)* → a list of all ConnectionRecord's that use PyVISA as the backend
- *connections(backend='MSL')* → a list of all ConnectionRecord's that use MSL as the backend
- *connections(properties={'baud_rate': 115200})* → a list of all ConnectionRecord's that specify a baud rate equal to 115200 in the Properties field

Returns

`list` of `ConnectionRecord` – The connection records that match the search criteria.

Raises

`NameError` – If the name of an input argument is not a `ConnectionRecord` property name or `flags`.

records(kwargs)**

Search the *Equipment-Register Database* to find all `EquipmentRecord`'s that match the specified criteria.

Parameters

`**kwargs` – The argument names can be any of the `EquipmentRecord` property names or a `flags` argument of type `int` for performing the search, see `re.search()`. For testing regex expressions online you can use [this website](#).

If a `kwarg` is `connection` then the value will be used to test which `EquipmentRecord`'s have a `connection` value that is either `None` or `ConnectionRecord`. See the examples below.

Examples

- `records()` → a list of all `EquipmentRecord`'s
- `records(manufacturer='Agilent')` → a list of all `EquipmentRecord`'s that are from Agilent
- `records(manufacturer='Agilent|Keysight')` → a list of all `EquipmentRecord`'s that are from Agilent or Keysight
- `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(manufacturer=r'H.*P')` → a list of all `EquipmentRecord`'s that have Hewlett Packard (or HP) as the manufacturer
- `records(description='I-V Converter')` → a list of all `EquipmentRecords` that contain 'I-V Converter' in the description field
- `records(connection=True)` → a list of all `EquipmentRecords` that can be connected to

Returns

`list` of `EquipmentRecord` – The equipment records that match the search criteria.

Raises

`NameError` – If the name of an input argument is not an `EquipmentRecord` property name or `flags`.

msl.equipment.dns_service_discovery module

Implementation of the Multicast DNS and DNS-Based Service Discovery protocols.

References

- RFC-1035 – *Domain Names - Implementation and Specification*, **ISI**, November 1987.
- RFC-6762 – *Multicast DNS*, **Apple Inc.**, February 2013.
- RFC-6763 – *DNS-Based Service Discovery*, **Apple Inc.**, February 2013.

`msl.equipment.dns_service_discovery.find_lxi(*, ip: list[str] | None = None, timeout: float = 1) → dict[str, dict[str, str | list[str]]]`

Find all LXI devices that support the mDNS and DNS Service Discovery protocols.

Parameters

- **ip** – The IP address(es) on the local computer to use to broadcast the discovery message. If not specified, broadcast on all network interfaces.
- **timeout** – The maximum number of seconds to wait for a reply.

Returns

The information about the HiSLIP, VXI-11 and SCPI-RAW devices that were found.

msl.equipment.exceptions module

Exceptions used by MSL-Equipment.

exception msl.equipment.exceptions.MSLConnectionError

Bases: `OSError`

Base class for all MSL `Connection` exceptions.

exception msl.equipment.exceptions.MSLTimeoutError

Bases: `MSLConnectionError`

A timeout exception for I/O operations.

exception msl.equipment.exceptions.ResourceClassNotFound(record)

Bases: `MSLConnectionError`

Exception if a resource class cannot be found to connect to the equipment.

exception msl.equipment.exceptions.AimTTiError

Bases: `MSLConnectionError`

Exception for equipment from Aim and Thurlby Thandar Instruments.

exception msl.equipment.exceptions.AvantesError

Bases: `MSLConnectionError`

Exception for equipment from Avantes.

```
exception msl.equipment.exceptions.BenthamError
```

Bases: [MSLConnectionError](#)

Exception for equipment from Bentham.

```
exception msl.equipment.exceptions.CMIErro
```

Bases: [MSLConnectionError](#)

Exception for equipment from the Czech Metrology Institute.

```
exception msl.equipment.exceptions.DataRayError
```

Bases: [MSLConnectionError](#)

Exception for equipment from DataRay Inc.

```
exception msl.equipment.exceptions.EnergetiqError
```

Bases: [MSLConnectionError](#)

Exception for equipment from Energetiq.

```
exception msl.equipment.exceptions.GPIBError(message: str, *, name: str = "", ibsta: int = -1, iberr: int = -1)
```

Bases: [MSLConnectionError](#)

Exception for equipment that use the GPIB interface.

Parameters

- **message** – The error message.
- **name** – The GPIB function name.
- **ibsta** – The status value.
- **iberr** – The error code.

```
exception msl.equipment.exceptions.GreisingerError
```

Bases: [MSLConnectionError](#)

Exception for equipment from Greisinger.

```
exception msl.equipment.exceptions.MKSInstrumentsError
```

Bases: [MSLConnectionError](#)

Exception for equipment from MKS Instruments.

```
exception msl.equipment.exceptions.NKTError
```

Bases: [MSLConnectionError](#)

Exception for equipment from NKT Photonics.

```
exception msl.equipment.exceptions.OmegaError
```

Bases: [MSLConnectionError](#)

Exception for equipment from OMEGA.

```
exception msl.equipment.exceptions.OptoSigmaError
```

Bases: [MSLConnectionError](#)

Exception for equipment from OptoSigma.

exception `msl.equipment.exceptions.OptronicLaboratoriesError`

Bases: `MSLConnectionError`

Exception for equipment from Optronic Laboratories.

exception `msl.equipment.exceptions.PicoTechError`

Bases: `MSLConnectionError`

Exception for equipment from Pico Technology.

exception `msl.equipment.exceptions.PrincetonInstrumentsError`

Bases: `MSLConnectionError`

Exception for equipment from Princeton Instruments.

exception `msl.equipment.exceptions.RaicolCrystalsError`

Bases: `MSLConnectionError`

Exception for equipment from Raicol Crystals.

exception `msl.equipment.exceptions.ThorlabsError`

Bases: `MSLConnectionError`

Exception for equipment from Thorlabs.

msl.equipment.factory module

Establish a connection to the equipment.

`msl.equipment.factory.connect(record, demo=None)`

Factory function to establish a connection to the equipment.

Parameters

- **record** (`EquipmentRecord`) – A record from an *Equipment-Register Database*.
- **demo** (`bool`, optional) – Whether to simulate a connection to the equipment by opening a connection in demo mode. This allows you to test your code if the equipment is not physically connected to a computer.
If `None` then the `demo` value is determined from the `DEMO_MODE` attribute.

Returns

A `Connection` subclass.

`msl.equipment.factory.find_interface(address: str) → Interface`

Find the interface for `address`.

Parameters

address – The address of a `ConnectionRecord`.

msl.equipment.hislip module

Implementation of the HiSLIP protocol for a client.

This module implements the following IVI Protocol Specification:

IVI-6.1: High-Speed LAN Instrument Protocol (HiSLIP) v2.0 April 23, 2020

```
class msl.equipment.hislip.MessageType(value, names=None, *values, module=None,
                                         qualname=None, type=None, start=1,
                                         boundary=None)
```

Bases: `IntEnum`

Message types.

`Initialize = 0`

`InitializeResponse = 1`

`FatalError = 2`

`Error = 3`

`AsyncLock = 4`

`AsyncLockResponse = 5`

`Data = 6`

`DataEnd = 7`

`DeviceClearComplete = 8`

`DeviceClearAcknowledge = 9`

`AsyncRemoteLocalControl = 10`

`AsyncRemoteLocalResponse = 11`

`Trigger = 12`

`Interrupted = 13`

`AsyncInterrupted = 14`

`AsyncMaximumMessageSize = 15`

`AsyncMaximumMessageSizeResponse = 16`

`AsyncInitialize = 17`

`AsyncInitializeResponse = 18`

`AsyncDeviceClear = 19`

`AsyncServiceRequest = 20`

`AsyncStatusQuery = 21`

```
AsyncStatusResponse = 22
AsyncDeviceClearAcknowledge = 23
AsyncLockInfo = 24
AsyncLockInfoResponse = 25
GetDescriptors = 26
GetDescriptorsResponse = 27
StartTLS = 28
AsyncStartTLS = 29
AsyncStartTLSResponse = 30
EndTLS = 31
AsyncEndTLS = 32
AsyncEndTLSResponse = 33
GetSaslMechanismList = 34
GetSaslMechanismListResponse = 35
AuthenticationStart = 36
AuthenticationExchange = 37
AuthenticationResult = 38

class msl.equipment.hislip.ErrorType(value, names=None, *values, module=None,
                                      qualname=None, type=None, start=1,
                                      boundary=None)
Bases: IntEnum
Error types.

UNIDENTIFIED = 0
BAD_HEADER = 1
CHANNELS_INACTIVATED = 2
INVALID_INIT_SEQUENCE = 3
MAX_CLIENTS = 4
BAD_MESSAGE_TYPE = 1
BAD_CONTROL_CODE = 2
BAD_VENDOR = 3
MESSAGE_TOO_LARGE = 4
```

AUTHENTICATION_FAILED = 5

```
exception msl.equipment.hislip.HiSLIPException(message_type, control_code,
                                                reason=None)
```

Bases: [Exception](#)

Base class for HiSLIP exceptions.

Parameters

- **message_type** ([MessageType](#)) – The message type.
- **control_code** ([int](#)) – The control code from the server response.
- **reason** ([str](#)) – Additional information to display in exception string.

property **message**

The error message that can be written to the server.

Type

[Message](#)

```
exception msl.equipment.hislip.FatalError(control_code, reason=None)
```

Bases: [HiSLIPException](#)

Exception for a fatal error.

Parameters

- **control_code** ([int](#)) – The control code from the server response.
- **reason** ([str](#)) – Additional information to display in exception string.

```
exception msl.equipment.hislip.Error(control_code, reason=None)
```

Bases: [HiSLIPException](#)

Exception for a non-fatal error.

Parameters

- **control_code** ([int](#)) – The control code from the server response.
- **reason** ([str](#)) – Additional information to display in exception string.

```
class msl.equipment.hislip.Message(control_code=0, parameter=0, payload=b'')
```

Bases: [object](#)

Create a new HiSLIP message.

Parameters

- **control_code** ([int](#), optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.
- **parameter** ([int](#), optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.
- **payload** ([bytes](#), optional) – The payload data.

header = <[_struct.Struct](#) object>

prologue = b'HS'

`type = None`

property length_payload

The length of the payload.

Type

`int`

pack()

Convert the message to bytes.

Returns

`bytearray` – The messaged packed as bytes.

static repack(unpack_fmt, pack_fmt, *args)

Convert arguments from one byte format to another.

Parameters

- **unpack_fmt** (`str`) – The format to convert the arguments to.
- **pack_fmt** (`str`) – The format that the arguments are currently in.
- ***args** – The arguments to convert.

Returns

`tuple` – The converted arguments.

property size

`int` The total size of the message.

`class msl.equipment.hislip.FatalErrorMessage(control_code=0, parameter=0, payload=b'')`

Bases: `Message`

Create a new HiSLIP message.

Parameters

- **control_code** (`int`, optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.
- **parameter** (`int`, optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.
- **payload** (`bytes`, optional) – The payload data.

`type = 2`

`class msl.equipment.hislip.ErrorMessage(control_code=0, parameter=0, payload=b'')`

Bases: `Message`

Create a new HiSLIP message.

Parameters

- **control_code** (`int`, optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.
- **parameter** (`int`, optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.

- **payload** (`bytes`, optional) – The payload data.

`type = 3`

`class msl.equipment.hislip.Initialize(major, minor, client_id, sub_address)`

Bases: `Message`

Create an Initialize message.

Parameters

- **major** (`int`) – The major version number of the HiSLIP protocol that the client supports.
- **minor** (`int`) – The minor version number of the HiSLIP protocol that the client supports.
- **client_id** (`bytes`) – The vendor ID of the client. Must have a length of 2 characters.
- **sub_address** (`bytes`) – A particular device managed by this server. For VISA clients this field corresponds to the VISA LAN device name (default is `hislip0`). The maximum length is 256 characters.

`type = 0`

`class msl.equipment.hislip.InitializeResponse(control_code=0, parameter=0, payload=b'')`

Bases: `Message`

Create a new HiSLIP message.

Parameters

- **control_code** (`int`, optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.
- **parameter** (`int`, optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.
- **payload** (`bytes`, optional) – The payload data.

`type = 1`

`property encrypted`

Whether encryption is optional or mandatory.

Type

`bool`

`property initial_encryption`

Whether the client shall switch to encrypted mode.

Type

`bool`

`property overlapped`

Whether the server is in overlapped or synchronous mode.

Type

`bool`

property protocol_version

The (major, minor) version numbers of the HiSLIP protocol that the client and server are to use.

Type

`tuple`

property session_id

The session ID.

Type

`int`

`class msl.equipment.hislip.Data(control_code=0, parameter=0, payload=b'')`

Bases: [Message](#)

Create a new HiSLIP message.

Parameters

- **control_code** (`int`, optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.
- **parameter** (`int`, optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.
- **payload** (`bytes`, optional) – The payload data.

`type = 6`

`class msl.equipment.hislip.DataEnd(control_code=0, parameter=0, payload=b'')`

Bases: [Message](#)

Create a new HiSLIP message.

Parameters

- **control_code** (`int`, optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.
- **parameter** (`int`, optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.
- **payload** (`bytes`, optional) – The payload data.

`type = 7`

`class msl.equipment.hislip.AsyncLock(control_code=0, parameter=0, payload=b'')`

Bases: [Message](#)

Create a new HiSLIP message.

Parameters

- **control_code** (`int`, optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.
- **parameter** (`int`, optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.
- **payload** (`bytes`, optional) – The payload data.

type = 4

```
class msl.equipment.hislip.AsyncLockResponse(control_code=0, parameter=0,
                                             payload=b'')
```

Bases: [Message](#)

Create a new HiSLIP message.

Parameters

- **control_code** ([int](#), optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.
- **parameter** ([int](#), optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.
- **payload** ([bytes](#), optional) – The payload data.

type = 5

property error

Whether the request was an invalid attempt to release a lock that was not acquired or to request a lock already granted.

Type

[bool](#)

property failed

Whether a lock was requested but not granted (timeout expired).

Type

[bool](#)

property success

Whether requesting or releasing the lock was successful.

Type

[bool](#)

property shared_released

Whether releasing a shared lock was successful.

Type

[bool](#)

```
class msl.equipment.hislip.AsyncLockInfo(control_code=0, parameter=0, payload=b'')
```

Bases: [Message](#)

Create a new HiSLIP message.

Parameters

- **control_code** ([int](#), optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.
- **parameter** ([int](#), optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.
- **payload** ([bytes](#), optional) – The payload data.

type = 24

```
class msl.equipment.hislip.AsyncLockInfoResponse(control_code=0, parameter=0,
                                                payload=b'')
```

Bases: [Message](#)

Create a new HiSLIP message.

Parameters

- **control_code** ([int](#), optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.
- **parameter** ([int](#), optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.
- **payload** ([bytes](#), optional) – The payload data.

type = 25

property exclusive

Whether the HiSLIP server has an exclusive lock with a client.

Type

[bool](#)

property num_locks

The number of HiSLIP clients that have a lock with the HiSLIP server.

Type

[int](#)

```
class msl.equipment.hislip.AsyncRemoteLocalControl(control_code=0, parameter=0,
                                                payload=b'')
```

Bases: [Message](#)

Create a new HiSLIP message.

Parameters

- **control_code** ([int](#), optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.
- **parameter** ([int](#), optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.
- **payload** ([bytes](#), optional) – The payload data.

type = 10

```
class msl.equipment.hislip.AsyncRemoteLocalResponse(control_code=0, parameter=0,
                                                payload=b'')
```

Bases: [Message](#)

Create a new HiSLIP message.

Parameters

- **control_code** ([int](#), optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.

- **parameter** (`int`, optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.
- **payload** (`bytes`, optional) – The payload data.

`type = 11`

`class msl.equipment.hislip.AsyncDeviceClear(control_code=0, parameter=0, payload=b'')`

Bases: [Message](#)

Create a new HiSLIP message.

Parameters

- **control_code** (`int`, optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.
- **parameter** (`int`, optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.
- **payload** (`bytes`, optional) – The payload data.

`type = 19`

`class msl.equipment.hislip.AsyncDeviceClearAcknowledge(control_code=0, parameter=0, payload=b'')`

Bases: [Message](#)

Create a new HiSLIP message.

Parameters

- **control_code** (`int`, optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.
- **parameter** (`int`, optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.
- **payload** (`bytes`, optional) – The payload data.

`type = 23`

`property feature_bitmap`

The feature bitmap that the server prefers.

Type

`int`

`class msl.equipment.hislip.DeviceClearComplete(control_code=0, parameter=0, payload=b'')`

Bases: [Message](#)

Create a new HiSLIP message.

Parameters

- **control_code** (`int`, optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.
- **parameter** (`int`, optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.

- **payload** (`bytes`, optional) – The payload data.

`type = 8`

```
class msl.equipment.hislip.DeviceClearAcknowledge(control_code=0, parameter=0,
                                                payload=b'')
```

Bases: `Message`

Create a new HiSLIP message.

Parameters

- **control_code** (`int`, optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.
- **parameter** (`int`, optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.
- **payload** (`bytes`, optional) – The payload data.

`type = 9`

```
class msl.equipment.hislip.Trigger(control_code=0, parameter=0, payload=b'')
```

Bases: `Message`

Create a new HiSLIP message.

Parameters

- **control_code** (`int`, optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.
- **parameter** (`int`, optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.
- **payload** (`bytes`, optional) – The payload data.

`type = 12`

```
class msl.equipment.hislip.AsyncMaximumMessageSize(control_code=0, parameter=0,
                                                    payload=b'')
```

Bases: `Message`

Create a new HiSLIP message.

Parameters

- **control_code** (`int`, optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.
- **parameter** (`int`, optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.
- **payload** (`bytes`, optional) – The payload data.

`type = 15`

```
class msl.equipment.hislip.AsyncMaximumMessageSizeResponse(control_code=0,
                                                          parameter=0,
                                                          payload=b'')
```

Bases: [Message](#)

Create a new HiSLIP message.

Parameters

- **control_code** (`int`, optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.
- **parameter** (`int`, optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.
- **payload** (`bytes`, optional) – The payload data.

`type = 16`

property maximum_message_size

The maximum message size that the server's synchronous channel accepts.

Type

`int`

`class msl.equipment.hislip.GetDescriptors(control_code=0, parameter=0, payload=b'')`

Bases: [Message](#)

Create a new HiSLIP message.

Parameters

- **control_code** (`int`, optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.
- **parameter** (`int`, optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.
- **payload** (`bytes`, optional) – The payload data.

`type = 26`

`class msl.equipment.hislip.GetDescriptorsResponse(control_code=0, parameter=0, payload=b'')`

Bases: [Message](#)

Create a new HiSLIP message.

Parameters

- **control_code** (`int`, optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.
- **parameter** (`int`, optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.
- **payload** (`bytes`, optional) – The payload data.

`type = 27`

`class msl.equipment.hislip.AsyncInitialize(control_code=0, parameter=0, payload=b'')`

Bases: [Message](#)

Create a new HiSLIP message.

Parameters

- **control_code** (`int`, optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.
- **parameter** (`int`, optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.
- **payload** (`bytes`, optional) – The payload data.

`type = 17`

```
class msl.equipment.hislip.AsyncInitializeResponse(control_code=0, parameter=0,
                                                    payload=b'')
```

Bases: `Message`

Create a new HiSLIP message.

Parameters

- **control_code** (`int`, optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.
- **parameter** (`int`, optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.
- **payload** (`bytes`, optional) – The payload data.

`type = 18`

`SECURE_CONNECTION_SUPPORTED = 1`

property secure_connection_supported

Whether secure connection capability is supported.

Type

`bool`

property server_vendor_id

The two-character vendor abbreviation of the server.

Type

`bytes`

```
class msl.equipment.hislip.AsyncStatusQuery(control_code=0, parameter=0, payload=b'')
```

Bases: `Message`

Create a new HiSLIP message.

Parameters

- **control_code** (`int`, optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.
- **parameter** (`int`, optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.
- **payload** (`bytes`, optional) – The payload data.

`type = 21`

```
class msl.equipment.hislip.AsyncStatusResponse(control_code=0, parameter=0,  
                                              payload=b'')
```

Bases: [Message](#)

Create a new HiSLIP message.

Parameters

- **control_code** ([int](#), optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.
- **parameter** ([int](#), optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.
- **payload** ([bytes](#), optional) – The payload data.

type = 22

property status

The status value.

Type

[int](#)

```
class msl.equipment.hislip.StartTLS(control_code=0, parameter=0, payload=b'')
```

Bases: [Message](#)

Create a new HiSLIP message.

Parameters

- **control_code** ([int](#), optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.
- **parameter** ([int](#), optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.
- **payload** ([bytes](#), optional) – The payload data.

type = 28

```
class msl.equipment.hislip.AsyncStartTLS(control_code=0, parameter=0, payload=b'')
```

Bases: [Message](#)

Create a new HiSLIP message.

Parameters

- **control_code** ([int](#), optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.
- **parameter** ([int](#), optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.
- **payload** ([bytes](#), optional) – The payload data.

type = 29

```
class msl.equipment.hislip.AsyncStartTLSResponse(control_code=0, parameter=0,  
                                                payload=b'')
```

Bases: [Message](#)

Create a new HiSLIP message.

Parameters

- **control_code** ([int](#), optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.
- **parameter** ([int](#), optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.
- **payload** ([bytes](#), optional) – The payload data.

type = 30

property busy

Whether the server is busy.

Type

[bool](#)

property success

Whether the request was successful.

Type

[bool](#)

property error

Whether there was an error processing the request.

Type

[bool](#)

```
class msl.equipment.hislip.EndTLS(control_code=0, parameter=0, payload=b'')
```

Bases: [Message](#)

Create a new HiSLIP message.

Parameters

- **control_code** ([int](#), optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.
- **parameter** ([int](#), optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.
- **payload** ([bytes](#), optional) – The payload data.

type = 31

```
class msl.equipment.hislip.AsyncEndTLS(control_code=0, parameter=0, payload=b'')
```

Bases: [Message](#)

Create a new HiSLIP message.

Parameters

- **control_code** (`int`, optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.
- **parameter** (`int`, optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.
- **payload** (`bytes`, optional) – The payload data.

`type = 32`

```
class msl.equipment.hislip.AsyncEndTLSResponse(control_code=0, parameter=0,
                                                payload=b'')
```

Bases: [Message](#)

Create a new HiSLIP message.

Parameters

- **control_code** (`int`, optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.
- **parameter** (`int`, optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.
- **payload** (`bytes`, optional) – The payload data.

`type = 33`

property busy

Whether the server is busy.

Type
 `bool`

property success

Whether the request was successful.

Type
 `bool`

property error

Whether there was an error processing the request.

Type
 `bool`

```
class msl.equipment.hislip.GetSaslMechanismList(control_code=0, parameter=0,
                                                payload=b'')
```

Bases: [Message](#)

Create a new HiSLIP message.

Parameters

- **control_code** (`int`, optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.
- **parameter** (`int`, optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.
- **payload** (`bytes`, optional) – The payload data.

type = 34

```
class msl.equipment.hislip.GetSaslMechanismListResponse(control_code=0,
                                                       parameter=0, payload=b'')
```

Bases: [Message](#)

Create a new HiSLIP message.

Parameters

- **control_code** ([int](#), optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.
- **parameter** ([int](#), optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.
- **payload** ([bytes](#), optional) – The payload data.

type = 35

property **data**

List of SASL mechanisms.

Type

[list](#)

```
class msl.equipment.hislip.AuthenticationStart(control_code=0, parameter=0,
                                                payload=b'')
```

Bases: [Message](#)

Create a new HiSLIP message.

Parameters

- **control_code** ([int](#), optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.
- **parameter** ([int](#), optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.
- **payload** ([bytes](#), optional) – The payload data.

type = 36

```
class msl.equipment.hislip.AuthenticationExchange(control_code=0, parameter=0,
                                                payload=b'')
```

Bases: [Message](#)

Create a new HiSLIP message.

Parameters

- **control_code** ([int](#), optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.
- **parameter** ([int](#), optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.
- **payload** ([bytes](#), optional) – The payload data.

type = 37

```
class msl.equipment.hislip.AuthenticationResult(control_code=0, parameter=0,
                                                payload=b'')
```

Bases: [Message](#)

Create a new HiSLIP message.

Parameters

- **control_code** ([int](#), optional) – This 8-bit field is a general parameter for the message. If the field is not defined for a message, 0 shall be sent.
- **parameter** ([int](#), optional) – This 32-bit field has various uses in different messages. If this field is not defined for a message, 0 shall be sent.
- **payload** ([bytes](#), optional) – The payload data.

type = 38

property data

Additional data returned by the server.

Type
 [bytes](#)

property error

Whether there was an error processing the request.

Type
 [bool](#)

property error_code

If authentication fails, the mechanism-dependent error code.

Type
 [int](#)

property success

Whether the request was successful.

Type
 [bool](#)

```
class msl.equipment.hislip.HiSLIPClient(host)
```

Bases: [object](#)

Base class for a HiSLIP client.

Parameters

host ([str](#)) – The hostname or IP address of the remote device.

close()

Close the TCP socket, if one is open.

connect(*port*=4880, *timeout*=10)

Connect to a specific port of the device.

Parameters

- **port** (`int`) – The port number to connect to.
- **timeout** (`float` or `None`, optional) – The maximum number of seconds to wait for the connection to be established.

`get_descriptors()`

Descriptors were added in HiSLIP version 2.0 to provide extra information about specific server capabilities.

Returns

`GetDescriptorsResponse` – The response.

`property maximum_server_message_size`

The maximum message size that the server accepts.

Type

`int`

`read(message, chunk_size=4096)`

Read a message from the server.

Parameters

- **message** (`Message`) – An instance of the type of message to read.
- **chunk_size** (`int`, optional) – The maximum number of bytes to receive at a time.

Returns

`Message` – The `message` that was passed in, but with its attributes updated with the information from the received data.

`get_timeout()`

Get the socket timeout value.

Returns

`float` or `None` – The timeout, in seconds, of the socket.

`set_timeout(timeout)`

Set the socket timeout value.

Parameters

`timeout` (`float` or `None`) – The timeout, in seconds, to use for the socket.

`property socket`

The reference to the socket.

Type

`socket`

`write(message)`

Write a message to the server.

Parameters

`message` (`Message`) – The message to write.

`class msl.equipment.hislip.SyncClient(host)`

Bases: `HiSLIPClient`

A synchronous connection to the HiSLIP server.

Parameters

host (`str`) – The hostname or IP address of the remote device.

device_clear_complete(*feature_bitmap*)

Send the device-clear complete message.

Also resets the message id.

Parameters

feature_bitmap (`int`) – The feature bitmap of the server (i.e., `AsyncDeviceClearAcknowledge.feature_bitmap`).

Returns

`DeviceClearAcknowledge` – The response.

initialize(*major*=1, *minor*=0, *client_id*=b'XX', *sub_address*=b'')

Initialize the synchronous connection.

Parameters

- **major** (`int`, optional) – The major version number of the HiSLIP protocol that the client supports.
- **minor** (`int`, optional) – The minor version number of the HiSLIP protocol that the client supports.
- **client_id** (`bytes`, optional) – The vendor ID of the client. Must have a length of 2 characters.
- **sub_address** (`bytes`, optional) – A particular device managed by this server. For VISA clients this field corresponds to the VISA LAN device name (default is `hislip0`). The maximum length is 256 characters.

Returns

`InitializeResponse` – The response.

property message_id

The id of the most-recent message that has completed.

Type

`int`

property message_id_received

The id of most-recent message that has been received from the server.

Type

`int`

receive(*size*=None, *max_size*=None, *chunk_size*=4096)

Receive data.

Parameters

- **size** (`int`, optional) – The number of bytes to read. If not specified, then read until a Response Message Terminator (RMT) is detected.
- **max_size** (`int`, optional) – The maximum number of bytes that can be read. If not specified, then there is no limit.

- **chunk_size** (`int`, optional) – The maximum number of bytes to receive at a time.

Returns

`bytearray` – The received data.

property rmt

`int` The current state of the Response Message Terminator (RMT).

send(data)

Send data with the Response Message Terminator (RMT) character.

Parameters

`data (bytes)` – The data to send.

Returns

`int` – The number of bytes sent.

trigger()

Send the trigger message (emulates a GPIB Group Execute Trigger event).

start_tls()

Send the *StartTLS* message.

end_tls()

Send the *EndTLS* message.

get_sasl_mechanism_list()

Request the list of SASL mechanisms from the server.

Returns

`GetSaslMechanismListResponse` – The response.

authentication_start(mechanism)

Send a SASL authentication method to the server.

Parameters

`mechanism (bytes)` – The selected mechanism to use for authentication.

write_authentication_exchange(data)

Send exchange data during the authentication transaction.

Parameters

`data (bytes)` – The data to send.

read_authentication_exchange()

Receive exchange data during the authentication transaction.

Returns

`AuthenticationExchange` – The exchange.

authentication_result()

Receive an authentication result from the server.

Returns

`AuthenticationResult` – The result.

```
class msl.equipment.hislip.AsyncClient(host)
```

Bases: [HiSLIPClient](#)

An asynchronous connection to the HiSLIP server.

Parameters

host ([str](#)) – The hostname or IP address of the remote device.

```
async_initialize(session_id)
```

Initialize the asynchronous connection.

Parameters

session_id ([int](#)) – The session ID.

Returns

[AsyncInitializeResponse](#) – The response.

```
async_maximum_message_size(size)
```

Exchange the maximum message sizes that are accepted between the client and server.

Parameters

size ([int](#)) – The maximum message size that the client accepts.

Returns

[AsyncMaximumMessageSizeResponse](#) – The maximum message size that the server accepts.

```
async_lock_request(timeout=None, lock_string="")
```

Request a lock.

Parameters

- **timeout** ([float](#), optional) – The number of seconds to wait to acquire a lock. A timeout of 0 indicates that the HiSLIP server should only grant the lock if it is available immediately.
- **lock_string** ([str](#), optional) – An ASCII string that identifies this lock. If not specified, then an exclusive lock is requested, otherwise the string indicates an identification of a shared-lock request. The maximum length is 256 characters.

Returns

[AsyncLockResponse](#) – The response.

```
async_lock_release(message_id)
```

Release a lock.

Parameters

message_id ([int](#)) – The most recent message id that was completed on the synchronous channel (i.e., [SyncClient.message_id](#)).

Returns

[AsyncLockResponse](#) – The response.

```
async_lock_info()
```

Request the lock status from the HiSLIP server.

Returns

[AsyncLockInfoResponse](#) – The response.

async_remote_local_control(*request, message_id*)

Send a GPIB-like remote/local control request.

Parameters

- **request** ([int](#)) – The request to perform.
 - 0 – Disable remote, *VI_GPIB_REN_DEASSERT*
 - 1 – Enable remote, *VI_GPIB_REN_ASSERT*
 - 2 – Disable remote and go to local, *VI_GPIB_REN_DEASSERT_GTL*
 - 3 – Enable Remote and go to remote,
VI_GPIB_REN_ASSERT_ADDRESS
 - 4 – Enable remote and lock out local, *VI_GPIB_REN_ASSERT_LLO*
 - 5 – Enable remote, go to remote, and set local lockout,
VI_GPIB_REN_ASSERT_ADDRESS_LLO
 - 6 – go to local without changing REN or lockout state,
VI_GPIB_REN_ADDRESS_GTL
- **message_id** ([int](#)) – The most recent message id that was completed on the synchronous channel (i.e., *SyncClient.message_id*).

Returns

AsyncRemoteLocalResponse – The response.

async_device_clear()

Send the device clear request.

Returns

AsyncDeviceClearAcknowledge – The response.

async_status_query(*synchronous*)

Status query transaction.

The status query provides an 8-bit status response from the server that corresponds to the VISA *viReadSTB* operation.

Parameters

synchronous (*SyncClient*) – The synchronous client that corresponds with this asynchronous client.

Returns

AsyncStatusResponse – The response.

async_start_tls(*synchronous*)

Initiate the secure connection transaction.

Parameters

synchronous (*SyncClient*) – The synchronous client that corresponds with this asynchronous client.

Returns

AsyncStartTLSResponse – The response.

async_end_tls(*synchronous*)

Initiate the end of the secure connection transaction.

Parameters

synchronous (*SyncClient*) – The synchronous client that corresponds with this asynchronous client.

Returns

AsyncEndTLSResponse – The response.

msl.equipment.record_types module

Records from *Equipment-Register Database*'s or *Connections Database*'s.

```
class msl.equipment.record_types.EquipmentRecord(alias='', calibrations=None,  
                                              category='', connection=None,  
                                              description='', is_operable=False,  
                                              maintenances=None, manufacturer='',  
                                              model='', serial='', team='',  
                                              unique_key='', **user_defined)
```

Bases: *Record*

Contains the information about an equipment record in an *Equipment-Register Database*.

Parameters

- **alias** (*str*) – An alias to use to reference this equipment by.
- **calibrations** (*list* of *CalibrationRecord*) – The calibration history of the equipment.
- **category** (*str*) – The category (e.g., Laser, DMM) that the equipment belongs to.
- **connection** (*ConnectionRecord*) – The information necessary to communicate with the equipment.
- **description** (*str*) – A description about the equipment.
- **is_operable** (*bool*) – Whether the equipment is able to be used.
- **maintenances** (*list* of *MaintenanceRecord*) – The maintenance history of the equipment.
- **manufacturer** (*str*) – The name of the manufacturer of the equipment.
- **model** (*str*) – The model number of the equipment.
- **serial** (*str*) – The serial number (or unique identifier) of the equipment.
- **team** (*str*) – The team (e.g., Light Standards) that the equipment belongs to.
- **unique_key** (*str*) – The key that uniquely identifies the equipment record in a database.
- ****user_defined** – All additional key-value pairs are added to the *user_defined* attribute.

alias

An alias to use to reference this equipment by.

The *alias* can be defined in 4 ways:

- by specifying it when the EquipmentRecord is created
- by setting the value after the EquipmentRecord has been created
- in the `<equipment>` XML tag in a *Configuration File*
- in the **Properties** field in a *Connections Database*

Type

`str`

calibrations

The calibration history of the equipment.

Type

`tuple of CalibrationRecord`

category

The category (e.g., Laser, DMM) that the equipment belongs to.

Type

`str`

description

A description about the equipment.

Type

`str`

is_operable

Whether the equipment is able to be used.

Type

`bool`

maintenances

The maintenance history of the equipment.

Type

`tuple of MaintenanceRecord`

manufacturer

The name of the manufacturer of the equipment.

Type

`str`

model

The model number of the equipment.

Type

`str`

serial

The serial number (or unique identifier) of the equipment.

Type

`str`

connection

The information necessary to communicate with the equipment.

Type

`ConnectionRecord`

team

The team (e.g., Light Standards) that the equipment belongs to.

Type

`str`

unique_key

The key that uniquely identifies the equipment record in a database.

Type

`str`

user_defined

User-defined, key-value pairs.

Type

`RecordDict`

connect(`demo=None`)

Establish a connection to the equipment.

Calls the `connect()` function.

Parameters

`demo (bool, optional)` – Whether to simulate a connection to the equipment by opening a connection in demo mode. This allows you to test your code if the equipment is not physically connected to a computer.

If `None` then the `demo` value is determined from the `DEMO_MODE` attribute.

Returns

A `Connection` subclass.

is_calibration_due(`months: int = 0`) → bool

Whether the equipment needs to be re-calibrated.

Parameters

`months` – The number of months to add to today's date to determine if the equipment needs to be re-calibrated within a certain amount of time. For example, if `months = 6` then that is a way of asking “*is a re-calibration due within the next 6 months?*”.

Returns

`True` if the equipment needs to be re-calibrated, `False` if it does not need to be re-calibrated (or it has never been calibrated).

property latest_calibration

The latest calibration or `None` if the equipment has never been calibrated.

Type

`CalibrationRecord`

next_calibration_date() → `date | None`

The next calibration date or `None` if the equipment has never been calibrated or if it is no longer in operation.

to_dict()

Convert this `EquipmentRecord` to a `dict`.

Returns

`dict` – The `EquipmentRecord` as a `dict`.

to_json()

Convert this `EquipmentRecord` to be `JSON` serializable.

Returns

`dict` – The `EquipmentRecord` as a `JSON`-serializable object.

to_xml()

Convert this `EquipmentRecord` to an XML `Element`.

Returns

`Element` – The `EquipmentRecord` as an XML element.

class `msl.equipment.record_types.CalibrationRecord(calibration_cycle=0,`
`calibration_date=None,`
`measurands=None,`
`report_date=None,`
`report_number="")`

Bases: `Record`

Contains the information about a calibration record in an `Equipment-Register Database`.

Parameters

- **calibration_cycle** (`int` or `float`) – The number of years that can pass before the equipment must be re-calibrated.
- **calibration_date** (`datetime.date`, `datetime.datetime` or `str`) – The date that the calibration was performed. If a `str` then in the format '`YYYY-MM-DD`'.
- **measurands** (`list` of `MeasurandRecord`) – The quantities that were measured.
- **report_date** (`datetime.date`, `datetime.datetime` or `str`) – The date that the report was issued. If a `str` then in the format '`YYYY-MM-DD`'.
- **report_number** (`str`) – The report number.

to_dict()

`dict`: Convert the Record to a `dict`.

calibration_cycle

The number of years that can pass before the equipment must be re-calibrated.

Type`float`**calibration_date**

The date that the calibration was performed.

Type`datetime.date`**measurands**

The quantities that were measured.

Type`RecordDict`**report_date**

The date that the report was issued.

Type`datetime.date`**report_number**

The report number.

Type`str`**to_json()**

Convert this `CalibrationRecord` to be JSON serializable.

Returns`dict` – The `CalibrationRecord` as a JSON-serializable object.**to_xml()**

Convert this `CalibrationRecord` to an XML Element.

Returns`Element` – The `CalibrationRecord` as a XML Element.

```
class msl.equipment.record_types.MeasurandRecord(calibration=None, conditions=None,  
                                                 type='', unit='')
```

Bases: `Record`

Contains the information about a measurement for a calibration.

Parameters

- **calibration** (`dict`) – The information about the calibration.
- **conditions** (`dict`) – The information about the conditions under which the measurement was performed.
- **type** (`str`) – The type of measurement (e.g., voltage, temperature, transmittance, ...).
- **unit** (`str`) – The unit that is associated with the measurement (e.g., V, deg C, %, ...).

to_dict()

`dict`: Convert the Record to a `dict`.

calibration

The information about calibration.

Type

`RecordDict`

conditions

The information about the measurement conditions.

Type

`RecordDict`

type

The type of measurement (e.g., voltage, temperature, transmittance, ...).

Type

`str`

unit

The unit that is associated with the measurement (e.g., V, deg C, %, ...).

Type

`str`

to_json()

Convert this `MeasurandRecord` to be `JSON` serializable.

Returns

`dict` – The `MeasurandRecord` as a `JSON`-serializable object.

to_xml()

Convert this `MeasurandRecord` to an XML `Element`.

Returns

`Element` – The `MeasurandRecord` as a XML `Element`.

class `msl.equipment.record_types.MaintenanceRecord(comment='', date=None)`

Bases: `Record`

Contains the information about a maintenance record in an *Equipment-Register Database*.

Parameters

- `comment` (`str`) – A description of the maintenance that was performed.
- `date` (`datetime.date`, `datetime.datetime` or `str`) – An object that can be converted to a `datetime.date` object. If a `str` then in the format 'YYYY-MM-DD'.

comment

A description of the maintenance that was performed.

Type

`str`

date

The date that the maintenance was performed.

Type

`datetime.date`

to_json()

Convert this *MaintenanceRecord* to be JSON serializable.

Returns

`dict` – The *MaintenanceRecord* as a JSON-serializable object.

to_xml()

Convert this *MaintenanceRecord* to an XML Element.

Returns

`Element` – The *MaintenanceRecord* as a XML Element.

to_dict()

`dict`: Convert the Record to a `dict`.

```
class msl.equipment.record_types.ConnectionRecord(address='', backend=Backend.MSL,
                                                interface=None, manufacturer='',
                                                model='', serial='', **properties)
```

Bases: *Record*

Contains the information about a connection record in a *Connections Database*.

Parameters

- **address** (`str`) – The address to use for the connection (see *Address Syntax* for examples).
- **backend** (`str`, `int`, or `Backend`) – The backend to use to communicate with the equipment. The value must be able to be converted to a `Backend` enum.
- **interface** (`str`, `int`, or `Interface`) – The interface to use to communicate with the equipment. If `None` then determines the *interface* based on the value of *address*. If specified then the value must be able to be converted to a `Interface` enum.
- **manufacturer** (`str`) – The name of the manufacturer of the equipment.
- **model** (`str`) – The model number of the equipment.
- **serial** (`str`) – The serial number (or unique identifier) of the equipment.
- **properties** – Additional key-value pairs that are required to communicate with the equipment.

address

The address to use for the connection (see *Address Syntax* for examples).

Type

`str`

backend

The backend to use to communicate with the equipment.

Type
Backend

interface

The interface that is used for the communication system that transfers data between a computer and the equipment (only used if the *backend* is equal to *MSL*).

Type
Interface

manufacturer

The name of the manufacturer of the equipment.

Type
str

model

The model number of the equipment.

Type
str

properties

Additional key-value pairs that are required to communicate with the equipment.

For example, communicating via RS-232 may require:

```
{'baud_rate': 19200, 'parity': 'even'}
```

See the *Connections Database* for examples on how to set the *properties*.

Type
dict

serial

The serial number (or unique identifier) of the equipment.

Type
str

to_json()

Convert this *ConnectionRecord* to be *JSON* serializable.

Returns

dict – The *ConnectionRecord* as a *JSON*-serializable object.

to_xml()

Convert this *ConnectionRecord* to an XML *Element*.

Returns

Element – The *ConnectionRecord* as a XML *Element*.

to_dict()

dict: Convert the Record to a *dict*.

class msl.equipment.record_types.**RecordDict**(*dictionary*)

Bases: *Mapping*

A read-only dictionary that supports attribute access via a key lookup.

copy()

RecordDict: Return a copy of the *RecordDict*.

to_xml(tag='RecordDict')

Convert the *RecordDict* to an XML *Element*

Parameters

tag (*str*) – The name of the *Element*.

Returns

Element – The *RecordDict* as an XML *Element*.

to_json()

dict: Convert the *RecordDict* to be JSON serializable.

class msl.equipment.record_types.Record

Bases: *object*

to_dict()

dict: Convert the Record to a *dict*.

to_json()

dict: Convert the Record to be JSON serializable.

This differs from *to_dict()* such that all values that are not JSON serializable, like *datetime.date* objects, are converted to a *str*.

to_xml()

Element: Convert the Record to an XML *Element*.

msl.equipment.resources package

MSL resources for connecting to equipment.

msl.equipment.resources.register(manufacturer=None, model=None, flags=0)

Use as a decorator to register a resource class.

Parameters

- **manufacturer** (*str*, optional) – The name of the manufacturer. Can be a regex pattern.
- **model** (*str*, optional) – The model number of the equipment. Can be a regex pattern.
- **flags** (*int*, optional) – The flags to use for the regex pattern.

msl.equipment.resources.find_resource_class(record)

Find the resource class for this *record*.

Parameters

record (*EquipmentRecord* or *ConnectionRecord*) – A record type. If the *properties* attribute contains a *resource_class_name* key with a value that is equal to the name of a resource class it forces this resource class to be returned, provided that a resource class with the requested name exists.

Returns

The *Connection* subclass or *None* if a resource class cannot be found.

Submodules

`msl.equipment.resources.utils module`

Utility functions/classes to help create modules in the `msl.equipment.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(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 comments.

Parameters

- `path (str)` – The path to a C/C++ header file.
- `remove_comments (bool, optional)` – Whether to remove the comments.

Returns

`list` of `str` – The list of lines in the header file.

`class msl.equipment.resources.utils.CHeader(path, remove_comments=True)`

Bases: `object`

Parses a C/C++ header file to determine the constants, enums, structs, callbacks and the function signatures.

Parameters

- `path (str)` – The path to the header file.
- `remove_comments (bool, optional)` – Whether to remove the comments.

`constants(ignore_ifdef=True)`

Finds the `#define` statements that are in the C/C++ header file.

Parameters

`ignore_ifdef (bool, optional)` – Whether to ignore the `#define` statements in between the `#ifdef` and `#endif` statements.

Returns

`dict` – 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 enum name, the enum data type, 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 contain 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, see `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*.

Subpackages

`msl.equipment.resources.aim_tti package`

Resources for equipment from Aim and Thurlby Thandar Instruments.

Submodules

`msl.equipment.resources.aim_tti.mx_series module`

Establishes a connection to an MX100QP, MX100TP or MX180TP DC power supply from [Aim](#) and [Thurlby Thandar Instruments](#)

`class msl.equipment.resources.aim_tti.mx_series.MXSeries(record)`

Bases: `object`

Establishes a connection to an MX100QP, MX100TP or MX180TP DC power supply from [Aim](#) and [Thurlby Thandar Instruments](#) for different interfaces:

- `Interface.PROLOGIX`
- `Interface.SERIAL`
- `Interface.SOCKET`

Do not instantiate this class directly. Use the `connect()` method to connect to the equipment.

Parameters

`record` ([EquipmentRecord](#)) – A record from an [Equipment-Register Database](#).

`clear()`

Send the clear, *CLS, command.

`decrement_current(channel)`

Decrement the current limit by step size of the output channel.

Parameters

`channel` (`int`) – The output channel. The first output channel is 1 (not 0).

See also:

`set_current_step_size()`

`decrement_voltage(channel, verify=True)`

Decrement the voltage by step size of the output channel.

Parameters

- `channel` (`int`) – The output channel. The first output channel is 1 (not 0).

- **verify** (`bool`, optional) – Whether to verify that the output voltage has stabilized at the decremented voltage before returning to the calling program.

See also:

`set_voltage_step_size()`

`event_status_register(as_integer=True)`

Read and clear the standard event status register.

Parameters

`as_integer` (`bool`, optional) – Whether to return the value as an `int`.

Returns

`int` or `str` – The event status register value. The data type depends on the value of `as_integer`. If a `str` is returned then it will have a length of 8. For example,

- '10000000' or the integer value 128
- '00100000' or the integer value 32

`get_current(channel)`

Get the output current of the output channel.

Parameters

`channel` (`int`) – The output channel. The first output channel is 1 (not 0).

Returns

`float` – The output current, in Amps.

`get_current_limit(channel)`

Get the current limit of the output channel.

Parameters

`channel` (`int`) – The output channel. The first output channel is 1 (not 0).

Returns

`float` – The current limit, in Amps.

`get_current_step_size(channel)`

Get the current limit step size of the output channel.

Parameters

`channel` (`int`) – The output channel. The first output channel is 1 (not 0).

Returns

`float` – The current limit step size, in Amps.

`get_over_current_protection(channel)`

Get the over-current protection trip point of the output channel.

Parameters

`channel` (`int`) – The output channel. The first output channel is 1 (not 0).

Returns

`float` or `None` – If the trip point is enabled then returns the trip point value, in Amps. Returns `None` if the over-current protection is disabled.

get_over_voltage_protection(*channel*)

Get the over-voltage protection trip point of the output channel.

Parameters

channel (`int`) – The output channel. The first output channel is 1 (not 0).

Returns

`float` or `None` – If the trip point is enabled then returns the trip point value, in Volts. Returns `None` if the over-voltage protection is disabled.

get_voltage(*channel*)

Get the output voltage of the output channel.

Parameters

channel (`int`) – The output channel. The first output channel is 1 (not 0).

Returns

`float` – The output voltage, in Volts.

get_voltage_range(*channel*)

Get the output voltage range index of the output channel.

Parameters

channel (`int`) – The output channel. The first output channel is 1 (not 0).

Returns

`int` – The output voltage range index. See the manual for more details. For example, 2 = 35V/3A.

get_voltage_setpoint(*channel*)

Get the set-point voltage of the output channel.

Parameters

channel (`int`) – The output channel. The first output channel is 1 (not 0).

Returns

`float` – The set-point voltage, in Volts.

get_voltage_step_size(*channel*)

Get the voltage step size of the output channel.

Parameters

channel (`int`) – The output channel. The first output channel is 1 (not 0).

Returns

`float` – The voltage step size, in Volts.

get_voltage_tracking_mode()

Get the voltage tracking mode of the unit.

Returns

`int` – The voltage tracking mode. See the manual for more details.

increment_current(*channel*)

Increment the current limit by step size of the output channel.

Parameters

channel (`int`) – The output channel. The first output channel is 1 (not 0).

See also:

`set_current_step_size()`

`increment_voltage(channel, verify=True)`

Increment the voltage by step size of the output channel.

Parameters

- **channel** (`int`) – The output channel. The first output channel is 1 (not 0).
- **verify** (`bool`, optional) – Whether to verify that the output voltage has stabilized at the incremented voltage before returning to the calling program.

See also:

`set_voltage_step_size()`

`is_output_on(channel)`

Check if the output channel is on or off.

Parameters

`channel` (`int`) – The output channel. The first output channel is 1 (not 0).

Returns

`bool` – Whether the output channel is on (`True`) or off (`False`).

`turn_on(channel)`

Turn the output channel on.

Parameters

`channel` (`int`) – The output channel. The first output channel is 1 (not 0).

`turn_on_multi(options=None)`

Turn multiple output channels on (the Multi-On feature).

Parameters

`options` (`dict`, optional) – Set the Multi-On option for each output channel before setting Multi-On. If not specified then uses the pre-programmed options. If a particular output channel is not included in `options` then uses the pre-programmed option for that channel. The keys are the output channel number and the value can be `False` (set the channel to NEVER, see the manual for more details), `True` (set the channel to QUICK, see the manual for more details) or a delay in milliseconds (as an `int`).

Examples:

- `{1: False}` → channel 1 does not turn on
- `{2: 100}` → channel 2 has a 100-ms delay
- `{1: 100, 3: True}` → channel 1 has a 100-ms delay and channel 3 turns on immediately
- `{1: 100, 2: 200, 3: 300}` → channel 1 has a 100-ms delay, channel 2 has a 200-ms delay and channel 3 has a 300-ms delay

See also:

`set_multi_on_delay()`, `set_multi_on_action()`

turn_off(*channel*)

Turn the output channel off.

Parameters

channel (`int`) – The output channel. The first output channel is 1 (not 0).

turn_off_multi(*options=None*)

Turn multiple output channels off (the Multi-Off feature).

Parameters

options (`dict`, optional) – Set the Multi-Off option for each output channel before setting Multi-Off. If not specified then uses the pre-programmed options. If a particular output channel is not included in *options* then uses the pre-programmed option for that channel. The keys are the output channel number and the value can be `False` (set the channel to NEVER, see the manual for more details), `True` (set the channel to QUICK, see the manual for more details) or a delay in milliseconds (as an `int`).

Examples:

- {1: `False`} → channel 1 does not turn off
- {2: 100} → channel 2 has a 100-ms delay
- {1: 100, 3: `True`} → channel 1 has a 100-ms delay and channel 3 turns off immediately
- {1: 100, 2: 200, 3: 300} → channel 1 has a 100-ms delay, channel 2 has a 200-ms delay and channel 3 has a 300-ms delay

See also:

`set_multi_off_delay()`, `set_multi_off_action()`

recall(*channel, index*)

Recall the settings of the output channel from the store.

Parameters

- **channel** (`int`) – The output channel. The first output channel is 1 (not 0).
- **index** (`int`) – The store index number, can be 0-49.

See also:

`save()`

recall_all(*index*)

Recall the settings for all output channels from the store.

Parameters

index (`int`) – The store index number, can be 0-49.

See also:

`save_all()`

reset()

Send the reset, *RST, command.

reset_trip()

Attempt to clear all trip conditions.

save(channel, index)

Save the present settings of the output channel to the store.

Parameters

- **channel** ([int](#)) – The output channel. The first output channel is 1 (not 0).
- **index** ([int](#)) – The store index number, can be 0-49.

See also:

[*recall\(\)*](#)**save_all(index)**

Save the settings of all output channels to the store.

Parameters

- **index** ([int](#)) – The store index number, can be 0-49.

See also:

[*recall_all\(\)*](#)**set_current_limit(channel, value)**

Set the current limit of the output channel.

Parameters

- **channel** ([int](#)) – The output channel. The first output channel is 1 (not 0).
- **value** ([float](#)) – The current limit, in Amps.

set_current_meter_averaging(channel, value)

Set the current meter measurement averaging of the output channel.

Parameters

- **channel** ([int](#)) – The output channel. The first output channel is 1 (not 0).
- **value** ([str](#)) – Can be ON, OFF, LOW, MED or HIGH.

set_current_step_size(channel, size)

Set the current limit step size of the output channel.

Parameters

- **channel** ([int](#)) – The output channel. The first output channel is 1 (not 0).
- **size** ([float](#)) – The current limit step size, in Amps.

set_multi_on_action(channel, action)

Set the Multi-On action of the output channel.

Parameters

- **channel** ([int](#)) – The output channel. The first output channel is 1 (not 0).
- **action** ([str](#)) – The Multi-On action, one of QUICK, NEVER or DELAY.

set_multi_on_delay(channel, delay)

Set the Multi-On delay, in milliseconds, of the output channel.

Parameters

- **channel** (`int`) – The output channel. The first output channel is 1 (not 0).
- **delay** (`int`) – The delay, in milliseconds.

set_multi_off_action(channel, action)

Set the Multi-Off action of the output channel.

Parameters

- **channel** (`int`) – The output channel. The first output channel is 1 (not 0).
- **action** (`str`) – The Multi-Off action, one of QUICK, NEVER or DELAY.

set_multi_off_delay(channel, delay)

Set the Multi-Off delay, in milliseconds, of the output channel.

Parameters

- **channel** (`int`) – The output channel. The first output channel is 1 (not 0).
- **delay** (`int`) – The delay, in milliseconds.

set_over_current_protection(channel, enable, value=None)

Set the over-current protection trip point of the output channel.

Parameters

- **channel** (`int`) – The output channel. The first output channel is 1 (not 0).
- **enable** (`bool`) – Whether to enable (`True`) or disable (`False`) the over-current protection trip point.
- **value** (`float`, optional) – If the trip point is enabled then you must specify a value, in Amps.

set_over_voltage_protection(channel, enable, value=None)

Set the over-voltage protection trip point of the output channel.

Parameters

- **channel** (`int`) – The output channel. The first output channel is 1 (not 0).
- **enable** (`bool`) – Whether to enable (`True`) or disable (`False`) the over-voltage protection trip point.
- **value** (`float`, optional) – If the trip point is enabled then you must specify a value, in Volts.

set_voltage(channel, value, verify=True)

Set the output voltage of the output channel.

Parameters

- **channel** (`int`) – The output channel. The first output channel is 1 (not 0).
- **value** (`float`) – The value, in Volts.

- **verify** (`bool`, optional) – Whether to verify that the output voltage has stabilized at *value* before returning to the calling program.

set_voltage_range(*channel*, *index*)

Set the output voltage range of the output channel.

Parameters

- **channel** (`int`) – The output channel. The first output channel is 1 (not 0).
- **index** (`int`) – The output voltage range index. See the manual for more details. For example, 2 = 35V/3A.

set_voltage_step_size(*channel*, *size*)

Set the voltage step size of the output channel.

Parameters

- **channel** (`int`) – The output channel. The first output channel is 1 (not 0).
- **size** (`float`) – The voltage step size, in Volts.

set_voltage_tracking_mode(*mode*)

Set the voltage tracking mode of the unit.

Parameters

mode (`int`) – The voltage tracking mode. See the manual for more details.

msl.equipment.resources.avantes package

Wrapper around the avaspec.dll SDK from Avantes.

Submodules

msl.equipment.resources.avantes.avaspec module

Wrapper around the avaspec.dll SDK from Avantes.

The wrapper was written using v9.7.0.0 of the SDK.

```
class msl.equipment.resources.avantes.avaspec.DeviceStatus(value, names=None,
                                                               *values, module=None,
                                                               qualname=None,
                                                               type=None, start=1,
                                                               boundary=None)
```

Bases: `IntEnum`

DeviceStatus enum.

UNKNOWN = 0

USB_AVAILABLE = 1

USB_IN_USE_BY_APPLICATION = 2

USB_IN_USE_BY_OTHER = 3

ETH_AVAILABLE = 4

ETH_IN_USE_BY_APPLICATION = 5

ETH_IN_USE_BY_OTHER = 6

ETH_ALREADY_IN_USE_USB = 7

```
class msl.equipment.resources.avantes.avaspec.InterfaceType(value, names=None,
                                                               *values, module=None,
                                                               qualname=None,
                                                               type=None, start=1,
                                                               boundary=None)
```

Bases: [IntEnum](#)

InterfaceType enum.

RS232 = 0

USB5216 = 1

USBMINI = 2

USB7010 = 3

ETH7010 = 4

```
class msl.equipment.resources.avantes.avaspec.SensType(value, names=None, *values,
                                                       module=None,
                                                       qualname=None, type=None,
                                                       start=1, boundary=None)
```

Bases: [IntEnum](#)

SensType enum.

SENS_HAMS8378_256 = 1

SENS_HAMS8378_1024 = 2

SENS_ILX554 = 3

SENS_HAMS9201 = 4

SENS_TCD1304 = 5

SENS_TSL1301 = 6

SENS_TSL1401 = 7

SENS_HAMS8378_512 = 8

SENS_HAMS9840 = 9

SENS_ILX511 = 10

SENS_HAMS10420_2048X64 = 11

```
SENS_HAMS11071_2048X64 = 12
SENS_HAMS7031_1024X122 = 13
SENS_HAMS7031_1024X58 = 14
SENS_HAMS11071_2048X16 = 15
SENS_HAMS11155_2048 = 16
SENS_SU256LSB = 17
SENS_SU512LDB = 18
SENS_HAMS11638 = 21
SENS_HAMS11639 = 22
SENS_HAMS12443 = 23
SENS_HAMG9208_512 = 24
SENS_HAMG13913 = 25
SENS_HAMS13496 = 26
```

class msl.equipment.resources.avantes.avaspec.AvsIdentityType

Bases: Structure

IdentityType Structure.

SerialNumber

Structure/Union member

Status

Structure/Union member

UserFriendlyName

Structure/Union member

class msl.equipment.resources.avantes.avaspec.BroadcastAnswerType

Bases: Structure

BroadcastAnswerType Structure.

InterfaceType

Structure/Union member

LocalIp

Structure/Union member

RemoteHostIp

Structure/Union member

port

Structure/Union member

reserved

Structure/Union member

serial

Structure/Union member

status

Structure/Union member

class msl.equipment.resources.avantes.avaspec.**ControlSettingsType**

Bases: Structure

ControlSettingsType Structure.

m_LaserDelay

Structure/Union member

m_LaserWaveLength

Structure/Union member

m_LaserWidth

Structure/Union member

m_StoreToRam

Structure/Union member

m_StrobeControl

Structure/Union member

class msl.equipment.resources.avantes.avaspec.**DarkCorrectionType**

Bases: Structure

DarkCorrectionType Structure.

m_Enable

Structure/Union member

m_ForgetPercentage

Structure/Union member

class msl.equipment.resources.avantes.avaspec.**DetectorType**

Bases: Structure

DetectorType Structure.

m_DefectivePixels

Structure/Union member

m_ExtOffset

Structure/Union member

m_Gain

Structure/Union member

m_NLEnable

Structure/Union member

m_NrPixels

Structure/Union member

m_Offset

Structure/Union member

m_Reserved

Structure/Union member

m_SensorType

Structure/Union member

m_aFit

Structure/Union member

m_aHighNLCounts

Structure/Union member

m_aLowNLCounts

Structure/Union member

m_aNLCorrect

Structure/Union member

class msl.equipment.resources.avantes.avaspec.SmoothingType

Bases: Structure

SmoothingType Structure.

m_SmoothModel

Structure/Union member

m_SmoothPix

Structure/Union member

class msl.equipment.resources.avantes.avaspec.SpectrumCalibrationType

Bases: Structure

SpectrumCalibrationType Structure.

m_CalInttime

Structure/Union member

m_Smoothing

Structure/Union member

m_aCalibConvers

Structure/Union member

class msl.equipment.resources.avantes.avaspec.IrradianceType

Bases: Structure

IrradianceType Structure.

m_CalibrationType

Structure/Union member

m_FiberDiameter

Structure/Union member

m_IntensityCalib

Structure/Union member

class msl.equipment.resources.avantes.avaspec.SpectrumCorrectionType

Bases: Structure

SpectrumCorrectionType Structure.

m_aSpectrumCorrect

Structure/Union member

class msl.equipment.resources.avantes.avaspec.TriggerType

Bases: Structure

TriggerType Structure.

m_Mode

Structure/Union member

m_Source

Structure/Union member

m_SourceType

Structure/Union member

class msl.equipment.resources.avantes.avaspec.MeasConfigType

Bases: Structure

MeasConfigType Structure.

m_Control

Structure/Union member

m_CorDynDark

Structure/Union member

m_IntegrationDelay

Structure/Union member

m_IntegrationTime

Structure/Union member

m_NrAverages

Structure/Union member

m_SaturationDetection

Structure/Union member

m_Smoothing

Structure/Union member

m_StartPixel

Structure/Union member

m_StopPixel
Structure/Union member

m_Trigger
Structure/Union member

class msl.equipment.resources.avantes.avaspec.TimeStampType
Bases: Structure
TimeStampType Structure.

m_Date
Structure/Union member

m_Time
Structure/Union member

class msl.equipment.resources.avantes.avaspec.StandAloneType
Bases: Structure
StandAloneType Structure.

m_Enable
Structure/Union member

m_Meas
Structure/Union member

m_Nmsr
Structure/Union member

class msl.equipment.resources.avantes.avaspec.DynamicStorageType
Bases: Structure
DynamicStorageType Structure.

m_Nmsr
Structure/Union member

m_Reserved
Structure/Union member

class msl.equipment.resources.avantes.avaspec.TempSensorType
Bases: Structure
TempSensorType Structure.

m_aFit
Structure/Union member

class msl.equipment.resources.avantes.avaspec.TecControlType
Bases: Structure
TecControlType Structure.

m_Enable
Structure/Union member

m_Setpoint
Structure/Union member

m_aFit
Structure/Union member

class msl.equipment.resources.avantes.avaspec.ProcessControlType
Bases: Structure
ProcessControlType Structure.

m_AnalogHigh
Structure/Union member

m_AnalogLow
Structure/Union member

m_DigitalHigh
Structure/Union member

m_DigitalLow
Structure/Union member

class msl.equipment.resources.avantes.avaspec.EthernetSettingsType
Bases: Structure
EthernetSettingsType Structure.

m_DhcpEnabled
Structure/Union member

m_Gateway
Structure/Union member

m_IpAddr
Structure/Union member

m_LinkStatus
Structure/Union member

m_NetMask
Structure/Union member

m_TcpPort
Structure/Union member

class msl.equipment.resources.avantes.avaspec.OemDataType
Bases: Structure
OemDataType Structure.

m_data
Structure/Union member

class msl.equipment.resources.avantes.avaspec.HeartbeatRespType
Bases: Structure
HeartbeatRespType Structure.

m_BitMatrix
Structure/Union member

m_Reserved
Structure/Union member

class msl.equipment.resources.avantes.avaspec.DeviceConfigType
Bases: Structure
DeviceConfigType Structure.

m_ConfigVersion
Structure/Union member

m_Detector
Structure/Union member

m_DynamicStorage
Structure/Union member

m_EthernetSettings
Structure/Union member

m_Irradiance
Structure/Union member

m_Len
Structure/Union member

m_OemData
Structure/Union member

m_ProcessControl
Structure/Union member

m_Reflectance
Structure/Union member

m_SpectrumCorrect
Structure/Union member

m_Standalone
Structure/Union member

m_TecControl
Structure/Union member

m_aReserved
Structure/Union member

m_aTemperature
Structure/Union member

m_aUserFriendlyId
Structure/Union member

`msl.equipment.resources.avantes.avaspec.MeasureCallback`

Used as a decorator for a callback function when a scan is available.

`class msl.equipment.resources.avantes.avaspec.Avantes(record)`

Bases: `ConnectionSDK`

Wrapper around the `avaspec.dll` SDK from Avantes.

The `properties` for an Avantes connection supports the following key-value pairs in the `Connections Database`:

```
'port_id': int, One of -1 (Ethernet+USB), 0 (USB) or 256 (Ethernet)  
↳[default: -1]  
'activate': bool, Whether to automatically activate the connection  
↳[default: True]
```

Do not instantiate this class directly. Use the `connect()` method to connect to the equipment.

Parameters

`record (EquipmentRecord)` – A record from an `Equipment-Register Database`.

`WM_MEAS_READY = 32769`

`SETTINGS_RESERVED_LEN = 9720`

`INVALID_AVIS_HANDLE_VALUE = 1000`

`USER_ID_LEN = 64`

`AVS_SERIAL_LEN = 10`

`MAX_TEMP_SENSORS = 3`

`ROOT_NAME_LEN = 6`

`VERSION_LEN = 16`

`AVASPEC_ERROR_MSG_LEN = 8`

`AVASPEC_MIN_MSG_LEN = 6`

`OEM_DATA_LEN = 4096`

`NR_WAVELEN_POL_COEF = 5`

`NR_NONLIN_POL_COEF = 8`

`MAX_VIDEO_CHANNELS = 2`

`NR_DEFECTIVE_PIXELS = 30`

`MAX_NR_PIXELS = 4096`

`NR_TEMP_POL_COEF = 5`

`NR_DAC_POL_COEF = 2`

`SAT_PEAK_INVERSION = 2`

```
SW_TRIGGER_MODE = 0
HW_TRIGGER_MODE = 1
SS_TRIGGER_MODE = 2
EXTERNAL_TRIGGER = 0
SYNC_TRIGGER = 1
EDGE_TRIGGER_SOURCE = 0
LEVEL_TRIGGER_SOURCE = 1
ILX_FIRST_USED_DARK_PIXEL = 2
ILX_USED_DARK_PIXELS = 14
ILX_TOTAL_DARK_PIXELS = 18
TCD_FIRST_USED_DARK_PIXEL = 0
TCD_USED_DARK_PIXELS = 12
TCD_TOTAL_DARK_PIXELS = 13
HAMS9840_FIRST_USED_DARK_PIXEL = 0
HAMS9840_USED_DARK_PIXELS = 8
HAMS9840_TOTAL_DARK_PIXELS = 8
HAMS10420_FIRST_USED_DARK_PIXEL = 0
HAMS10420_USED_DARK_PIXELS = 4
HAMS10420_TOTAL_DARK_PIXELS = 4
HAMS11071_FIRST_USED_DARK_PIXEL = 0
HAMS11071_USED_DARK_PIXELS = 4
HAMS11071_TOTAL_DARK_PIXELS = 4
HAMS7031_FIRST_USED_DARK_PIXEL = 0
HAMS7031_USED_DARK_PIXELS = 4
HAMS7031_TOTAL_DARK_PIXELS = 4
HAMS11155_TOTAL_DARK_PIXELS = 20
MIN_ILX_INTTIME = 1.1
MILLI_TO_MICRO = 1000
NR_DIGITAL_OUTPUTS = 13
NR_DIGITAL_INPUTS = 13
```

```
NTC1_ID = 0
NTC2_ID = 1
TEC_ID = 2
NR_ANALOG_OUTPUTS = 2
ETH_CONN_STATUS_CONNECTING = 0
ETH_CONN_STATUS_CONNECTED = 1
ETH_CONN_STATUS_CONNECTED_NOMON = 2
ETH_CONN_STATUS_NOCONNECTION = 3
class DeviceStatus(value, names=None, *values, module=None, qualname=None,
                   type=None, start=1, boundary=None)
    Bases: IntEnum
    DeviceStatus enum.
    UNKNOWN = 0
    USB_AVAILABLE = 1
    USB_IN_USE_BY_APPLICATION = 2
    USB_IN_USE_BY_OTHER = 3
    ETH_AVAILABLE = 4
    ETH_IN_USE_BY_APPLICATION = 5
    ETH_IN_USE_BY_OTHER = 6
    ETH_ALREADY_IN_USE_USB = 7
class InterfaceType(value, names=None, *values, module=None, qualname=None,
                     type=None, start=1, boundary=None)
    Bases: IntEnum
    InterfaceType enum.
    RS232 = 0
    USB5216 = 1
    USBMINI = 2
    USB7010 = 3
    ETH7010 = 4
class SensType(value, names=None, *values, module=None, qualname=None, type=None,
               start=1, boundary=None)
    Bases: IntEnum
    SensType enum.
```

```
SENS_HAMS8378_256 = 1
SENS_HAMS8378_1024 = 2
SENS_ILX554 = 3
SENS_HAMS9201 = 4
SENS_TCD1304 = 5
SENS_TSL1301 = 6
SENS_TSL1401 = 7
SENS_HAMS8378_512 = 8
SENS_HAMS9840 = 9
SENS_ILX511 = 10
SENS_HAMS10420_2048X64 = 11
SENS_HAMS11071_2048X64 = 12
SENS_HAMS7031_1024X122 = 13
SENS_HAMS7031_1024X58 = 14
SENS_HAMS11071_2048X16 = 15
SENS_HAMS11155_2048 = 16
SENS_SU256LSB = 17
SENS_SU512LDB = 18
SENS_HAMS11638 = 21
SENS_HAMS11639 = 22
SENS_HAMS12443 = 23
SENS_HAMG9208_512 = 24
SENS_HAMG13913 = 25
SENS_HAMS13496 = 26

class AvsIdentityType
    Bases: Structure
    IdentityType Structure.

SerialNumber
    Structure/Union member

Status
    Structure/Union member
```

UserFriendlyName
Structure/Union member

class BroadcastAnswerType
Bases: Structure
BroadcastAnswerType Structure.

InterfaceType
Structure/Union member

LocalIp
Structure/Union member

RemoteHostIp
Structure/Union member

port
Structure/Union member

reserved
Structure/Union member

serial
Structure/Union member

status
Structure/Union member

class ControlSettingsType
Bases: Structure
ControlSettingsType Structure.

m_LaserDelay
Structure/Union member

m_LaserWaveLength
Structure/Union member

m_LaserWidth
Structure/Union member

m_StoreToRam
Structure/Union member

m_StrobeControl
Structure/Union member

class DarkCorrectionType
Bases: Structure
DarkCorrectionType Structure.

m_Enable
Structure/Union member

m_ForgetPercentage
Structure/Union member

class DetectorType
Bases: Structure
DetectorType Structure.

m_DefectivePixels
Structure/Union member

m_ExtOffset
Structure/Union member

m_Gain
Structure/Union member

m_NLEnable
Structure/Union member

m_NrPixels
Structure/Union member

m_Offset
Structure/Union member

m_Reserved
Structure/Union member

m_SensorType
Structure/Union member

m_aFit
Structure/Union member

m_aHighNLCounts
Structure/Union member

m_aLowNLCounts
Structure/Union member

m_aNLCorrect
Structure/Union member

class SmoothingType
Bases: Structure
SmoothingType Structure.

m_SmoothModel
Structure/Union member

m_SmoothPix
Structure/Union member

```
class SpectrumCalibrationType
    Bases: Structure
        SpectrumCalibrationType Structure.

    m_CalInttime
        Structure/Union member

    m_Smoothing
        Structure/Union member

    m_aCalibConvers
        Structure/Union member

class IrradianceType
    Bases: Structure
        IrradianceType Structure.

    m_CalibrationType
        Structure/Union member

    m_FiberDiameter
        Structure/Union member

    m_IntensityCalib
        Structure/Union member

class SpectrumCorrectionType
    Bases: Structure
        SpectrumCorrectionType Structure.

    m_aSpectrumCorrect
        Structure/Union member

class TriggerType
    Bases: Structure
        TriggerType Structure.

    m_Mode
        Structure/Union member

    m_Source
        Structure/Union member

    m_SourceType
        Structure/Union member

class MeasConfigType
    Bases: Structure
        MeasConfigType Structure.

    m_Control
        Structure/Union member
```

m_CorDynDark
Structure/Union member

m_IntegrationDelay
Structure/Union member

m_IntegrationTime
Structure/Union member

m_NrAverages
Structure/Union member

m_SaturationDetection
Structure/Union member

m_Smoothing
Structure/Union member

m_StartPixel
Structure/Union member

m_StopPixel
Structure/Union member

m_Trigger
Structure/Union member

class TimeStampType
Bases: Structure
TimeStampType Structure.

m_Date
Structure/Union member

m_Time
Structure/Union member

class StandAloneType
Bases: Structure
StandAloneType Structure.

m_Enable
Structure/Union member

m_Meas
Structure/Union member

m_Nmsr
Structure/Union member

class DynamicStorageType
Bases: Structure
DynamicStorageType Structure.

m_Nmsr
Structure/Union member

m_Reserved
Structure/Union member

class TempSensorType
Bases: Structure
TempSensorType Structure.

m_aFit
Structure/Union member

class TecControlType
Bases: Structure
TecControlType Structure.

m_Enable
Structure/Union member

m_Setpoint
Structure/Union member

m_aFit
Structure/Union member

class ProcessControlType
Bases: Structure
ProcessControlType Structure.

m_AnalogHigh
Structure/Union member

m_AnalogLow
Structure/Union member

m_DigitalHigh
Structure/Union member

m_DigitalLow
Structure/Union member

class EthernetSettingsType
Bases: Structure
EthernetSettingsType Structure.

m_DhcpEnabled
Structure/Union member

m_Gateway
Structure/Union member

m_IpAddr
Structure/Union member

m_LinkStatus
Structure/Union member

m_NetMask
Structure/Union member

m_TcpPort
Structure/Union member

class OemDataType
Bases: Structure
OemDataType Structure.

m_data
Structure/Union member

class HeartbeatResponseType
Bases: Structure
HeartbeatResponseType Structure.

m_BitMatrix
Structure/Union member

m_Reserved
Structure/Union member

class DeviceConfigType
Bases: Structure
DeviceConfigType Structure.

m_ConfigVersion
Structure/Union member

m_Detector
Structure/Union member

m_DynamicStorage
Structure/Union member

m_EthernetSettings
Structure/Union member

m_Irradiance
Structure/Union member

m_Len
Structure/Union member

m_OemData
Structure/Union member

m_ProcessControl
Structure/Union member

m_Reflectance
Structure/Union member

m_SpectrumCorrect
Structure/Union member

m_StandAlone
Structure/Union member

m_TecControl
Structure/Union member

m_aReserved
Structure/Union member

m_aTemperature
Structure/Union member

m_aUserFriendlyId
Structure/Union member

activate()
Activates the spectrometer for communication.
Raises
AvantesError – If there was an error.

deactivate()
Closes communication with the spectrometer.

get_analog_in(analog_id)
Get the status of the specified analog input.
Parameters
analog_id (`int`) – The identifier of the analog input to get.

- AS5216:
 - 0 = thermistor on optical bench (NIR 2.0 / NIR2.2 / NIR 2.5 / TEC)
 - 1 = 1V2
 - 2 = 5VIO
 - 3 = 5VUSB
 - 4 = AI2 = pin 18 at 26-pins connector
 - 5 = AI1 = pin 9 at 26-pins connector
 - 6 = NTC1 onboard thermistor
 - 7 = Not used
- Mini:
 - 0 = NTC1 onboard thermistor
 - 1 = Not used
 - 2 = Not used

- 3 = Not used
 - 4 = AI2 = pin 13 on micro HDMI = pin 11 on HDMI Terminal
 - 5 = AI1 = pin 16 on micro HDMI = pin 17 on HDMI Terminal
 - 6 = Not used
 - 7 = Not used
- AS7010:
 - 0 = thermistor on optical bench (NIR 2.0 / NIR2.2 / NIR 2.5 / TEC)
 - 1 = Not used
 - 2 = Not used
 - 3 = Not used
 - 4 = AI2 = pin 18 at 26-pins connector
 - 5 = AI1 = pin 9 at 26-pins connector
 - 6 = digital temperature sensor, returns degrees Celsius, not Volts
 - 7 = Not used

Returns

`float` – The analog input value [Volts or degrees Celsius].

Raises

`AvantesError` – If there was an error.

get_com_port_name()

Get the IP address of the device.

Returns

`str` – The IP address of the device.

Raises

`AvantesError` – If there was an error.

get_com_type()

Get the communication protocol.

Returns

`str` – The communication type as defined below:

- RS232 = 0
- USB5216 = 1
- USBMINI = 2
- USB7010 = 3
- ETH7010 = 4
- UNKNOWN = -1

Raises

`AvantesError` – If there was an error.

get_dark_pixel_data()

Get the optically black pixel values of the last performed measurement.

You must call [get_data\(\)](#) before you call this method.

Returns

`numpy.ndarray` – The dark pixels.

Raises

[AvantesError](#) – If there was an error.

get_dll_version()

Get the DLL version number.

Returns

`str` – The DLL version number

get_digital_in(*digital_id*)

Get the status of the specified digital input.

Parameters

`digital_id (int)` – The identifier of the digital input to get.

- AS5216:

- 0 = DI1 = Pin 24 at 26-pins connector
- 1 = DI2 = Pin 7 at 26-pins connector
- 2 = DI3 = Pin 16 at 26-pins connector

- Mini:

- 0 = DI1 = Pin 7 on Micro HDMI = Pin 5 on HDMI terminal
- 1 = DI2 = Pin 5 on Micro HDMI = Pin 3 on HDMI Terminal
- 2 = DI3 = Pin 3 on Micro HDMI = Pin 1 on HDMI Terminal
- 3 = DI4 = Pin 1 on Micro HDMI = Pin 19 on HDMI Terminal
- 4 = DI5 = Pin 4 on Micro HDMI = Pin 2 on HDMI Terminal
- 5 = DI6 = Pin 2 on Micro HDMI = Pin 14 on HDMI Terminal

- AS7010:

- 0 = DI1 = Pin 24 at 26-pins connector
- 1 = DI2 = Pin 7 at 26-pins connector
- 2 = DI3 = Pin 16 at 26-pins

Returns

`int` – The digital input value.

Raises

[AvantesError](#) – If there was an error.

get_handle_from_serial(*serial=None*)

Get the handle ID for the specified serial number.

Parameters

serial (`str`) – The serial number. Default is to get the status for this object.

Returns

`int` – The handle.

Raises

AvantesError – If there was an error.

static find(path='avaspecx64.dll', port_id=-1, nmax=16)

Returns device information for each spectrometer that is connected.

Parameters

- **path** (`str`) – The path to the Avantes SDK.
- **port_id** (`int`) – ID of port to be used. One of:
 - -1: Use both Ethernet (AS7010) and USB ports
 - 0: Use USB port
 - 1..255: Not supported in v9.7 of the SDK
 - 256: Use Ethernet port (AS7010)
- **nmax** (`int`, optional) – The maximum number of devices that can be in the list.

Returns

`list` of **AvsIdentityType** – The information about the devices.

get_status_by_serial(serial=None)

Get the handle ID for the specified serial number.

Parameters

serial (`str`, optional) – The serial number. Default is to get the status for this object.

Returns

`int` – The status.

Raises

AvantesError – If there was an error.

done()

Closes communication and releases internal storage.

disconnect()

Closes communication with the spectrometer.

get_ip_config()

Retrieve IP settings from the spectrometer.

Use this function to read the Ethernet settings of the spectrometer, without having to read the complete device configuration structure.

Returns

EthernetSettingsType – The Ethernet settings of the spectrometer.

Raises

AvantesError – If there was an error.

get_lambda()

Returns the wavelength values corresponding to the pixels if available.

Returns

numpy.ndarray – The wavelength value of each pixel.

Raises

AvantesError – If there was an error.

get_num_devices()

Scans for attached devices and returns the number of devices detected.

Deprecated function, replaced by [*update_usb_devices\(\)*](#). The functionality is identical.

Returns

int – The number of devices found.

get_oem_parameter()

Returns the OEM data structure available on the spectrometer.

Returns

OemDataType – The OEM parameters.

Raises

AvantesError – If there was an error.

get_parameter()

Returns the device information of the spectrometer.

Returns

DeviceConfigType – The device parameters.

Raises

AvantesError – If there was an error.

get_saturated_pixels()

Returns, for each pixel, if a pixel was saturated (1) or not (0).

Returns

numpy.ndarray – The saturation state of each pixel.

Raises

AvantesError – If there was an error.

get_version_info()

Returns software version information.

Returns

- *str* – FPGA software version.
- *str* – Firmware version.
- *str* – DLL version.

Raises

AvantesError – If there was an error.

get_data()

Returns the pixel values of the last performed measurement.

Returns

- `int` – Tick count the last pixel of the spectrum was received by the microcontroller. Ticks are in 10 microsecond units since the spectrometer started.
- `numpy.ndarray` – The pixel values.

Raises

`AvantesError` – If there was an error.

get_num_pixels()

Returns the number of pixels of a spectrometer.

Raises

`AvantesError` – If there was an error.

heartbeat(`req_type`)

Monitor the (heartbeat) functions of the spectrometer.

This function applies only to the AS7010 platform. See the DLL manual for more details.

Parameters

`req_type` (`int`) – The heartbeat request values used to control heartbeat functions.

Returns

`HeartbeatRespType` – The heartbeat response structure received from the spectrometer.

Raises

`AvantesError` – If there was an error.

init(`port_id`)

Initializes the communication interface with the spectrometers and the internal data structures.

For Ethernet devices this function will create a list of available Ethernet spectrometers within all the network interfaces of the host.

Parameters

`port_id` (`int`) – ID of port to be used. One of:

- -1: Use both Ethernet (AS7010) and USB ports
- 0: Use USB port
- 1..255: Not supported in v9.7 of the SDK
- 256: Use Ethernet port (AS7010)

Returns

`int` – On success, the number of connected or found devices.

Raises

`AvantesError` – If no devices were found.

measure(*num_measurements*, *window_handle*=None)

Starts measurement on the spectrometer.

Parameters

- **num_measurements** (`int`) – Number of measurements to acquire. Use -1 to measure continuously until `stop_measure()` is called.
- **window_handle** (`ctypes.c_void_p`, optional) – Window handle to notify application measurement result data is available. The DLL sends a message to the window with command: `WM_MEAS_READY`, with `SUCCESS(0)`, the number of scans that were saved in RAM (if `m_StoreToRAM` parameter > 0, see `ControlSettingsType`), or `INVALID_MEAS_DATA` as `WPARAM` value and `a_hDevice` as `LPARAM` value. Set this value to `None` if a callback is not needed.

Raises

`AvantesError` – If there was an error.

measure_callback(*num_measurements*, *callback*=None)

Starts measurement on the spectrometer.

Parameters

- **num_measurements** (`int`) – Number of measurements to acquire. Use -1 to measure continuously until `stop_measure()` is called.
- **callback** (`MeasureCallback`, optional) – A function to notify application measurement result data is available. The DLL will call the given function to notify a measurement is ready and pass two parameters. The first parameter is a reference to the DLL handle. The second parameter is a reference to an integer value: `SUCCESS(0)` if a new scan is available, or the number of scans that were saved in RAM (if `m_StoreToRAM` parameter > 0, see `ControlSettingsType`), or `INVALID_MEAS_DATA(-8)`. Set this value to `None` if a callback is not needed.

Examples

```
from msl.equipment.resources.avantes import MeasureCallback

@MeasureCallback
def avantes_callback(handle, info):
    print('The DLL handle is:', handle.contents.value)
    if info.contents.value == 0: # equals 0 if everything is okay
        print(' callback data:', ava.get_data())

# here "ava" is a reference to the AvaSpec class
ava.measure_callback(-1, avantes_callback)
```

Raises

`AvantesError` – If there was an error.

poll_scan()

Determines if new measurement results are available.

Returns

`int` – Whether there is a scan available: 0 (No) or 1 (Yes).

Raises

`AvantesError` – If there was an error.

prepare_measure(*config*)

Prepares measurement on the spectrometer using the specified measurement configuration.

Parameters

`config` (`MeasConfigType`) – The measurement configuration.

Raises

`AvantesError` – If there was an error.

register(*handle*)

Installs an application windows handle to which device attachment/removal messages have to be sent.

Parameters

`handle` (`cetypes.c_void_p`) – Application window handle.

Raises

`AvantesError` – If there was an error.

reset_device()

Performs a hard reset on the given spectrometer.

This function only works with the AS7010 platform.

During reset of the spectrometer, all spectrometer HW modules (microprocessor and USB controller) will be reset at once. The spectrometer will start its reset procedure right after sending the command response back to the host.

Raises

`AvantesError` – If there was an error.

set_analog_out(*port_id*, *value*)

Sets the analog output value for the specified analog identifier.

Parameters

- **port_id** (`int`) – Identifier for one of the two output signals:

– AS5216:

 * 0 = AO1 = pin 17 at 26-pins connector

 * 1 = AO2 = pin 26 at 26-pins connector

– Mini:

 * 0 = AO1 = Pin 12 on Micro HDMI = Pin 10 on HDMI terminal

 * 1 = AO2 = Pin 14 on Micro HDMI = Pin 12 on HDMI terminal

– AS7010:

 * 0 = AO1 = pin 17 at 26-pins connector

- * 1 = AO2 = pin 26 at 26-pins connector
- **value** (`float`) – DAC value to be set in Volts (internally an 8-bits DAC is used) with range 0 - 5.0V.

Raises

`AvantesError` – If there was an error.

`set_digital_out(port_id, value)`

Sets the digital output value for the specified digital identifier.

Parameters

- **port_id** (`int`) – Identifier for one of the 10 output signals:
 - AS5216:
 - * 0 = DO1 = pin 11 at 26-pins connector
 - * 1 = DO2 = pin 2 at 26-pins connector
 - * 2 = DO3 = pin 20 at 26-pins connector
 - * 3 = DO4 = pin 12 at 26-pins connector
 - * 4 = DO5 = pin 3 at 26-pins connector
 - * 5 = DO6 = pin 21 at 26-pins connector
 - * 6 = DO7 = pin 13 at 26-pins connector
 - * 7 = DO8 = pin 4 at 26-pins connector
 - * 8 = DO9 = pin 22 at 26-pins connector
 - * 9 = DO10 = pin 25 at 26-pins connector
 - Mini:
 - * 0 = DO1 = Pin 7 on Micro HDMI = Pin 5 on HDMI terminal
 - * 1 = DO2 = Pin 5 on Micro HDMI = Pin 3 on HDMI Terminal
 - * 2 = DO3 = Pin 3 on Micro HDMI = Pin 1 on HDMI Terminal
 - * 3 = DO4 = Pin 1 on Micro HDMI = Pin 19 on HDMI Terminal
 - * 4 = DO5 = Pin 4 on Micro HDMI = Pin 2 on HDMI Terminal
 - * 5 = DO6 = Pin 2 on Micro HDMI = Pin 14 on HDMI Terminal
 - * 6 = Not used
 - * 7 = Not used
 - * 8 = Not used
 - * 9 = Not used
 - AS7010:
 - * 0 = DO1 =pin 11 at 26-pins connector
 - * 1 = DO2 = pin 2 at 26-pins connector
 - * 2 = DO3 = pin 20 at 26-pins connector

- * 3 = DO4 = pin 12 at 26-pins connector
 - * 4 = DO5 = pin 3 at 26-pins connector
 - * 5 = DO6 = pin 21 at 26-pins connector
 - * 6 = DO7 = pin 13 at 26-pins connector
 - * 7 = DO8 = pin 4 at 26-pins connector
 - * 8 = DO9 = pin 22 at 26-pins connector
 - * 9 = DO10 = pin 25 at 26-pins connector
- **value** (`int`) – The value to be set (0 or 1).

Raises

AvantesError – If there was an error.

set_oem_parameter(*parameter*)

Sends the OEM data structure to the spectrometer.

Parameters

parameter (`OemDataType`) – The OEM data structure.

Raises

AvantesError – If there was an error.

set_parameter(*parameter*)

Overwrites the device configuration.

Please note that `OemDataType` is part of the `DeviceConfigType` in EEPROM (see section 3.5 of DLL manual). Precautions must be taken to prevent OEM data overwrites when using `set_parameter()` method together with `set_oem_parameter()`.

Parameters

parameter (`DeviceConfigType`) – The device parameters.

Raises

AvantesError – If there was an error.

set_prescan_mode(*boolean*)

If a prescan is set, the first measurement result will be skipped.

This function is only useful for the AvaSpec-3648 because this detector can be operated in prescan mode, or clear-buffer mode (see DLL manual).

Parameters

boolean (`bool`) – If `True`, the first measurement result will be skipped (prescan mode), else the detector will be cleared before each new scan (clear-buffer mode).

Raises

AvantesError – If there was an error.

set_pwm_out(*port_id, frequency, duty_cycle*)

Selects the PWM functionality for the specified digital output.

The PWM functionality is not supported on the Mini.

Parameters

- **port_id** (`int`) – Identifier for one of the 6 PWM output signals:
 - 0 = DO1 = pin 11 at 26-pins connector
 - 1 = DO2 = pin 2 at 26-pins connector
 - 2 = DO3 = pin 20 at 26-pins connector
 - 4 = DO5 = pin 3 at 26-pins connector
 - 5 = DO6 = pin 21 at 26-pins connector
 - 6 = DO7 = pin 13 at 26-pins connector
- **frequency** (`int`) – Desired PWM frequency (500 - 300000) [Hz]. For the AS5216, the frequency of outputs 0, 1 and 2 is the same (the last specified frequency is used) and also the frequency of outputs 4, 5 and 6 is the same. For the AS7010, you can define six different frequencies.
- **duty_cycle** (`int`) – Percentage high time in one cycle (0 - 100). For the AS5216, channels 0, 1 and 2 have a synchronized rising edge, the same holds for channels 4, 5 and 6. For the AS7010, rising edges are unsynchronized.

Raises

AvantesError – If there was an error.

set_sensitivity_mode(*value*)

Set the sensitivity mode.

This method is supported by the following detector types: HAMS9201, HAMG9208_512, SU256LSB and SU512LDB with the appropriate firmware version.

Parameters

value (`int`) – 0 for low noise, >0 for high sensitivity

Raises

AvantesError – If there was an error.

set_sync_mode(*enable*)

Disables/enables support for synchronous measurement.

Parameters

enable (`bool`) – `False` to disable sync mode, `True` to enable sync mode.

Raises

AvantesError – If there was an error.

stop_measure()

Stops the measurement.

Raises

AvantesError – If there was an error.

suppress_stray_light(*factor*)

Returns the stray light corrected pixel values of a dark corrected measurement.

Parameters

factor (`float`) – Multiplication factor for the stray light algorithm.

Returns

- `numpy.ndarray` – Scope minus dark.
- `numpy.ndarray` – Stray light suppressed.

Raises

`AvantesError` – If there was an error.

`update_eth_devices(nmax=16)`

Return the number of Ethernet devices that are connected to the computer.

Internally checks the list of connected Ethernet devices and returns the number of devices attached. If the `properties` attribute contains a key 'port_id' with a value of -1 then the returned value also includes the number of USB devices.

Parameters

`nmax` (`int`, optional) – The maximum number of devices that can be found.

Returns

`list` of `BroadcastAnswerType` – The information about the devices.

`update_usb_devices()`

Return the number of USB devices that are connected to the computer.

Internally checks the list of connected USB devices and returns the number of devices attached. If the `properties` attribute contains a key 'port_id' with a value of -1 then the returned value also includes the number of Ethernet devices.

Returns

`int` – The number of devices found.

`use_high_res_adc(enable)`

Enable the 16-bit AD converter.

When using the 16 bit ADC in full High Resolution mode (0..65535), please note that the irradiance intensity calibration, as well as the nonlinearity calibration are based on the 14bit ADC range. Therefore, if using the nonlinearity correction or irradiance calibration in your own software using the High Resolution mode, you need to apply the additional correction with ADCFactor (= 4.0), as explained in detail in section 4.6.1 and 4.6.3 of the manual.

Parameters

`enable` (`bool`) – If `True` use a 16-bit AD converter, otherwise use a 14-bit ADC.

Raises

`AvantesError` – If there was an error.

`msl.equipment.resources.bentham` package

Resources for equipment from Bentham.

Submodules

`msl.equipment.resources.bentham.benhw32 module`

A wrapper around the 32-bit Bentham benhw32_cdecl SDK.

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

Bases: `Server32`

A wrapper around the 32-bit Bentham benhw32_cdecl SDK.

Do not instantiate this class directly.

The SDK is provided for 32-bit Windows only. This module is run on a 32-bit `Server` so that the the `Bentham` class can be run on a 64-bit Python interpreter in order to access the functions in the SDK from a 64-bit process.

`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 `Bentham32` class.

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

Bases: `Connection`

A wrapper around the `Bentham32` class.

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

The `properties` for a Bentham connection supports the following key-value pairs in the `Connections Database`:

```
'cfg': str, the path to the System.cfg file [default: None]  
'atr': str, the path to the System.atr file [default: None]
```

If the `cfg` and `atr` values are not defined in the `Connections Database` then you will have to call `build_system_model()`, `load_setup()` and `initialise()` (in that order) to configure the SDK.

Do not instantiate this class directly. Use the `connect()` method to connect to the equipment.

Parameters

`record` (`EquipmentRecord`) – A record from an `Equipment-Register Database`.

`auto_measure()`

`build_system_model(path)`

Set the model configuration file.

Parameters

`path` (`str`) – The path to the `System.cfg` file.

`disconnect()`

Disconnect from the SDK and from the 32-bit server.

`errcheck(result, *args, **kwargs)`

Checks whether a function call to the SDK was successful.

`get(hw_id, token, index)`

`get_component_list()`

`get_hardware_type(hw_id)`

```
get_mono_items(hw_id)
property wavelength
initialise()
    Initialize the connection.
load_setup(path)
    Load the setup file.

Parameters
    path (str) – The path to the System.atr file.

park()
select_wavelength(wavelength)
set(hw_id, token, index, value)
version()
    str: The version number of the SDK.
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.cmi package**](#)

Resources for equipment from [CMI](#).

Submodules

[**msl.equipment.resources.cmi.sia3 module**](#)

Establishes a connection to the Switched Integrator Amplifier (SIA3 board) that is designed by the [Czech Metrology Institute](#).

```
class msl.equipment.resources.cmi.sia3.IntegrationTime(value, names=None, *values,
                                                       module=None,
                                                       qualname=None, type=None,
                                                       start=1, boundary=None)
```

Bases: [IntEnum](#)

The amount of time to integrate the photo-diode signal.

```
TIME_50u = 5
TIME_100u = 6
TIME_1m = 7
TIME_10m = 8
TIME_20m = 9
TIME_100m = 10
TIME_200m = 11
TIME_500m = 12
TIME_1 = 13
TIME_2 = 14
```

class msl.equipment.resources.cmi.sia3.SIA3(*record*)
Bases: *ConnectionSerial*

Establishes a connection to the Switched Integrator Amplifier (SIA3 board) that is designed by the Czech Metrology Institute.

Do not instantiate this class directly. Use the `connect()` method to connect to the equipment.

Parameters

record (*EquipmentRecord*) – A record from an *Equipment-Register Database*.

GAIN

The gain (i.e., the integration time)

alias of *IntegrationTime*

set_integration_time(*time*)

Set the integration time (i.e., the gain).

Parameters

time (*IntegrationTime*) – The integration time as a *IntegrationTime* enum value or member name, e.g., `sia.set_integration_time('10m')`, `sia.set_integration_time(sia.GAIN.TIME_10m)` and `sia.set_integration_time(8)` are all equivalent statements.

set_ps(*ps*)

Set the timer pre-scale value.

The timer pre-scale value divides the microprocessor internal frequency by something similar to 2^{PS} . Therefore, to reach a 2-second integration time the *ps* value must be set to the maximum value of 7.

Parameters

ps (*int*) – The timer pre-scale value. Must be in the range [0, 7].

Raises

ValueError – If the value of *ps* is invalid.

msl.equipment.resources.dataray package

Resources for equipment from DataRay.

Submodules

msl.equipment.resources.dataray.datarayocx_32 module

Load the 32-bit DATARAYOCX library using [MSL-LoadLib](#).

```
class msl.equipment.resources.dataray.datarayocx_32.DataRay0CX32(host, port,  
**kwargs)
```

Bases: [Server32](#)

Communicates with the 32-bit DATARAYOCX library.

Tested with the WinCamD-LCM-8.0D36 software version.

`wait_to_configure()`

Wait until the camera has been configured.

`capture(timeout)`

Capture an image.

`start()`

Start the camera.

`stop()`

Stop the camera.

`shutdown_handler()`

Stop the camera and close the application.

msl.equipment.resources.dataray.datarayocx_64 module

Establishes a connection to the DATARAYOCX library developed by DataRay Inc.

```
class msl.equipment.resources.dataray.datarayocx_64.DataRay0CX64(record)
```

Bases: [Connection](#)

A wrapper around the [DataRay0CX32](#) class.

This class can be used with either a 32- or 64-bit Python interpreter to call the 32-bit functions in the DATARAYOCX library. A GUI is created to configure and visualize the images taken by the camera.

Tested with the WinCamD-LCM-8.0D36 software version.

The [properties](#) for a DataRay connection supports the following key-value pairs in the [Connections Database](#):

```
'area_filter': int, area filter: 1=1pixel, 2=3pixels, 3=5pixels, 4=7pixels, 5=9pixels [default: 1]
'camera_index': int, the camera to use (between 0 and 7; 0=first camera found) [default: 0]
'centroid_method': int, the centroid method to use (0, 1 or 2) [default: 0]
'filter': float, percent full scale filter (0, 0.1, 0.2, 0.5, 1, 2, 5 or 10) [default: 0.2]
'major_minor_method': int, the major/minor method to use (0, 1 or 2) [default: 0]
'ui_size': int, the size of the User Interface (value=height of a button in pixels) [default: 25]
```

Do not instantiate this class directly. Use the `connect()` method to connect to the equipment.

Parameters

`record (EquipmentRecord)` – A record from an *Equipment-Register Database*.

`wait_to_configure()`

Wait until the camera has been configured.

This is a blocking call and waits until you close the popup Window.

`capture(timeout=10, restart=False)`

Capture an image.

Parameters

- `timeout (float, optional)` – The maximum number of seconds to wait to capture an image.
- `restart (bool, optional)` – Whether to keep the camera running after the image is captured.

Returns

`dict` – The information about the captured image. The key-value pairs are:

- `adc_peak_%: float`, the maximum ADC value (as a percentage)
- `area_filter: int`, area filtering applies a convolution to the pixels (1=1pixel, 2=3pixels, 3=5pixels, 4=7pixels, 5=9pixels)
- `centroid: tuple`, the (x, y) centroid value
- `centroid_filter_size: int`, centroid filter size (in pixels)
- `centroid_type: int`, the centroid type (0, 1 or 2)
- `clip_level_centroid: float`, the centroid clip level (between 0 and 1)
- `clip_level_geo: float`, the geometric clip level (between 0 and 1)
- `crosshair: float`, the angle between the horizontal x-axis and the solid crosshair line (in degrees)
- `eff_2w: float`, the effective beam size (in um)

- `effective_centroid`: `tuple`, the effective (x, y) centroid value
- `effective_geo_centroid`: `tuple`, the effective (x, y) geometric centroid value
- `ellip`: `float`, the ratio between the minor/major axis
- `elp`: `float`, the beam azimuthal angle for ISO 11146 (in degrees)
- `exposure_time`: `float`, the exposure time (in ms)
- `filter_full_scale`: `float`, used in the triangular weighting smoothing function
- `image`: `numpy.ndarray`, the camera image
- `image_zoom`: `float`, the zoom factor of the image
- `imager_gain`: `float`, the gain used by the imager
- `inc_p`: `float`, Dxx power (in Watts)
- `inc_power_area`: `float`, Dxx beam area (in mm²)
- `inc_power_major`: `float`, Dxx beam diameter along the major axis (in mm)
- `inc_power_mean`: `float`, Dxx mean beam diameter (in mm)
- `inc_power_minor`: `float`, Dxx beam diameter along the minor axis (in mm)
- `is_fast_update`: `bool`, whether the camera is in fast update or normal mode
- `is_full_resolution`: `bool`, whether the camera is set to full resolution
- `maj_iso`: `float`, the ISO 11146 beam size along the major axis (in um)
- `major`: `float`, the beam size along the major axis (in um)
- `major_minor_method`: `int`, the major/minor method used (0, 1 or 2)
- `mean`: `float`, the mean beam size (in um)
- `mean_theta`: `float`, Dxx mean angle (in mrad)
- `min_iso`: `float`, the ISO 11146 beam size along the minor axis (in um)
- `minor`: `float`, the beam size along the minor axis (in um)
- `num_averages`: `int`, the number of averages per capture
- `num_captures`: `int`, the number of images that have been captured
- `orient`: `float`, the angle between the horizontal x-axis and the major or minor axis closest to the horizontal x-axis (in degrees)
- `peak`: `tuple`, the peak location in (x, y), usually zero
- `pixel_width_um`: `float`, the width of a pixel (in um)
- `pixel_height_um`: `float`, the height of a pixel (in um)

- `pk_to_avg`: `float`, the peak to average value
- `plateau_uniformity`: `float`, the flatness of the plateau (between 0 and 1)
- `profile_x`: `numpy.ndarray`, the profile in the X direction
- `profile_y`: `numpy.ndarray`, the profile in the Y direction
- `roi`: `tuple`, the selected region of interest (x, y, width, height)
- `xc`: `float`, the centroid position along the x axis (in um)
- `xg`: `float`, the geometric centroid position along the x axis (in um)
- `xp`: `float`, the peak-intensity centroid position along the x axis (in um)
- `xu`: `float`, the user-selected centroid position along the x axis (in um)
- `yc`: `float`, the centroid position along the y axis (in um)
- `yg`: `float`, the geometric centroid position along the y axis (in um)
- `yp`: `float`, the peak-intensity centroid position along the y axis (in um)
- `yu`: `float`, the user-selected centroid position along the y axis (in um)

Raises

`DataRayError` – If not successful.

disconnect()

Disconnect from the camera.

start()

Start the camera.

stop()

Stop the camera.

msl.equipment.resources.electron_dynamics package

Resources for equipment from Electron Dynamics.

Submodules

msl.equipment.resources.electron_dynamics.tc_series module

Establishes a connection to a TC Series Temperature Controller from Electron Dynamics Ltd.

class `msl.equipment.resources.electron_dynamics.tc_series.TCSeries(record)`

Bases: `ConnectionSerial`

Establishes a connection to a TC Series Temperature Controller from Electron Dynamics Ltd.

Do not instantiate this class directly. Use the `connect()` method to connect to the equipment.

Parameters

record (*EquipmentRecord*) – A record from an *Equipment-Register Database*.

get_alarm()

Get the alarm parameters.

Returns

list – The alarm parameters. See [set_alarm\(\)](#) for more details.

get_control()

Get the control parameters.

Returns

list – The control parameters. See [set_control\(\)](#) for more details.

get_output()

Get the output parameters.

Returns

list – The output parameters. See [set_output\(\)](#) for more details.

get_sensor()

Get the sensor parameters.

Returns

list – The sensor parameters. See [set_sensor\(\)](#) for more details.

get_setpoint()

Get the setpoint parameters.

Returns

list – The setpoint parameters. See [set_setpoint\(\)](#) for more details.

get_status()

Get the status.

Returns

list – The status parameters.

set_alarm(*alarm_type*, *minimum*, *maximum*, *ok_min*, *ok_max*, *limit_min*, *limit_max*)

Set the alarm parameters.

This corresponds to the `c` command in the Commands manual.

Parameters

- **alarm_type** (`int`) – The alarm type. One of:
 - 0: None
 - 1: Minimum
 - 2: Maximum
 - 3: Both
- **minimum** (`float`) – Sets the temperature below which the alarm is activated.

- **maximum** (`float`) – Sets the temperature above which the alarm is activated.
- **ok_min** (`float`) – Sets the lower temperature difference point from the setpoint for temperature OK.
- **ok_max** (`float`) – Sets the higher temperature difference point from the setpoint for temperature OK.
- **limit_min** (`float`) – Sets the temperature minimum, below which the drive output is disabled.
- **limit_max** (`float`) – Sets the temperature maximum, above which the drive output is disabled.

set_control(*control_type*, *p*, *i*, *d*, *d_filter*, *dead_band*, *power_up_state*)

Set the control type and the control parameters.

This corresponds to the a command in the Commands manual.

Parameters

- **control_type** (`int`) – The control type. One of:
 - 0: None
 - 1: On/Off - Output drive is only fully On (heating or cooling) or Off
 - 2: Proportional (P)
 - 3: Proportional and Integral (PI)
 - 4: Proportional, Integral and Derivative (PID)
- **p** (`float`) – The proportional (gain) value. With proportional action, the controller output is proportional to the temperature error from the setpoint. The proportional terms sets the gain for this where: Output = (setpoint-actual temperature) * proportional term
- **i** (`float`) – The integral value. With integral action, the controller output is proportional to the amount of time the error is present. Integral action eliminates offset. The integral term is a time unit in seconds. NB for larger effects of integration reduce the integral time, also for operation without integral, integral time can be set to a large number e.g. 1e6.
- **d** (`float`) – The derivative value. With derivative action, the controller output is proportional to the rate of change of the measurement or error. The controller output is calculated by the rate of change of the measurement with time, in seconds. To increase the derivative action increase the derivative value. See also Derivative Filter (*d_filter*).
- **d_filter** (`float`) – The derivative filter is a low pass filter function on the derivative value. This allows the filtration of noise components which are a problem with a pure derivative function. The filter value should be set to be between 0 and 1.
- **dead_band** (`float`) – For use with On/Off control the dead band specifies the temperature range around the set point where the output is zero. For example:

- Temperature > setpoint + dead_band (Fully Cooling)
 - Temperature < setpoint - dead_band (Fully Heating)
 - Temperature < setpoint + dead_band AND > setpoint-dead_band (Output off)
- **power_up_state** (`int`) – This sets the temperature control state from power up. One of:
 - 0: Off
 - 1: On
 - 2: Same as the last setting prior to power off

set_output(*polarity*, *minimum*, *maximum*, *frequency*)

Set the output parameters.

This corresponds to the **g** command in the Commands manual.

Parameters

- **polarity** (`int`) – Sets the polarity of the output drive. One of:
 - 0: Negative
 - 1: Positive
- **minimum** (`float`) – Sets the minimum value limit of the output. Range -100 to +100.
- **maximum** (`float`) – Sets the maximum value limit of the output. Range -100 to +100.
- **frequency** (`float`) – Sets the pulse-width modulation repetition frequency of the output drive. Range 20 to 1000 Hz.

set_output_drive(*mode*, *value*)

Set the output drive state and value.

This corresponds to the **m** command in the Commands manual.

Parameters

- **mode** (`int`) – The drive mode. Either 0 (Off) or 1 (On).
- **value** (`int`) – Percent output.

set_sensor(*sensor*, *x2*, *x*, *c*, *unit*, *averaging*, *rl=22000*)

Set the sensor type and the sensor parameters.

This corresponds to the **e** command in the Commands manual.

Parameters

- **sensor** (`int`) – The sensor type. One of:
 - 0: Voltage
 - 1: PT100
 - 2: LM35
 - 3: LM50

- 4: LM60
 - 5: LM61
 - 6: NTC Thermistor
 - 7: RES
 - 8: PT1000
 - 9: RTD
- **x2** (`float`) –
 - **x** (`float`) –
 - **c** (`float`) – The x_2 , x and c parameters are quadratic coefficients than can be used to convert the sensor voltage into a temperature, i.e., $\text{temperature} = (v * v * x_2) + (v * x) + c$, where v is the measured sensor voltage.
- For NTC thermistors x_2 is the beta value as specified for the thermistor type, x is the resistance at 25 deg C, and, c is still the offset.
- **unit** (`str`) – The temperature units. One of:
 - 'C': Centigrade
 - 'F': Fahrenheit
 - 'K': Kelvin
 - 'V': Voltage
 - 'R': Resistance
- **averaging** (`int`) – Set the averaging to be 0 (Off) or 1 (On).
 - **r1** (`float`, optional) – Used for NTC or RES sensors. This value corresponds to the RL drive resistance.

`set_setpoint`(*method*, *value*, *pot_range*, *pot_offset*)

Set the setpoint parameters.

This corresponds to the **i** command in the Commands manual.

Parameters

- **method** (`int`) – The temperature setpoint can be set via software or by altering the potentiometer on the temperature controller hardware. One of:
 - 0: Potentiometer
 - 1: Software
 - 2: Input
- **value** (`float`) – The setpoint value.
- **pot_range** (`float`) – The temperature range of the potentiometer.
- **pot_offset** (`float`) – The minimum temperature point of the potentiometer.

set_test(*mode*, *arg1*, *arg2*, *arg3*, *arg4*, *arg5*, *arg6*, *arg7*)

Set the test parameters.

This corresponds to the **k** command in the Commands manual.

Parameters

- **mode** (`int`) – The test mode. One of:
 - 0: Off
 - 1: Normal
 - 2: Temperature cycle
 - 3: Temperature ramp
 - 4: Auto tune
- **arg1** (`float`) – The maximum cycle value or the test time (in seconds).
- **arg2** (`float`) – The start temperature or setpoint value.
- **arg3** (`float`) – The end temperature or end peak test value.
- **arg4** (`float`) – The rate 1 or the calc PID value.
- **arg5** (`float`) – The rate 2 or the run PID value.
- **arg6** (`float`) – The time 1 or undo PID value.
- **arg7** (`float`) – The time 2 or auto cal value.

msl.equipment.resources.energetiq package

Resources for equipment from Energetiq.

Submodules

msl.equipment.resources.energetiq.eq99 module

Communicate with the EQ-99 Manager from Energetiq.

class `msl.equipment.resources.energetiq.eq99.EQ99(record)`

Bases: `ConnectionSerial`

Communicate with the EQ-99 Manager from Energetiq.

Do not instantiate this class directly. Use the `connect()` method to connect to the equipment.

Parameters

record (`EquipmentRecord`) – A record from an *Equipment-Register Database*.

identity()

Query the instrument identification.

Returns

`str` – Returns the identification string for the instrument in the following format: *Energetiq Model SN Ver Build*

reset()

Resets the instrument to factory defaults and the output is shut off.

The unit remains in remote mode.

get_beep()

Query whether beeps are enabled.

Returns

`bool` – Whether beeps are enabled.

set_beep(beep=2)

Set the beep value.

Parameters

`beep` (`int` or `bool`, optional) – Causes the instrument to beep, or enables or disabled the beep sound for error messages and other events that generate and audible response. Possible values are

- 0 or `False` – Disable the beep sound
- 1 or `True` – Enable the beep sound
- 2 – Generate one beep

get_brightness()

Query the display brightness.

Returns

`int` – Returns the value of the display brightness (between 0 and 100).

set_brightness(brightness)

Set the display brightness.

Parameters

`brightness` (`int`) – Sets the display brightness level from 0 to 100 percent. There are only 8 brightness levels (each separated by about 12.5 percent) and the brightness value is used to select an appropriate level.

delay(milliseconds)

Specify a delay to use in command processing.

Parameters

`milliseconds` (`int`) – Causes command processing to be delayed for the specified number of milliseconds. Valid range is from 1 to 30000 milliseconds.

condition_register()

Query LDLS condition register.

The condition register reflects the state of the instrument at the time the condition register is read.

The bitmask sequence is as follows:

Index	Value	Description
0	1	Interlock
1	2	Controller not detected
2	4	Controller fault
3	8	Lamp fault
4	16	Output on
5	32	Lamp on
6	64	Laser on
7	128	Laser stable
8	256	Shutter open

Returns

- `int` – The condition register value.
- `str` – The condition register as a bitmask string. For example, a value of 336 is expressed as '`000010101`' (meaning that the interlock is closed, there are no faults, the output is on, the lamp is off, the laser is on, the laser is not stable and the shutter is open).

`event_register()`

Query LDLS event register.

Returns the LDLS event register. The event register reflects the occurrence of any condition since the last time the event register was read. For example, if the output was turned on and then turned off, the Output on the bit in the condition register will be zero, but the same bit in the event register will be one.

The bitmask sequence is as follows:

Index	Value	Description
0	1	Interlock
1	2	Controller not detected
2	4	Controller fault
3	8	Lamp fault
4	16	Output on
5	32	Lamp on
6	64	Laser on
7	128	Laser stable
8	256	Shutter open

Returns

- `int` – The event register value.
- `str` – The event register as a bitmask string. For example, a value of 256 is expressed as '`000000001`' and 32 as '`000001000`'.

`get_exposure_time()`

Query the exposure time.

Returns

`int` – The exposure time, in milliseconds.

set_exposure_time(milliseconds)

Set the exposure time.

Exposure time is used when the shutter exposure mode is set to *Exposure mode* (see [set_exposure_mode\(\)](#)). An exposure is triggered by a shutter button press or the shutter trigger input.

Parameters

`milliseconds (int)` – The exposure time, in milliseconds, from 100 to 30000 ms.

get_exposure_mode()

Query the exposure mode.

Returns

`int` – The exposure mode (0=Manual, 1=Exposure).

set_exposure_mode(mode)

Set the exposure mode.

Same as the Shutter setting in the menu.

Parameters

`mode (int or bool)` – The exposure mode (0=Manual, 1=Exposure).

get_output()

Query the output state.

Returns

`bool` – Whether the output is enabled.

set_output(enable)

Turn the output on or off.

Parameters

`enable (int or bool)` – Whether to enable the output.

get_lamptime()

Query the lamp runtime.

Returns

`float` – The number of hours accumulated while the lamp was on.

set_lamptime(hours)

Set the lamp runtime.

Resets the runtime to the new value. Useful for resetting the runtime to zero when the lamp has been serviced or replaced, or when moving the manager to a new LDLS system.

Parameters

`hours (float)` – The lamp runtime, in hours, between 0 and 9999.

get_shutter_init()

Query the power-up shutter state.

Returns

`int` – The power-up shutter state.

- 0 – Shutter is closed on power-up
- 1 – Shutter is open on power-up

`set_shutter_init(state)`

Set the power-up shutter state

Parameters

`state (int or bool)` – Sets the initial state of the shutter on power-up of the manager

- 0 or `False` – Shutter is closed on power-up
- 1 or `True` – Shutter is open on power-up

`get_shutter_state()`

Query the shutter state.

Returns

`bool` – The state of the shutter.

- `False` – Shutter is closed
- `True` – Shutter is open

`set_shutter_state(state)`

Open, close, or trigger the shutter.

A close command (state equals 0) will always close the shutter, regardless of exposure mode. An open command (state equals 1) will open the shutter if exposure mode is set to Manual, or trigger a shutter if exposure mode is set to Exposure.

Parameters

`state (int or bool)` – The state of the shutter.

- 0 or `False` – Close the shutter
- 1 or `True` – Open or trigger the shutter

`get_trigger_mode()`

Query the trigger mode.

Returns

`int` – The trigger mode. See `set_trigger_mode()` for more details.

`set_trigger_mode(mode)`

Set the trigger mode.

The trigger mode controls how the shutter trigger input controls the operation of the shutter. For more information on trigger modes, see *Shutter Operation* in the *Operating the Instrument* section of the manual for more details.

Parameters

`mode (int)` – The trigger mode.

- 0 – Positive edge trigger

- 1 – Negative edge trigger
- 2 – Positive level trigger
- 3 – Negative level trigger
- 4 – Off (trigger disabled)

get_message_buffer()

Query the internal message buffer.

Returns

`str` – The value of the internal message buffer.

set_message_buffer(*message*)

Set the message buffer.

Parameters

message (`str`) – Sets the internal message buffer, up to a maximum of 16 characters. If more than 16 characters are specified then the additional characters are silently ignored.

get_remote_mode_error()

Query whether errors are displayed while in remote mode.

Returns

`bool` – Whether errors are displayed while in remote mode.

set_remote_mode_error(*enable*)

Set whether to display errors while in remote mode.

This command controls if the instrument will display errors while in remote mode. If set to zero, then errors will not be displayed. If set to one, errors will be displayed. Errors will always accumulate in the error queue.

Parameters

enable (`int` or `bool`) – Whether to display errors while in remote mode.

- 0 or `False` – No not display errors in remote mode
- 1 or `True` – Display errors in remote mode

serial_number()

Query the serial number of the instrument.

Returns

`str` – The serial number of the instrument. This is the same information that is part of the *IDN? query.

get_termination()

Query response terminator.

Returns the current response terminator setting. See `set_termination()` for a complete definition of possible return values.

Returns

`int` – The response terminator.

set_termination(*value*)

Set the response terminator character(s).

This command controls the termination characters used for responses to queries.

Parameters

value (`int`) – The response terminator character(s)

- 0 or 1 – <CR><LF>
- 2 or 3 – <CR>
- 4 or 5 – <LF>
- 6 or 7 – no terminator

run_time()

Query run time.

Returns

`str` – Returns the elapsed time since the unit has been turned on. Format is in HH:MM:SS.ss, where HH is hours, MM is minutes, SS is seconds, and ss is hundredths of a second.

timer()

Query time since the last time this method was called.

Returns

`str` – Returns the elapsed time since the last time this method was called, or, if this is the first time calling this method then the time since unit has been turned on. Format is in HH:MM:SS.ss, where HH is hours, MM is minutes, SS is seconds, and ss is hundredths of a second.

version()

Query the firmware version.

Returns

`str` – Returns the firmware version. This is the same information that is part of the *IDN? query.

msl.equipment.resources.greisinger package

Resources for equipment from Greisinger.

Submodules

msl.equipment.resources.greisinger.gmh3000 module

Communicate with a Greisinger GMH 3000 Series thermometer.

class `msl.equipment.resources.greisinger.gmh3000.GMH3000`(*record*: `EquipmentRecord`)

Bases: `ConnectionSerial`

Communicate with a Greisinger GMH 3000 Series thermometer.

The *properties* for a thermometer connection supports the following key-value pairs in the *Connections Database*:

```
'gmh_address': int, The GMH address of the device [default: 1]
```

Do not instantiate this class directly. Use the `connect()` method to connect to the equipment.

Parameters

`record` (*EquipmentRecord*) – A record from an *Equipment-Register Database*.

channel_count() → `int`

Get the number of channels.

Returns

`int` – The channel count.

clear_max_value() → `float`

Clear the maximum value stored in the device memory.

Returns

`float` – The current value.

clear_min_value() → `float`

Clear the minimum value stored in the device memory.

Returns

`float` – The current value.

display_range() → `tuple[float, float]`

Get the range of the display.

Returns

- `float` – The minimum value that the device can display.
- `float` – The maximum value that the device can display.

firmware_version() → `tuple[int, int]`

Get the version information of the firmware.

Returns

- `int` – The version number.
- `int` – The version identifier.

id_number() → `str`

Get the device ID (serial) number.

Returns

`str` – The ID (serial) number of the device.

max_value() → `float`

Get the maximum value that has been read.

Returns

`float` – The maximum value that has been read since the device was turned on or since `clear_max_value()` was called.

measurement_range() → `tuple[float, float]`

Get the measurement range.

Returns

- `float` – The minimum value that the device can measure.
- `float` – The maximum value that the device can measure.

min_value() → `float`

Get the minimum value that has been read.

Returns

`float` – The minimum value that has been read since the device was turned on or since `clear_min_value()` was called.

offset_correction() → `float`

Get the offset-correction value.

The zero point (intercept in a linear calibration equation) of the measurement will be displaced by this value to compensate for deviations in the temperature probe or in the measuring device.

Returns

`float` – The offset-correction value.

power_off_time() → `int`

Get the power-off time.

Returns

`int` – The number of minutes that the device will automatically power off as soon as this time has elapsed if no key is pressed or if no interface communication takes place. A value of 0 means that power off is disabled.

resolution() → `int`

Get the measurement resolution.

Returns

`int` – The number of digits after the decimal point that is acquired for the measured value.

scale_correction() → `float`

Get the scale-correction factor.

The scale (slope in a linear calibration equation) of the measurement will be changed by this factor to compensate for deviations in the temperature probe or in the measuring device.

Returns

`float` – The scale-correction factor.

set_power_off_time(minutes: int) → `int`

Set the power-off time.

Parameters

minutes (int) – The number of minutes that the device will automatically power off as soon as this time has elapsed if no key is pressed or if no interface communication takes place. A value of 0 means that power off is disabled.

Returns

`int` – The actual power-off time that the device was set to. If you set the power-off time to a value greater than the maximum time allowed, the device automatically coerces the value to be the maximum time.

status() → `int`

Get the system status.

The status value represents a bit mask:

Index	Value	Description
0	1	Max. alarm
1	2	Min. alarm
2	4	Display range overrun
3	8	Display range underrun
4	16	Reserved
5	32	Reserved
6	64	Reserved
7	128	Reserved
8	256	Measuring range overrun
9	512	Measuring range underrun
10	1024	Sensor error
11	2048	Reserved
12	4096	System fault
13	8192	Calculation not possible
14	16384	Reserved
15	32768	Low battery

Returns

`int` – The system status.

unit() → `str`

Get the measurement unit.

Returns

`str` – The measurement unit.

value() → `float`

Get the current measurement value.

Returns

`float` – The current value.

`msl.equipment.resources.mks_instruments package`

Resources for equipment from MKS Instruments.

Submodules

`msl.equipment.resources.mks_instruments.pr4000b module`

Flow and Pressure controller, PR4000B, from MKS Instruments.

`class msl.equipment.resources.mks_instruments.pr4000b.PR4000B(record)`

Bases: `ConnectionSerial`

Flow and Pressure controller, PR4000B, from MKS Instruments.

The default settings for the RS232 connection are:

- Baud rate = 9600
- Data bits = 7
- Stop bits = 1
- Parity = ODD
- Flow control = None

The baud rate and parity can be changed on the controller. The data bits, stop bits, and flow control cannot be changed. A null modem (cross over) cable is required when using a USB to RS232 converter. RS485 support is not implemented.

Do not instantiate this class directly. Use the `connect()` method to connect to the equipment.

Parameters

`record (EquipmentRecord)` – A record from an *Equipment-Register Database*.

```
ERROR_CODES = {'#E001': 'Communication Error', '#E002': 'ADC Overflow or Underflow', '#E003': 'Range Error, Setpoint < 0 or out of range', '#E010': 'Syntax Error', '#E020': 'Failed to execute command', '#W001': 'Offset > 250 mV'}
```

```
UNITS = {0: 'ubar', 1: 'mbar', 2: 'bar', 3: 'mTor', 4: 'Torr', 5: 'KTor', 6: 'Pa', 7: 'kPa', 8: 'mH2O', 9: 'cH2O', 10: 'PSI', 11: 'N/qm', 12: 'SCCM', 13: 'SLM', 14: 'SCM', 15: 'SCFH', 16: 'SCFM', 17: 'mA', 18: 'V', 19: '%', 20: 'C'}
```

```
SIGNAL_MODES = {0: 'METER', 1: 'OFF', 2: 'INDEP', 3: 'EXTRN', 4: 'SLAVE', 5: 'RTD'}
```

```
LIMIT_MODES = {0: 'SLEEP', 1: 'LIMIT', 2: 'BAND', 3: 'MLIMIT', 4: 'MBAND'}
```

```
TAGS = {0: 'SP', 1: 'VA', 2: 'CH', 3: 'FL', 4: 'PR', 5: 'EX'}
```

auto_zero(channel)

Auto zero a channel

Parameters

channel (`int`) – The channel, either 1 or 2.

Returns

`int` – The offset.

default(mode)

Reset to the default configuration.

Parameters

mode (`str`) – The mode to reset. One of Pressure, Flow, P or F (case insensitive).

displays_enable(display, enable)

Turn a display on or off.

Parameters

- **display** (`int`) – The display number [1, 4].
- **enable** (`bool`) – Whether to turn the display on, `True`, or off, `False`.

displays_setup(display, line, tag, channel)

Configure a display.

Parameters

- **display** (`int`) – The display number [1, 4].
- **line** (`int`) – The line number, 1 or 2.
- **tag** (`int` or `str`) – The tag to use (0=SP, 1=VA, 2=CH, 3=FL, 4=PR, 5=EX). For example, setting tag to 4 or 'PR' are equivalent.
- **channel** (`int`) – The channel, either 1 or 2.

display_4(enable)

Whether to enable or disable display 4.

Parameters

enable (`bool`) – Whether to enable or disable display 4.

external_input(channel)

Return the external input of a channel

Parameters

channel (`int`) – The channel, either 1 or 2.

Returns

`float` – The external input.

get_access_channel(channel)

Get the setpoint and the state of the valve of a particular channel.

Parameters

channel (`int`) – The channel, either 1 or 2.

Returns

- `float` – The setpoint value.
- `bool` – Whether the valve is on, `True`, or off, `False`.

get_actual_value(*channel*)

Get the actual value of a particular channel.

Parameters

`channel` (`int`) – The channel, either 1 or 2.

Returns

`float` – The value.

get_address()

Get the address.

Returns

`int` – The address.

get_dead_band(*channel*)

Get the dead band of a particular channel.

Parameters

`channel` (`int`) – The channel, either 1 or 2.

Returns

`float` – The dead band.

get_dialog()

Get the current dialog index that is displayed.

Returns

`int` – The dialog index.

get_display_text()

Get the display text.

Returns

`str` – The display text.

get_external_input_range(*channel*)

Get the external input range of a channel.

Parameters

`channel` (`int`) – The channel, either 1 or 2.

Returns

`int` – The external input range.

get_external_output_range(*channel*)

Get the external output range of a channel.

Parameters

`channel` (`int`) – The channel, either 1 or 2.

Returns

`int` – The external output range.

get_formula_relay(*channel*)

Get the relay formula of a particular channel.

Parameters

channel (`int`) – The channel, either 1 or 2.

Returns

str – The formula.

get_formula_temporary(*channel*)

Get the temporary formula of a particular channel.

Parameters

channel (`int`) – The channel, either 1 or 2.

Returns

str – The formula.

get_gain(*channel*)

Get the gain of a particular channel.

Parameters

channel (`int`) – The channel, either 1 or 2.

Returns

float – The gain.

get_input_range(*channel*)

Get the input range of a channel.

Parameters

channel (`int`) – The channel, either 1 or 2.

Returns

int – The input range.

get_interface_mode()

Get the interface mode.

Returns

int – The interface mode.

get_limit_mode(*channel*)

Get the limit mode of a particular channel.

Parameters

channel (`int`) – The channel, either 1 or 2.

Returns

- **int** – The index of the limit mode.
- **str** – The name of the limit mode.

get_linearization_point(*channel, point*)

Get the point in the linearization table of a particular channel.

Parameters

- **channel** (`int`) – The channel, either 1 or 2.

- **point** (`int`) – The point in the table [0, 10].

Returns

- `float` – The x value.
- `float` – The y value.

get_linearization_size(channel)

Get the size of the linearization table of a particular channel.

Parameters

`channel` (`int`) – The channel, either 1 or 2.

Returns

`int` – The size of the table.

get_lower_limit(channel)

Get the lower limit of a particular channel.

Parameters

`channel` (`int`) – The channel, either 1 or 2.

Returns

`float` – The lower limit.

get_offset(channel)

Get the offset of a particular channel.

Parameters

`channel` (`int`) – The channel, either 1 or 2.

Returns

`int` – The offset.

get_output_range(channel)

Get the output range of a channel.

Parameters

`channel` (`int`) – The channel, either 1 or 2.

Returns

`int` – The output range.

get_range(channel)

Get the range and unit of a channel.

Parameters

`channel` (`int`) – The channel, either 1 or 2.

Returns

- `float` – The range.
- `int` – The unit index.
- `str` – The unit name.

get_relays(channel)

Get the relay state of a channel.

Parameters

`channel` (`int`) – The channel, either 1 or 2.

Returns

`bool` – Whether the relay is enabled or disabled.

get_remote_mode()

Get the remote operation mode.

Returns

`bool` – Whether the remote operation mode is enabled, `True`, or disabled, `False`.

get_resolution()

Get whether 16-bit resolution is enabled.

Returns

`bool` – Whether 16-bit resolution is enabled, `True`, or disabled, `False`.

get_rtd_offset(*channel*)

Get the RTD offset of a particular channel.

Parameters

`channel` (`int`) – The channel, either 1 or 2.

Returns

`int` – The offset.

get_scale(*channel*)

Get the scale of a particular channel.

Parameters

`channel` (`int`) – The channel, either 1 or 2.

Returns

`float` – The scale.

get_setpoint(*channel*)

Get the setpoint of a particular channel.

Parameters

`channel` (`int`) – The channel, either 1 or 2.

Returns

`float` – The setpoint.

get_signal_mode(*channel*)

Get the signal mode of a particular channel.

Parameters

`channel` (`int`) – The channel, either 1 or 2.

Returns

- `int` – The index number of the signal mode.
- `str` – The name of the signal mode.

get_upper_limit(*channel*)

Get the upper limit of a particular channel.

Parameters

- **channel** (`int`) – The channel, either 1 or 2.

Returns

- `float` – The upper limit.

get_valves(*channel*)

Get the state of the valve of a channel.

Parameters

- **channel** (`int`) – The channel, either 1 or 2.

Returns

- `bool` – Whether the valve is enabled or disabled.

identity()

Returns the identity.

Returns

- `str` – The identity (e.g., PR42vvrrssss, where vv is the version, rr is the release and ssss is the serial number).

lock()

Lock setup.

request_key()

Requests most recent key that was pressed.

Returns

- `int` – The key that was most recently pressed.
- `int` – The number of key presses that occurred since the last time this method was called.

reset_status()

Send the reset/status command.

set_access_channel(*channel*, *setpoint*, *valve*)

Set the setpoint and the state of the valve for a particular channel.

Parameters

- **channel** (`int`) – The channel, either 1 or 2.
- **setpoint** (`float`) – The setpoint value.
- **valve** (`bool`) – Whether to enable or disable the valve.

Returns

- `float` – The actual setpoint value.

set_actual_value(*channel*, *setpoint*)

Set the actual value of a particular channel.

Parameters

- **channel** (`int`) – The channel, either 1 or 2.

- **setpoint** (`float`) – The setpoint.

Returns

`float` – The actual value.

set_address(*address*)

Set the address.

Parameters

`address` (`int`) – The address [0, 31].

set_dead_band(*channel*, *band*)

Set the dead band of a particular channel.

Parameters

- `channel` (`int`) – The channel, either 1 or 2.
- `band` (`float`) – The dead band [0.0% to 9.9% of full scale].

set_dialog(*index*)

Set the display dialog.

Parameters

`index` (`int`) – The dialog index (between 0 and 29 inclusive). See Appendix D of the manual for more information.

set_display_text(*text*, *clear=True*)

Set the display text.

To view the text on the display you must call `set_dialog()` with the index equal to 3.

Parameters

- `text` (`str`) – The text to display. Maximum 32 characters.
- `clear` (`bool`, optional) – Whether to clear the current display text before setting the new text.

set_external_input_range(*channel*, *range*)

Set the external input range of a channel.

Parameters

- `channel` (`int`) – The channel, either 1 or 2.
- `range` (`int`) – The external input range [1, 10] in Volts.

set_external_output_range(*channel*, *range*)

Set the external output range of a channel.

Parameters

- `channel` (`int`) – The channel, either 1 or 2.
- `range` (`int`) – The external output range [1, 10] in Volts.

set_formula_relay(*channel*, *formula*)

Set the relay formula of a particular channel.

Parameters

- **channel** (`int`) – The channel, either 1 or 2.
- **formula** (`str`) – The relay formula.

set_formula_temporary(*channel, formula*)

Set the temporary formula of a particular channel.

Parameters

- **channel** (`int`) – The channel, either 1 or 2.
- **formula** (`str`) – The temporary formula.

set_gain(*channel, gain*)

Set the gain of a particular channel.

Parameters

- **channel** (`int`) – The channel, either 1 or 2.
- **gain** (`float`) – The gain [0.001, 2.000].

set_input_range(*channel, range*)

Set the input range of a channel.

Parameters

- **channel** (`int`) – The channel, either 1 or 2.
- **range** (`int`) – The input range [1, 10] in Volts.

set_interface_mode(*mode*)

Set the interface mode.

Parameters

mode (`int`) – The interface mode.

set_limit_mode(*channel, mode*)

Set the limit mode of a channel.

Parameters

- **channel** (`int`) – The channel, either 1 or 2.
- **mode** (`int` or `str`) – The limit mode as either an index number [0, 4] or a name (e.g., SLEEP).

set_linearization_point(*channel, point, x, y*)

Set a point in the linearization table of a particular channel.

Parameters

- **channel** (`int`) – The channel, either 1 or 2.
- **point** (`int`) – The point in the table [0, 10].
- **x** (`float`) – The x value [-5% to 100% of full scale].
- **y** (`float`) – The y value [-5% to 100% of full scale].

set_linearization_size(*channel, size*)

Set the size of the linearization table of a particular channel.

Parameters

- **channel** (`int`) – The channel, either 1 or 2.
- **size** (`int`) – The size of the table.

set_lower_limit(*channel, limit*)

Set the lower limit of a particular channel.

Parameters

- **channel** (`int`) – The channel, either 1 or 2.
- **limit** (`float`) – The lower limit [-5% to 110% of full scale].

set_offset(*channel, offset*)

Set the offset of a particular channel.

Parameters

- **channel** (`int`) – The channel, either 1 or 2.
- **offset** (`int`) – The offset [-250, 250].

set_output_range(*channel, range*)

Set the output range of a channel.

Parameters

- **channel** (`int`) – The channel, either 1 or 2.
- **range** (`int`) – The output range [1, 10] in Volts.

set_range(*channel, range, unit*)

Set the range and unit of a channel.

Parameters

- **channel** (`int`) – The channel, either 1 or 2.
- **range** (`float`) – The range value.
- **unit** (`int` or `str`) – The unit as either an index number [0, 20] or a name (e.g., kPa).

set_relays(*channel, enable*)

Set the relay state of a channel.

Parameters

- **channel** (`int`) – The channel, either 1 or 2.
- **enable** (`bool`) – Whether to enable or disable the relay.

set_remote_mode(*enable*)

Set the remote operation mode to be enable or disabled.

Parameters

- **enable** (`bool`) – Whether to enable or disable remote operation.

set_resolution(*enable*)

Set the 16-bit resolution to be enable or disabled.

Parameters

enable (`bool`) – Whether to enable or disable 16-bit resolution.

set_rtd_offset(*channel*, *offset*)

Set the RTD offset of a particular channel.

Parameters

- **channel** (`int`) – The channel, either 1 or 2.
- **offset** (`int`) – The RTD offset [-250, 250].

set_scale(*channel*, *scale*)

Set the scale of a particular channel.

Parameters

- **channel** (`int`) – The channel, either 1 or 2.
- **scale** (`float`) – The scale.

set_setpoint(*channel*, *setpoint*)

Set the setpoint of a particular channel.

Parameters

- **channel** (`int`) – The channel, either 1 or 2.
- **setpoint** (`float`) – The setpoint.

set_signal_mode(*channel*, *mode*)

Set the range and unit of a channel.

Parameters

- **channel** (`int`) – The channel, either 1 or 2.
- **mode** (`int` or `str`) –

The signal mode as either an index number (e.g., between 0 and 5 inclusive)
or a name (e.g., INDEP).

set_tweak_control(*enable*)

Set tweak control.

Parameters

enable (`bool`) – Whether to switch tweak control on or off.

set_upper_limit(*channel*, *limit*)

Set the upper limit of a particular channel.

Parameters

- **channel** (`int`) – The channel, either 1 or 2.
- **limit** (`float`) – The upper limit [-5% to 110% of full scale].

set_valves(*channel, enable*)

Set the state of the valve of a channel.

Parameters

channel (`int`) – The channel, either 1 or 2.

status()

Request status bits.

Returns

- `int` – The status value.
- `str` – The binary representation of the value.

unlock()

Unlock setup.

msl.equipment.resources.nkt package

Resources for equipment from NKT Photonics.

Submodules

msl.equipment.resources.nkt.nktpdll module

Wrapper around the NKTPDLL.dll SDK from NKT Photonics.

The wrapper was written using v2.1.2.766 of the SDK.

msl.equipment.resources.nkt.nktpdll.PortStatusCallback

Use as a decorator for a callback function when a port status changes.

msl.equipment.resources.nkt.nktpdll.DeviceStatusCallback

Use as a decorator for a callback function when a device status changes.

msl.equipment.resources.nkt.nktpdll.RegisterStatusCallback

Use as a decorator for a callback function when a register status changes.

class msl.equipment.resources.nkt.nktpdll.DateTimeType

Bases: Structure

The DateTimeType struct (24 hour format).

Day

Structure/Union member

Hour

Structure/Union member

Min

Structure/Union member

Month

Structure/Union member

Sec

Structure/Union member

Year

Structure/Union member

```
class msl.equipment.resources.nkt.nktpdll.ParameterSetType
```

Bases: Structure

The ParameterSet struct.

This is how calculation on parameter sets is done internally by modules:

DAC_value = (value * (X/Y)) + Offset

where, value is either ParameterSetType::StartVal or ParameterSetType::FactoryVal

value = (ADC_value * (X/Y)) + Offset

where, value often is available via another measurement register.

Denominator

Structure/Union member

ErrorHandler

Structure/Union member

FactoryVal

Structure/Union member

LLimit

Structure/Union member

Numerator

Structure/Union member

Offset

Structure/Union member

StartVal

Structure/Union member

ULimit

Structure/Union member

Unit

Structure/Union member

```
class msl.equipment.resources.nkt.nktpdll.DeviceModeTypes(value, names=None,  
                                         *values, module=None,  
                                         qualname=None,  
                                         type=None, start=1,  
                                         boundary=None)
```

Bases: IntEnum

The DeviceModeTypes enum.

DevModeDisabled = 0

```
DevModeAnalyzeInit = 1
DevModeAnalyze = 2
DevModeNormal = 3
DevModeLogDownload = 4
DevModeError = 5
DevModeTimeout = 6
DevModeUpload = 7

class msl.equipment.resources.nkt.nktpdll.DeviceStatusTypes(value, names=None,
                                                               *values, module=None,
                                                               qualname=None,
                                                               type=None, start=1,
                                                               boundary=None)
```

Bases: [IntEnum](#)

The DeviceStatusTypes enum.

```
DeviceModeChanged = 0
DeviceLiveChanged = 1
DeviceTypeChanged = 2
DevicePartNumberChanged = 3
DevicePCBVersionChanged = 4
DeviceStatusBitsChanged = 5
DeviceErrorCodeChanged = 6
DeviceBlVerChanged = 7
DeviceFwVerChanged = 8
DeviceModuleSerialChanged = 9
DevicePCBSerialChanged = 10
DeviceSysTypeChanged = 11
```

```
class msl.equipment.resources.nkt.nktpdll.ParamSetUnitTypes(value, names=None,
                                                               *values, module=None,
                                                               qualname=None,
                                                               type=None, start=1,
                                                               boundary=None)
```

Bases: [IntEnum](#)

The ParamSetUnitTypes enum

```
UnitNone = 0
```

UnitmV = 1
UnitV = 2
UnituA = 3
UnitmA = 4
UnitA = 5
UnituW = 6
UnitcmW = 7
UnitdmW = 8
UnitmW = 9
UnitW = 10
UnitmC = 11
UnitcC = 12
UnitdC = 13
Unitpm = 14
Unitdnm = 15
Unitnm = 16
UnitPerCent = 17
UnitPerMille = 18
UnitcmA = 19
UnitdmA = 20
UnitRPM = 21
UnitdBm = 22
UnitcBm = 23
UnitmBm = 24
UnitdB = 25
UnitcB = 26
UnitmB = 27
Unitdpm = 28
UnitcV = 29
UnitdV = 30

Unitlm = 31

Unitdlm = 32

Unitclm = 33

Unitmlm = 34

```
class msl.equipment.resources.nkt.nktpdll.RegisterPriorityTypes(value,
                                                               names=None,
                                                               *values,
                                                               module=None,
                                                               qualname=None,
                                                               type=None,
                                                               start=1,
                                                               boundary=None)
```

Bases: [IntEnum](#)

The RegisterPriorityTypes enum.

RegPriority_Low = 0

RegPriority_High = 1

```
class msl.equipment.resources.nkt.nktpdll.RegisterDataTypes(value, names=None,
                                                               *values, module=None,
                                                               qualname=None,
                                                               type=None, start=1,
                                                               boundary=None)
```

Bases: [IntEnum](#)

The RegisterDataTypes enum.

RegData_Unknown = 0

RegData_Mixed = 1

RegData_U8 = 2

RegData_S8 = 3

RegData_U16 = 4

RegData_S16 = 5

RegData_U32 = 6

RegData_S32 = 7

RegData_F32 = 8

RegData_U64 = 9

RegData_S64 = 10

RegData_F64 = 11

```
RegData_Ascii = 12
RegData_Paramset = 13
RegData_B8 = 14
RegData_H8 = 15
RegData_B16 = 16
RegData_H16 = 17
RegData_B32 = 18
RegData_H32 = 19
RegData_B64 = 20
RegData_H64 = 21
RegData_DateTime = 22
```

```
class msl.equipment.resources.nkt.nktpdll.RegisterStatusTypes(value, names=None,
                                                               *values,
                                                               module=None,
                                                               qualname=None,
                                                               type=None, start=1,
                                                               boundary=None)
```

Bases: [IntEnum](#)

The RegisterStatusTypes enum.

```
RegSuccess = 0
RegBusy = 1
RegNacked = 2
RegCRCErr = 3
RegTimeout = 4
RegComError = 5
```

```
class msl.equipment.resources.nkt.nktpdll.PortStatusTypes(value, names=None,
                                                               *values, module=None,
                                                               qualname=None,
                                                               type=None, start=1,
                                                               boundary=None)
```

Bases: [IntEnum](#)

The PortStatusTypes enum

```
PortStatusUnknown = 0
PortOpening = 1
```

```
PortOpened = 2
PortOpenFail = 3
PortScanStarted = 4
PortScanProgress = 5
PortScanDeviceFound = 6
PortScanEnded = 7
PortClosing = 8
PortClosed = 9
PortReady = 10
```

```
class msl.equipment.resources.nkt.nktpd11.NKT(record)
```

Bases: [Connection](#)

Wrapper around the NKTPDLL.dll SDK from NKT Photonics.

The [properties](#) for a NKT connection supports the following key-value pairs in the [Connections Database](#):

```
'sdk_path': str, The path to the SDK [default: 'NKTPDLL.dll']
'open_port': bool, Whether to automatically open the port [default: True]
'auto': bool, Whether to open the port with bus scanning [default: True]
'live': bool, Whether to open the port in live mode [default: True]
```

Do not instantiate this class directly. Use the [connect\(\)](#) method to connect to the equipment.

Parameters

record ([EquipmentRecord](#)) – A record from an [Equipment-Register Database](#).

RegisterStatusCallback

alias of [CFunctionType](#)

PortStatusCallback

alias of [CFunctionType](#)

DeviceStatusCallback

alias of [CFunctionType](#)

```
class DeviceModeTypes(value, names=None, *values, module=None, qualname=None,
type=None, start=1, boundary=None)
```

Bases: [IntEnum](#)

The DeviceModeTypes enum.

DevModeDisabled = 0

DevModeAnalyzeInit = 1

DevModeAnalyze = 2

```
    DevModeNormal = 3
    DevModeLogDownload = 4
    DevModeError = 5
    DevModeTimeout = 6
    DevModeUpload = 7

class DeviceStatusTypes(value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None)
    Bases: IntEnum

    The DeviceStatusTypes enum.

    DeviceModeChanged = 0
    DeviceLiveChanged = 1
    DeviceTypeChanged = 2
    DevicePartNumberChanged = 3
    DevicePCBVersionChanged = 4
    DeviceStatusBitsChanged = 5
    DeviceErrorCodeChanged = 6
    DeviceBlVerChanged = 7
    DeviceFwVerChanged = 8
    DeviceModuleSerialChanged = 9
    DevicePCBSerialChanged = 10
    DeviceSysTypeChanged = 11

class ParamSetUnitTypes(value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None)
    Bases: IntEnum

    The ParamSetUnitTypes enum

    UnitNone = 0
    UnitmV = 1
    UnitV = 2
    UnituA = 3
    UnitmA = 4
    UnitA = 5
```

UnituW = 6
UnitcmW = 7
UnitdmW = 8
UnitmW = 9
UnitW = 10
UnitmC = 11
UnitcC = 12
UnitdC = 13
Unitpm = 14
Unitdnm = 15
Unitnm = 16
UnitPerCent = 17
UnitPerMille = 18
UnitcmA = 19
UnitdmA = 20
UnitRPM = 21
UnitdBm = 22
UnitcBm = 23
UnitmBm = 24
UnitdB = 25
UnitcB = 26
UnitmB = 27
Unitdpm = 28
UnitcV = 29
UnitdV = 30
Unitlm = 31
Unitdlm = 32
Unitclm = 33
Unitmlm = 34

```
class RegisterPriorityTypes(value, names=None, *values, module=None,
                           qualname=None, type=None, start=1, boundary=None)
```

Bases: [IntEnum](#)

The RegisterPriorityTypes enum.

```
RegPriority_Low = 0
```

```
RegPriority_High = 1
```

```
class RegisterDataTypes(value, names=None, *values, module=None, qualname=None,
                        type=None, start=1, boundary=None)
```

Bases: [IntEnum](#)

The RegisterDataTypes enum.

```
RegData_Unknown = 0
```

```
RegData_Mixed = 1
```

```
RegData_U8 = 2
```

```
RegData_S8 = 3
```

```
RegData_U16 = 4
```

```
RegData_S16 = 5
```

```
RegData_U32 = 6
```

```
RegData_S32 = 7
```

```
RegData_F32 = 8
```

```
RegData_U64 = 9
```

```
RegData_S64 = 10
```

```
RegData_F64 = 11
```

```
RegData_Ascii = 12
```

```
RegData_Paramset = 13
```

```
RegData_B8 = 14
```

```
RegData_H8 = 15
```

```
RegData_B16 = 16
```

```
RegData_H16 = 17
```

```
RegData_B32 = 18
```

```
RegData_H32 = 19
```

```
RegData_B64 = 20
```

```
RegData_H64 = 21

RegData_DateTime = 22

class RegisterStatusTypes(value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None)
    Bases: IntEnum

    The RegisterStatusTypes enum.

    RegSuccess = 0

    RegBusy = 1

    RegNacked = 2

    RegCRCErr = 3

    RegTimeout = 4

    RegComError = 5

class PortStatusTypes(value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None)
    Bases: IntEnum

    The PortStatusTypes enum

    PortStatusUnknown = 0

    PortOpening = 1

    PortOpened = 2

    PortOpenFail = 3

    PortScanStarted = 4

    PortScanProgress = 5

    PortScanDeviceFound = 6

    PortScanEnded = 7

    PortClosing = 8

    PortClosed = 9

    PortReady = 10

static close_ports(names=None)
    Close the specified port name(s).

Parameters
    names (str, list of str, optional) – If None then close all opened ports. If a str then the name of a port. Otherwise a list of names. Port names are case sensitive.
```

Raises

NKTError – If there was an error calling this method.

static load_sdk(path=None)

Load the SDK.

Parameters

path (`str`, optional) – The path to `NKTPDLL.dll`. If not specified then searches for the library.

static device_get_all_types(opened_ports=None, size=255)

Returns all device types (module types) from the internal device list.

Parameters

- **opened_ports** (`str` or `list` of `str`, optional) – A port or a list of opened ports. If not specified then the `get_open_ports()` method will be called and the types for each port will be returned.
- **size** (`int`, optional) – The maximum number of bytes that the device list can be.

Returns

`dict` – The port names are the keys and each value is `dict` with the module type as the keys and its corresponding device ID as the value.

Raises

NKTError – If there was an error calling this method.

device_create(device_id, wait_ready)

Creates a device in the internal device list.

If the `open_ports()` function has been called with `live = 1` then the kernel immediately starts to monitor the device.

Parameters

- **device_id** (`int`) – The device id (module address).
- **wait_ready** (`bool`) – `False` - Don't wait for the device to be ready.
`True` - Wait up to 2 seconds for the device to complete its analyze cycle.
(All standard registers being successfully read)

Raises

NKTError – If there was an error calling this method.

device_exists(device_id)

Checks if a specific device already exists in the internal device list.

Parameters

device_id (`int`) – The device id (module address).

Returns

`bool` – Whether the device exists.

Raises

NKTError – If there was an error calling this method.

device_get_boot_loader_version(*device_id*)

Returns the boot-loader version (int) for a given device id.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated read.

Parameters

device_id (`int`) – The device id (module address).

Returns

`int` – The boot-loader version.

Raises

`NKTError` – If there was an error calling this method.

device_get_boot_loader_version_str(*device_id*)

Returns the boot-loader version (string) for a given device id.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated read.

Parameters

device_id (`int`) – The device id (module address).

Returns

`str` – The boot-loader version.

Raises

`NKTError` – If there was an error calling this method.

device_get_error_code(*device_id*)

Returns the error code for a given device id.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated read.

Parameters

device_id (`int`) – The device id (module address).

Returns

`int` – The error code.

Raises

`NKTError` – If there was an error calling this method.

device_get_firmware_version(*device_id*)

Returns the firmware version (int) for a given device id.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated read.

Parameters

device_id (`int`) – The device id (module address).

Returns

`int` – The firmware version.

Raises

`NKTError` – If there was an error calling this method.

device_get_firmware_version_str(device_id)

Returns the firmware version (string) for a given device id.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated read.

Parameters

device_id (`int`) – The device id (module address).

Returns

`str` – The firmware version.

Raises

`NKTError` – If there was an error calling this method.

device_get_live(device_id)

Returns the internal device live status for a specific device id.

Requires the port being already opened with the `open_ports()` function and the device being already created, either automatically or with the `device_create()` function.

Parameters

device_id (`int`) – The device id (module address).

Returns

`bool` – Whether live mode is enabled.

Raises

`NKTError` – If there was an error calling this method.

device_get_mode(device_id)

Returns the internal device mode for a specific device id.

Requires the port being already opened with the `open_ports()` function and the device being already created, either automatically or with the `device_create()` function.

Parameters

device_id (`int`) – The device id (module address).

Returns

`DeviceModeTypes` – The device mode type.

Raises

`NKTError` – If there was an error calling this method.

device_get_module_serial_number_str(device_id)

Returns the module serial number (string) for a given device id.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated read.

Parameters

device_id (`int`) – The device id (module address).

Returns

`str` – The serial number.

Raises

`NKTError` – If there was an error calling this method.

device_get_part_number_str(device_id)

Returns the part number for a given device id.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated read.

Parameters

device_id (`int`) – The device id (module address).

Returns

`str` – The part number.

Raises

`NKTError` – If there was an error calling this method.

device_get_pcb_serial_number_str(device_id)

Returns the PCB serial number (string) for a given device id.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated read.

Parameters

device_id (`int`) – The device id (module address).

Returns

`str` – The part number.

Raises

`NKTError` – If there was an error calling this method.

device_get_pcb_version(device_id)

Returns the PCB version for a given device id.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated read.

Parameters

device_id (`int`) – The device id (module address).

Returns

`int` – The PCB version number.

Raises

`NKTError` – If there was an error calling this method.

device_get_status_bits(device_id)

Returns the status bits for a given device id.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated read.

Parameters

device_id (`int`) – The device id (module address).

Returns

`int` – The status bits.

Raises

`NKTError` – If there was an error calling this method.

device_get_type(*device_id*)

Returns the module type for a specific device id.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated read.

Parameters

device_id (`int`) – The device id (module address).

Returns

`int` – The module type.

Raises

`NKTError` – If there was an error calling this method.

device_remove(*device_id*)

Remove a specific device from the internal device list.

Parameters

device_id (`int`) – The device id (module address).

Raises

`NKTError` – If there was an error calling this method.

device_remove_all()

Remove all devices from the internal device list.

No confirmation is given, the list is simply cleared.

Raises

`NKTError` – If there was an error calling this method.

device_set_live(*device_id*, *live_mode*)

Sets the internal device live status for a specific device id (module address).

Parameters

- **device_id** (`int`) – The device id (module address).
- **live_mode** (`bool`) – Set to `True` to enable.

Raises

`NKTError` – If there was an error calling this method.

disconnect()

Disconnect from the port.

Raises

`NKTError` – If there was an error calling this method.

static get_all_ports(*size*=255)

Returns a list of all ports.

Parameters

size (`int`, optional) – The maximum size of the string buffer to fetch the results.

Returns

`list` of `str` – A list of port names.

get_modules(size=255)

Returns all device types (module types) from the device.

Parameters

size (`int`, optional) – The maximum number of bytes that the device list can be.

Returns

`dict` – The module type as the keys and its corresponding device ID as the value.

Raises

`NKTError` – If there was an error calling this method.

static get_legacy_bus_scanning()

Get the bus-scanning mode.

Returns

`bool` – `True` if in legacy mode otherwise in normal mode.

static get_open_ports(size=255)

Returns a list of already-opened ports.

Parameters

size (`int`, optional) – The maximum size of the string buffer to fetch the results.

Returns

`list` of `str` – A list of port names that are already open.

get_port_error_msg()

Retrieve error message for the port.

Returns

`str` – The error message. An empty string indicates no error.

Raises

`NKTError` – If there was an error calling this method.

get_port_status()

Get the status of the port.

Returns

`PortStatusTypes` – The port status.

Raises

`NKTError` – If there was an error calling this method.

static open_ports(names=None, auto=True, live=True)

Open the specified port(s).

Repeated calls to this function is allowed to reopen and/or rescan for devices.

Parameters

- **names** (`str`, `list` of `str`, optional) – If `None` then open all available ports. If a `str` then the name of a port. Otherwise a `list` of names. Port names are case sensitive. Example port names are '`'AcoustikPort1'`', '`'COM6'`'.

- **auto** (`bool`, optional) – If `True` then automatically start bus scanning and add the found devices in the internal device list. If `False` then bus scanning and device creation is not automatically handled. The port is automatically closed if no devices are found.
- **live** (`bool`, optional) – If `True` then keep all the found or created devices in live mode, which means the Interbus kernel keeps monitoring all the found devices and their registers. Please note that this will keep the modules watchdog alive as long as the port is open. If `False` then disable continuous monitoring of the registers. No callback is possible on register changes. Use the `register_read()`, `register_write()` and `register_write_read()` methods.

Raises

`NKTError` – If there was an error calling this method.

`point_to_point_port_add(host_address, host_port, client_address, client_port, protocol, ms_timeout=100)`

Creates or modifies a point to point port.

Parameters

- **host_address** (`str`) – The local ip address, e.g., '192.168.1.67'.
- **host_port** (`int`) – The local port number.
- **client_address** (`str`) – The remote ip address, e.g., '192.168.1.100'.
- **client_port** (`int`) – The remote port number.
- **protocol** (`int`) – Either 0 (TCP) or 1 (UDP).
- **ms_timeout** (`int`, optional) – Telegram timeout value in milliseconds.

Raises

`NKTError` – If there was an error calling this method.

`point_to_point_port_del()`

Delete the point-to-point port.

Raises

`NKTError` – If there was an error calling this method.

`point_to_point_port_get()`

Retrieve the information about the point-to-point port setting.

Returns

`dict` – The information about the point-to-point port setting.

Raises

`NKTError` – If there was an error calling this method.

`register_read(device_id, reg_id, index=-1)`

Reads a register value and returns the result.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated read.

Parameters

- **device_id** (`int`) – The device id (module address).
- **reg_id** (`int`) – The register id (register address).
- **index** (`int`, optional) – Value index. Typically -1, but could be used to extract data from a specific position in the register. Index is byte counted.

Returns

`bytes` – The register value.

Raises

`NKTError` – If there was an error calling this method.

register_create(*device_id*, *reg_id*, *priority*, *data_type*)

Creates a register in the internal register list.

If the `open_ports()` function has been called with the *live* = 1 then the kernel immediately starts to monitor the register.

Parameters

- **device_id** (`int`) – The device id (module address).
- **reg_id** (`int`) – The register id (register address).
- **priority** (`int`) – The `RegisterPriorityTypes` (monitoring priority).
- **data_type** (`int`) – The `RegisterDataTypes`, not used internally but could be used in a common callback function to determine data type.

Raises

`NKTError` – If there was an error calling this method.

register_exists(*device_id*, *reg_id*)

Checks if a specific register already exists in the internal register list.

Parameters

- **device_id** (`int`) – The device id (module address).
- **reg_id** (`int`) – The register id (register address).

Returns

`bool` – Whether the register exists.

Raises

`NKTError` – If there was an error calling this method.

register_get_all(*device_id*)

Returns the register ids (register addresses) from the internal register list.

Parameters

`device_id` (`int`) – The device id (module address).

Returns

`list` of `int` – The register ids.

Raises

`NKTError` – If there was an error calling this method.

register_read_ascii(device_id, reg_id, index=-1)

Reads an ascii string from the register.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated read.

Parameters

- **device_id** (`int`) – The device id (module address).
- **reg_id** (`int`) – The register id (register address).
- **index** (`int`, optional) – Value index. Typically -1, but could be used to extract data from a specific position in the register. Index is byte counted.

Returns

`str` – The ascii value.

Raises

`NKTError` – If there was an error calling this method.

register_read_f32(device_id, reg_id, index=-1)

Reads 32-bit float value from the register.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated read.

Parameters

- **device_id** (`int`) – The device id (module address).
- **reg_id** (`int`) – The register id (register address).
- **index** (`int`, optional) – Value index. Typically -1, but could be used to extract data from a specific position in the register. Index is byte counted.

Returns

`float` – The 32-bit float value.

Raises

`NKTError` – If there was an error calling this method.

register_read_f64(device_id, reg_id, index=-1)

Reads 64-bit double value from the register.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated read.

Parameters

- **device_id** (`int`) – The device id (module address).
- **reg_id** (`int`) – The register id (register address).
- **index** (`int`, optional) – Value index. Typically -1, but could be used to extract data from a specific position in the register. Index is byte counted.

Returns

`float` – The 64-bit double value.

Raises

`NKTError` – If there was an error calling this method.

register_read_s16(device_id, reg_id, index=-1)

Reads 16-bit signed short value from the register.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated read.

Parameters

- **device_id** (`int`) – The device id (module address).
- **reg_id** (`int`) – The register id (register address).
- **index** (`int`, optional) – Value index. Typically -1, but could be used to extract data from a specific position in the register. Index is byte counted.

Returns

`int` – The 16-bit signed short value.

Raises

`NKTError` – If there was an error calling this method.

register_read_s32(device_id, reg_id, index=-1)

Reads 32-bit signed long value from the register.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated read.

Parameters

- **device_id** (`int`) – The device id (module address).
- **reg_id** (`int`) – The register id (register address).
- **index** (`int`, optional) – Value index. Typically -1, but could be used to extract data from a specific position in the register. Index is byte counted.

Returns

`int` – The 32-bit signed long value.

Raises

`NKTError` – If there was an error calling this method.

register_read_s64(device_id, reg_id, index=-1)

Reads 64-bit signed long long value from the register.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated read.

Parameters

- **device_id** (`int`) – The device id (module address).
- **reg_id** (`int`) – The register id (register address).
- **index** (`int`, optional) – Value index. Typically -1, but could be used to extract data from a specific position in the register. Index is byte counted.

Returns

`int` – The 64-bit signed long long value.

Raises

`NKTError` – If there was an error calling this method.

register_read_s8(device_id, reg_id, index=-1)

Reads 8-bit signed char value from the register.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated read.

Parameters

- **device_id** (`int`) – The device id (module address).
- **reg_id** (`int`) – The register id (register address).
- **index** (`int`, optional) – Value index. Typically -1, but could be used to extract data from a specific position in the register. Index is byte counted.

Returns

`int` – The 8-bit signed char value.

Raises

`NKTError` – If there was an error calling this method.

register_read_u16(device_id, reg_id, index=-1)

Reads 16-bit unsigned short value from the register.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated read.

Parameters

- **device_id** (`int`) – The device id (module address).
- **reg_id** (`int`) – The register id (register address).
- **index** (`int`, optional) – Value index. Typically -1, but could be used to extract data from a specific position in the register. Index is byte counted.

Returns

`int` – The 16-bit unsigned short value.

Raises

`NKTError` – If there was an error calling this method.

register_read_u32(device_id, reg_id, index=-1)

Reads 32-bit unsigned long value from the register.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated read.

Parameters

- **device_id** (`int`) – The device id (module address).
- **reg_id** (`int`) – The register id (register address).
- **index** (`int`, optional) – Value index. Typically -1, but could be used to extract data from a specific position in the register. Index is byte counted.

Returns

`int` – The 32-bit unsigned long value.

Raises

`NKTError` – If there was an error calling this method.

register_read_u64(device_id, reg_id, index=-1)

Reads 64-bit unsigned long long value from the register.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated read.

Parameters

- **device_id** (`int`) – The device id (module address).
- **reg_id** (`int`) – The register id (register address).
- **index** (`int`, optional) – Value index. Typically -1, but could be used to extract data from a specific position in the register. Index is byte counted.

Returns

`int` – The 64-bit unsigned long long value.

Raises

`NKTError` – If there was an error calling this method.

register_read_u8(device_id, reg_id, index=-1)

Reads 8-bit unsigned char value from the register.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated read.

Parameters

- **device_id** (`int`) – The device id (module address).
- **reg_id** (`int`) – The register id (register address).
- **index** (`int`, optional) – Value index. Typically -1, but could be used to extract data from a specific position in the register. Index is byte counted.

Returns

`int` – The 8-bit unsigned char value.

Raises

`NKTError` – If there was an error calling this method.

register_remove(device_id, reg_id)

Remove a specific register from the internal register list.

Parameters

- **device_id** (`int`) – The device id (module address).
- **reg_id** (`int`) – The register id (register address).

Raises

`NKTError` – If there was an error calling this method.

register_remove_all(device_id)

Remove all registers from the internal register list.

No confirmation given, the list is simply cleared.

Parameters

device_id (`int`) – The device id (module address).

Raises

NKTError – If there was an error calling this method.

register_write(*device_id*, *reg_id*, *data*, *index*=-1)

Writes a register value.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated write.

Parameters

- **device_id** (`int`) – The device id (module address).
- **reg_id** (`int`) – The register id (register address).
- **data** (`bytes`) – The data to write to the register.
- **index** (`int`, optional) – Value index. Typically -1, but could be used to write a value in a multi-value register.

Raises

NKTError – If there was an error calling this method.

register_write_ascii(*device_id*, *reg_id*, *string*, *write_eol*, *index*=-1)

Writes a string to the register value.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated write.

Parameters

- **device_id** (`int`) – The device id (module address).
- **reg_id** (`int`) – The register id (register address).
- **string** (`str`) – The string to write to the register.
- **write_eol** (`bool`) – Whether to append the End Of Line character (a null character) to the string.
- **index** (`int`, optional) – Value index. Typically -1, but could be used to write a value in a mixed-type register.

Raises

NKTError – If there was an error calling this method.

register_write_f32(*device_id*, *reg_id*, *value*, *index*=-1)

Writes a 32-bit float register value.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated write.

Parameters

- **device_id** (`int`) – The device id (module address).
- **reg_id** (`int`) – The register id (register address).
- **value** (`float`) – The 32-bit float to write to the register.
- **index** (`int`, optional) – Value index. Typically -1, but could be used to write a value in a multi-value register.

Raises

NKTError – If there was an error calling this method.

register_write_f64(device_id, reg_id, value, index=-1)

Writes a 64-bit double register value.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated write.

Parameters

- **device_id** (`int`) – The device id (module address).
- **reg_id** (`int`) – The register id (register address).
- **value** (`float`) – The 64-bit double to write to the register.
- **index** (`int`, optional) – Value index. Typically -1, but could be used to write a value in a multi-value register.

Raises

NKTError – If there was an error calling this method.

register_write_read(device_id, reg_id, data, index=-1)

Writes then reads a register value.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated write followed by a dedicated read.

Parameters

- **device_id** (`int`) – The device id (module address).
- **reg_id** (`int`) – The register id (register address).
- **data** (`bytes`) – The data to write to the register.
- **index** (`int`, optional) – Value index. Typically -1, but could be used to write a value in a multi-value register.

Returns

`bytes` – The data that was written to the register.

Raises

NKTError – If there was an error calling this method.

register_write_read_ascii(device_id, reg_id, string, write_eol, index=-1)

Writes then reads a string register value.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated write followed by a dedicated read.

Parameters

- **device_id** (`int`) – The device id (module address).
- **reg_id** (`int`) – The register id (register address).
- **string** (`str`) – The string to write to the register.
- **write_eol** (`bool`) – Whether to append the End Of Line character (a null character) to the string.

- **index** (`int`, optional) – Value index. Typically -1, but could be used to write a value in a multi-value register.

Returns

`str` – The string that was written to the register.

Raises

`NKTError` – If there was an error calling this method.

register_write_read_f32(*device_id*, *reg_id*, *value*, *index*=-1)

Writes then reads a 32-bit float register value.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated write followed by a dedicated read.

Parameters

- **device_id** (`int`) – The device id (module address).
- **reg_id** (`int`) – The register id (register address).
- **value** (`float`) – The 32-bit float value to write to the register.
- **index** (`int`, optional) – Value index. Typically -1, but could be used to write a value in a multi-value register.

Returns

`float` – The 32-bit float value that was written to the register.

Raises

`NKTError` – If there was an error calling this method.

register_write_read_f64(*device_id*, *reg_id*, *value*, *index*=-1)

Writes then reads a 64-bit double register value.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated write followed by a dedicated read.

Parameters

- **device_id** (`int`) – The device id (module address).
- **reg_id** (`int`) – The register id (register address).
- **value** (`float`) – The 64-bit double value to write to the register.
- **index** (`int`, optional) – Value index. Typically -1, but could be used to write a value in a multi-value register.

Returns

`float` – The 64-bit double value that was written to the register.

Raises

`NKTError` – If there was an error calling this method.

register_write_read_s16(*device_id*, *reg_id*, *value*, *index*=-1)

Writes then reads a 16-bit signed short register value.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated write followed by a dedicated read.

Parameters

- **device_id** (`int`) – The device id (module address).
- **reg_id** (`int`) – The register id (register address).
- **value** (`int`) – The 16-bit signed short value to write to the register.
- **index** (`int`, optional) – Value index. Typically -1, but could be used to write a value in a multi-value register.

Returns

`int` – The 16-bit signed short value that was written to the register.

Raises

`NKTErr` – If there was an error calling this method.

register_write_read_s32(*device_id*, *reg_id*, *value*, *index*=-1)

Writes then reads a 32-bit signed long register value.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated write followed by a dedicated read.

Parameters

- **device_id** (`int`) – The device id (module address).
- **reg_id** (`int`) – The register id (register address).
- **value** (`int`) – The 32-bit signed long value to write to the register.
- **index** (`int`, optional) – Value index. Typically -1, but could be used to write a value in a multi-value register.

Returns

`int` – The 32-bit signed long value that was written to the register.

Raises

`NKTErr` – If there was an error calling this method.

register_write_read_s64(*device_id*, *reg_id*, *value*, *index*=-1)

Writes then reads a 64-bit signed long long register value.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated write followed by a dedicated read.

Parameters

- **device_id** (`int`) – The device id (module address).
- **reg_id** (`int`) – The register id (register address).
- **value** (`int`) – The 64-bit signed long long value to write to the register.
- **index** (`int`, optional) – Value index. Typically -1, but could be used to write a value in a multi-value register.

Returns

`int` – The 64-bit signed long long value that was written to the register.

Raises

`NKTErr` – If there was an error calling this method.

register_write_read_s8(device_id, reg_id, value, index=-1)

Writes then reads a 8-bit signed char register value.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated write followed by a dedicated read.

Parameters

- **device_id** (`int`) – The device id (module address).
- **reg_id** (`int`) – The register id (register address).
- **value** (`int`) – The 8-bit signed char value to write to the register.
- **index** (`int`, optional) – Value index. Typically -1, but could be used to write a value in a multi-value register.

Returns

`int` – The 8-bit signed char value that was written to the register.

Raises

`NKTError` – If there was an error calling this method.

register_write_read_u16(device_id, reg_id, value, index=-1)

Writes then reads a 16-bit unsigned short register value.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated write followed by a dedicated read.

Parameters

- **device_id** (`int`) – The device id (module address).
- **reg_id** (`int`) – The register id (register address).
- **value** (`int`) – The 16-bit unsigned short value to write to the register.
- **index** (`int`, optional) – Value index. Typically -1, but could be used to write a value in a multi-value register.

Returns

`int` – The 16-bit unsigned short value that was written to the register.

Raises

`NKTError` – If there was an error calling this method.

register_write_read_u32(device_id, reg_id, value, index=-1)

Writes then reads a 32-bit unsigned long register value.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated write followed by a dedicated read.

Parameters

- **device_id** (`int`) – The device id (module address).
- **reg_id** (`int`) – The register id (register address).
- **value** (`int`) – The 32-bit unsigned long value to write to the register.
- **index** (`int`, optional) – Value index. Typically -1, but could be used to write a value in a multi-value register.

Returns

`int` – The 32-bit unsigned long value that was written to the register.

Raises

`NKTError` – If there was an error calling this method.

register_write_read_u64(device_id, reg_id, value, index=-1)

Writes then reads a 64-bit unsigned long long register value.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated write followed by a dedicated read.

Parameters

- `device_id (int)` – The device id (module address).
- `reg_id (int)` – The register id (register address).
- `value (int)` – The 64-bit unsigned long long value to write to the register.
- `index (int, optional)` – Value index. Typically -1, but could be used to write a value in a multi-value register.

Returns

`int` – The 64-bit unsigned long long value that was written to the register.

Raises

`NKTError` – If there was an error calling this method.

register_write_read_u8(device_id, reg_id, value, index=-1)

Writes then reads a 8-bit unsigned char register value.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated write followed by a dedicated read.

Parameters

- `device_id (int)` – The device id (module address).
- `reg_id (int)` – The register id (register address).
- `value (int)` – The 8-bit unsigned char value to write to the register.
- `index (int, optional)` – Value index. Typically -1, but could be used to write a value in a multi-value register.

Returns

`int` – The 8-bit unsigned char value that was written to the register.

Raises

`NKTError` – If there was an error calling this method.

register_write_s16(device_id, reg_id, value, index=-1)

Writes a 16-bit signed short register value.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated write.

Parameters

- `device_id (int)` – The device id (module address).

- **reg_id** (`int`) – The register id (register address).
- **value** (`int`) – The 16-bit signed short to write to the register.
- **index** (`int`, optional) – Value index. Typically -1, but could be used to write a value in a multi-value register.

Raises

`NKTError` – If there was an error calling this method.

register_write_s32(*device_id*, *reg_id*, *value*, *index*=-1)

Writes a 32-bit signed long register value.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated write.

Parameters

- **device_id** (`int`) – The device id (module address).
- **reg_id** (`int`) – The register id (register address).
- **value** (`int`) – The 32-bit signed long to write to the register.
- **index** (`int`, optional) – Value index. Typically -1, but could be used to write a value in a multi-value register.

Raises

`NKTError` – If there was an error calling this method.

register_write_s64(*device_id*, *reg_id*, *value*, *index*=-1)

Writes a 64-bit signed long long register value.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated write.

Parameters

- **device_id** (`int`) – The device id (module address).
- **reg_id** (`int`) – The register id (register address).
- **value** (`int`) – The 64-bit signed long long to write to the register.
- **index** (`int`, optional) – Value index. Typically -1, but could be used to write a value in a multi-value register.

Raises

`NKTError` – If there was an error calling this method.

register_write_s8(*device_id*, *reg_id*, *value*, *index*=-1)

Writes a 8-bit signed char register value.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated write.

Parameters

- **device_id** (`int`) – The device id (module address).
- **reg_id** (`int`) – The register id (register address).
- **value** (`int`) – The 8-bit signed char to write to the register.

- **index** (`int`, optional) – Value index. Typically -1, but could be used to write a value in a multi-value register.

Raises

`NKTError` – If there was an error calling this method.

register_write_u16(*device_id*, *reg_id*, *value*, *index*=-1)

Writes a 16-bit unsigned short register value.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated write.

Parameters

- **device_id** (`int`) – The device id (module address).
- **reg_id** (`int`) – The register id (register address).
- **value** (`int`) – The 16-bit unsigned short to write to the register.
- **index** (`int`, optional) – Value index. Typically -1, but could be used to write a value in a multi-value register.

Raises

`NKTError` – If there was an error calling this method.

register_write_u32(*device_id*, *reg_id*, *value*, *index*=-1)

Writes a 32-bit unsigned long register value.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated write.

Parameters

- **device_id** (`int`) – The device id (module address).
- **reg_id** (`int`) – The register id (register address).
- **value** (`int`) – The 32-bit unsigned long to write to the register.
- **index** (`int`, optional) – Value index. Typically -1, but could be used to write a value in a multi-value register.

Raises

`NKTError` – If there was an error calling this method.

register_write_u64(*device_id*, *reg_id*, *value*, *index*=-1)

Writes a 64-bit unsigned long long register value.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated write.

Parameters

- **device_id** (`int`) – The device id (module address).
- **reg_id** (`int`) – The register id (register address).
- **value** (`int`) – The 64-bit unsigned long long to write to the register.
- **index** (`int`, optional) – Value index. Typically -1, but could be used to write a value in a multi-value register.

Raises

NKTError – If there was an error calling this method.

register_write_u8(*device_id*, *reg_id*, *value*, *index*=-1)

Writes a 8-bit unsigned char register value.

It is not necessary to open the port, create the device or register before using this function, since it will do a dedicated write.

Parameters

- **device_id** (*int*) – The device id (module address).
- **reg_id** (*int*) – The register id (register address).
- **value** (*int*) – The 8-bit unsigned char to write to the register.
- **index** (*int*, optional) – Value index. Typically -1, but could be used to write a value in a multi-value register.

Raises

NKTError – If there was an error calling this method.

static set_legacy_bus_scanning(*mode*)

Set the bus-scanning mode to normal or legacy.

Parameters

mode (*bool*) – If *False* then bus scanning is set to normal mode and allows for a rolling masterId. In this mode the masterId is changed for each message to allow for out-of-sync detection. If *True* then bus scanning is set to legacy mode and fixes the masterId at address 66(0x42). Some older modules do not accept masterIds other than 66(0x42).

static set_callback_device_status(*callback*)

Enables/Disables a callback for device status changes.

Parameters

callback (*DeviceStatusCallback*) – A *DeviceStatusCallback* object. Pass in *None* to disable callbacks.

Note: Due to a risk of circular runaway leading to stack overflow, it is not allowed to call functions in the DLL from within the callback function. If a call is made to a function in the DLL the function will raise an exception.

Examples

```
from ctypes import c_ubyte
from msl.equipment.resources import NKT

@NKT.DeviceStatusCallback
def device_callback(port, dev_id, status, length, address):
    # 'address' is an integer and represents the address of c_void_p
    # from the callback
```

(continues on next page)

(continued from previous page)

```

data = bytearray((c_ubyte * length).from_address(address) [:])
print('The port is {!r}'.format(port))
print('The device ID is {}'.format(dev_id))
print('The device status is {!r}'.format(NKT.
    DeviceStatusTypes(status)))
print('The device data is {!r}'.format(data))

NKT.set_callback_device_status(device_callback)

```

static set_callback_port_status(callback)

Enables/Disables a callback for port status changes.

Used by the [open_ports\(\)](#) and [close_ports\(\)](#) functions.

Parameters

callback (*PortStatusCallback*) – A *PortStatusCallback* object.
Pass in **None** to disable callbacks.

Note: Due to a risk of circular runaway leading to stack overflow, it is not allowed to call functions in the DLL from within the callback function. If a call is made to a function in the DLL the function will raise an exception.

Examples

```

from msl.equipment.resources import NKT

@NKT.PortStatusCallback
def port_callback(port, status, cur_scan, max_scan, device):
    print('The port is {!r}'.format(port))
    print('The port status is {!r}'.format(NKT.
        PortStatusTypes(status)))
    print('Current scanned address or device found address is {}'.format(cur_scan))
    print('There are {} addresses to scan in total'.format(max_scan))
    print('Found device with type {}'.format(device))

NKT.set_callback_port_status(port_callback)

```

static set_callback_register_status(callback)

Enables/Disables a callback for register status changes.

Parameters

callback (*RegisterStatusCallback*) – A *RegisterStatusCallback* object.
Pass in **None** to disable callbacks.

Note: Due to a risk of circular runaway leading to stack overflow, it is not allowed to call functions in the DLL from within the callback function. If a call is made to a function in the

DLL the function will raise an exception.

Examples

```
from ctypes import c_ubyte
from msl.equipment.resources import NKT

@NKT.RegisterStatusCallback
def register_callback(port, dev_id, reg_id, reg_status, reg_type,_
    ↴length, address):
    # 'address' is an integer and represents the address of c_void_p
    ↴from the callback
    data = bytearray((c_ubyte * length).from_address(address)[:])
    print('The port is {!r}'.format(port))
    print('The device ID is {}'.format(dev_id))
    print('The register ID is {}'.format(reg_id))
    print('The register status is {!r}'.format(NKT.
    ↴RegisterStatusTypes(reg_status)))
    print('The register type is {!r}'.format(NKT.
    ↴RegisterDataTypes(reg_type)))
    print('The register data is {!r}'.format(data))

NKT.set_callback_register_status(register_callback)
```

`msl.equipment.resources.nkt.nktpdll.unknown_error(result)`

msl.equipment.resources.omega package

Resources for equipment from [OMEGA](#).

Submodules

msl.equipment.resources.omega.ithx module

OMEGA iTHX Series Temperature and Humidity Chart Recorder.

This class is compatible with the following model numbers:

- iTHX-W3
- iTHX-D3
- iTHX-SD
- iTHX-M
- iTHX-W
- iTHX-2

```
class msl.equipment.resources.omega.iTHX(record)
```

Bases: [ConnectionSocket](#)

OMEGA iTHX Series Temperature and Humidity Chart Recorder.

The [properties](#) for an iTHX connection supports the following key-value pairs in the [Connections Database](#):

```
'nprobes': int, the number of probes the device has  
' nbytes': int, the number of bytes to read from each probe
```

as well as those key-value pairs supported by the parent [ConnectionSocket](#) class.

Do not instantiate this class directly. Use the [connect\(\)](#) method to connect to the equipment.

Parameters

record ([EquipmentRecord](#)) – A record from an [Equipment-Register Database](#).

temperature(*probe=1, celsius=True, nbytes=None*)

Read the temperature.

Parameters

- **probe** ([int](#), optional) – The probe number to read the temperature of (for iTHX's that contain multiple probes).
- **celsius** (class:[bool](#), optional) – [True](#) to return the temperature in celsius, [False](#) for fahrenheit.
- **nbytes** (class:[int](#), optional) – The number of bytes to read. If [None](#) then read until the termination character sequence.

Returns

[float](#) or [tuple](#) of [float](#) – The temperature.

humidity(*probe=1, nbytes=None*)

Read the percent humidity.

Parameters

- **probe** ([int](#), optional) – The probe number to read the humidity of (for iTHX's that contain multiple probes).
- **nbytes** (class:[int](#), optional) – The number of bytes to read. If [None](#) then read until the termination character sequence.

Returns

[float](#) or [tuple](#) of [float](#) – The percent humidity.

dewpoint(*probe=1, celsius=True, nbytes=None*)

Read the dew point.

Parameters

- **probe** ([int](#), optional) – The probe number to read the dew point of (for iTHX's that contain multiple probes).
- **celsius** ([bool](#), optional) – [True](#) to return the dew point in celsius, [False](#) for fahrenheit.

- **nbytes** (class:`int`, optional) – The number of bytes to read. If `None` then read until the termination character sequence.

Returns

`float` or `tuple` of `float` – The dew point.

temperature_humidity(*probe=1, celsius=True, nbytes=None*)

Read the temperature and the humidity.

Parameters

- **probe** (`int`, optional) – The probe number to read the temperature and humidity of (for iTHX's that contain multiple probes).
- **celsius** (`bool`, optional) – `True` to return the temperature in celsius, `False` for fahrenheit.
- **nbytes** (class:`int`, optional) – The number of bytes to read. If `None` then read until the termination character sequence. If specified, *nbytes* is the combined value to read both values.

Returns

- `float` – The temperature.
- `float` – The humidity.

temperature_humidity_dewpoint(*probe=1, celsius=True, nbytes=None*)

Read the temperature, the humidity and the dew point.

Parameters

- **probe** (`int`, optional) – The probe number to read the temperature, humidity and dew point (for iTHX's that contain multiple probes).
- **celsius** (`bool`, optional) – If `True` then return the temperature and dew point in celsius, `False` for fahrenheit.
- **nbytes** (`int`, optional) – The number of bytes to read. If `None` then read until the termination character sequence. If specified, *nbytes* is the combined value to read all three values.

Returns

- `float` – The temperature.
- `float` – The humidity.
- `float` – The dew point.

reset(*wait=True, password=None, port=2002, timeout=10*)

Power reset the iServer.

Some iServers accept the reset command to be sent via the TCP/UDP protocol and some require the reset command to be sent via the Telnet protocol.

Parameters

- **wait** (`bool`, optional) – Whether to wait for the connection to the iServer to be re-established before returning to the calling program. Rebooting an iServer takes about 10 to 15 seconds.

- **password** (`str`, optional) – The administrator’s password of the iServer. If not specified then uses the default manufacturer’s password. Only used if the iServer needs to be reset via the Telnet protocol.
- **port** (`int`, optional) – The port to use for the Telnet connection. Only used if the iServer needs to be reset via the Telnet protocol.
- **timeout** (`float`, optional) – The timeout value to use during the Telnet session. Only used if the iServer needs to be reset via the Telnet protocol.

start_logging(*path*, *wait*=60, *nprobes*=None, *nbytes*=None, *celsius*=True, *msg_format*=None, *db_timeout*=10, *validator*=None)

Start logging the temperature, humidity and dew point to the specified path.

The information is logged to an [SQLite](#) database. To stop logging press CTRL+C.

Parameters

- **path** (`str`) – The path to the [SQLite](#) database. If you only specify a directory then a database with the default filename, `model_serial.sqlite3`, is created/opened in this directory.
- **wait** (`int`, optional) – The number of seconds to wait between each log event.
- **nprobes** (`int`, optional) – The number of probes that the iServer has (1 or 2). If not specified then gets the value defined in [properties](#). Default is 1.
- **nbytes** (`int`, optional) – The number of bytes to read from each probe (the probes are read sequentially). The value is passed to [temperature_humidity_dewpoint\(\)](#). If not specified then gets the value defined in [properties](#). Default is `None`.
- **celsius** (`bool`, optional) – `True` to return the temperature and dew point in celsius, `False` for fahrenheit.
- **msg_format** (`str`, optional) – The format to use for the INFO logging messages each time data is read from an iServer. The format must use the `str.format()` syntax, `{}`. The positional arguments to `str.format()` are the values from the iServer, where the values are `(temperature, humidity, dewpoint)` for a 1-probe sensor and `(temperature1, humidity1, dewpoint1, temperature2, humidity2, dewpoint2)` for a 2-probe sensor. The keyword arguments to `str.format()` are the attributes of an [EquipmentRecord](#).

Examples:

- `T={0} H={1} D={2}`
- `{connection[address]} T={0:.1f} H={1:.1f} D={2:.1f}`
- `T1={0} T2={3} H1={1} H2={4} D1={2} D2={5}`
- `{alias} {serial} -> T={0}C H={1}% D={2}C`
- **db_timeout** (`float`, optional) – The number of seconds the connection to the database should wait for the lock to go away until raising an exception.

- **validator** – A callback that is used to validate the data. The callback must accept two arguments (*data*, *ithx*), where *data* is a `tuple` of the temperature, humidity and dewpoint values for each probe and *ithx* is the *iTHX* instance (i.e., *self*). The callback must return a value whose truthiness decides whether to insert the data into the database. If the returned value evaluates to `True` then the data is inserted into the database.

```
static data(path, start=None, end=None, as_datetime=True, select='*')
```

Fetch all the log records between two dates.

Parameters

- **path** (`str`) – The path to the `SQLite` database.
- **start** (`datetime` or `str`, optional) – Include all records that have a timestamp \geq *start*. If a `str` then in the ISO 8601 `yyyy-mm-dd` or `yyyy-mm-ddTHH:MM:SS` format.
- **end** (`datetime` or `str`, optional) – Include all records that have a timestamp \leq *end*. If a `str` then in the ISO 8601 `yyyy-mm-dd` or `yyyy-mm-ddTHH:MM:SS` format.
- **as_datetime** (`bool`, optional) – Whether to fetch the timestamps in the database as `datetime` objects. If `False` then the timestamps will be of type `str` and this function will return much faster if requesting data over a large date range.
- **select** (`str` or `list` of `str`, optional) – The field name(s) in the database table to use for the SELECT SQL command (e.g., '`datetime`', '`temperature`' or `['datetime', 'humidity']`).

Returns

`list` of `tuple` – A list of (`pid`, `datetime`, `temperature`, `humidity`, `dewpoint`, ...) log records, depending on the value of *select*.

```
msl.equipment.resources.omega.ithx.convert_datetime(value)
```

Convert a date and time to a `datetime` object.

Parameters

value (`bytes`) – The datetime value from an SQLite database.

Returns

`datetime.datetime` – The *value* as a `datetime` object.

msl.equipment.resources.optosigma package

Resources for equipment from OptoSigma.

Submodules

`msl.equipment.resources.optosigma.shot702 module`

Two-axis stage controller (SHOT-702) from OptoSigma.

`class msl.equipment.resources.optosigma.shot702.SHOT702(record)`

Bases: `ConnectionSerial`

Two-axis stage controller (SHOT-702) from OptoSigma.

Do not instantiate this class directly. Use the `connect()` method to connect to the equipment.

Parameters

`record (EquipmentRecord)` – A record from an *Equipment-Register Database*.

`get_input_status()`

Get the I/O input connector status.

Returns

`status (int)` – Can either be 0 or 1 – see manual.

`get_speed()`

Get the speed that each stage moves to a position.

Returns

`dict` –

The speed of each stage. The returned value has the form:

```
{  
    'stage1' : (minimum, maximum, acceleration),  
    'stage2' : (minimum, maximum, acceleration),  
}
```

`get_speed_origin()`

Get the speed that each stage moves to the origin.

Returns

`dict` –

The speed of each stage. The returned value has the form:

```
{  
    'stage1' : (minimum, maximum, acceleration),  
    'stage2' : (minimum, maximum, acceleration),  
}
```

`get_steps()`

Get the number of steps for each stage.

Returns

- `int` – The number of steps for stage 1.

- `int` – The number of steps for stage 2.

get_travel_per_pulse()

Get the travels per pulse for each stage.

Returns

- `float` – The travel per pulse for stage 1.
- `float` – The travel per pulse for stage 2.

get_version()

Get the version number.

Returns

`str` – The version number.

home(*stage*)

Move the stage(s) to the home position.

Parameters

`stage` (`int` or `str`) – The stage(s) to home. Allowed values:

- 1 (home stage 1),
- 2 (home stage 2), or
- 'W' (home stages 1 and 2).

Raises

`OptoSigmaError` – If there was an error processing the command.

is_moving()

Whether a stage is busy moving.

Returns

`bool` – Whether a stage is busy moving.

move(*stage, direction*)

Start moving the stage(s), at the minimum speed, in the specified direction.

Parameters

- `stage` (`int` or `str`) – The stage(s) to move. Allowed values:
 - 1 (start moving stage 1),
 - 2 (start moving stage 2), or
 - 'W' (start moving stages 1 and 2).
- `direction` (`str`) – The direction that the stage(s) should move. Allowed values are:
 - '+' or '-' (move a single stage in the specified direction)
 - '++' (move stage 1 in the + direction, stage 2 in the + direction)
 - '+-' (move stage 1 in the + direction, stage 2 in the - direction)
 - '-+' (move stage 1 in the - direction, stage 2 in the + direction)
 - '--' (move stage 1 in the - direction, stage 2 in the - direction)

Raises

OptoSigmaError – If there was an error processing the command.

`move_absolute(stage, *position)`

Move the stage(s) to the specified position.

Examples:

- `move_absolute(1, 1000)`
 - move stage 1 to position 1000 in the + direction
- `move_absolute(2, -5000)`
 - move stage 2 to position 5000 in the - direction
- `move_absolute('W', 1000, -5000)`
 - move stage 1 to position 1000 in the + direction, and
 - move stage 2 to position 5000 in the - direction

Parameters

- **stage** (`int` or `str`) – The stage(s) to move. Allowed values: 1, 2, 'W'.
- **position** (`int` or `tuple of int`) – The position the stage(s) should move to.

Raises

OptoSigmaError – If there was an error processing the command.

`move_relative(stage, *num_pulses)`

Move the stage(s) by a relative amount.

Examples:

- `move_relative(1, 1000)`
 - move stage 1 by 1000 pulses in the + direction
- `move_relative(2, -5000)`
 - move stage 2 by 5000 pulses in the - direction
- `move_relative('W', 1000, -5000)`
 - move stage 1 by 1000 pulses in the + direction, and
 - move stage 2 by 5000 pulses in the - direction

Parameters

- **stage** (`int` or `str`) – The stage(s) to move. Allowed values: 1, 2, 'W'.
- **num_pulses** (`int` or `tuple of int`) – The number of pulses the stage(s) should move.

Raises

OptoSigmaError – If there was an error processing the command.

set_mode(*stage, mode*)

Set whether the stage(s) can be moved by hand or by the motor.

Parameters

- **stage** (`int` or `str`) – The stage(s) to set the mode of. Allowed values:
 - 1 (set the mode for stage 1),
 - 2 (set the mode for stage 2), or
 - 'W' (set the mode for stages 1 and 2).
- **mode** (`int`) – Whether the stage(s) can be moved by hand (0) or motor (1).

Raises

`OptoSigmaError` – If there was an error processing the command.

set_origin(*stage*)

Set the origin of the stage(s) to its current position.

Parameters

- stage** (`int` or `str`) – The stage(s) to set the home of. Allowed values:
- 1 (set the home for stage 1),
 - 2 (set the home for stage 2), or
 - 'W' (set the home for stages 1 and 2).

Raises

`OptoSigmaError` – If there was an error processing the command.

set_output_status(*status*)

Set the I/O output status.

Parameters

- status** (`int`) – Can either be 0 or 1 – see manual.

Raises

`OptoSigmaError` – If there was an error processing the command.

set_speed(*stage, minimum, maximum, acceleration*)

Set the minimum, maximum and acceleration values when moving to a position.

Examples:

- `set_speed(1, 100, 1000, 50)`
 - set stage 1 to a minimum speed of 100 PPS, maximum speed of 1000 PPS and a 50 ms acceleration/deceleration time.
- `set_speed(2, 1000, 5000, 200)`
 - set stage 2 to a minimum speed of 1000 PPS, maximum speed of 5000 PPS and a 200 ms acceleration/deceleration time.
- `set_speed('W', [100,1000], [1000,5000], [50,200])`
 - set stage 1 to a minimum speed of 100 PPS, maximum speed of 1000 PPS and a 50 ms acceleration/deceleration time.

- set stage 2 to a minimum speed of 1000 PPS, maximum speed of 5000 PPS and a 200 ms acceleration/deceleration time.

Parameters

- **stage** (`int` or `str`) – The stage(s) to set the setting for. Allowed values: 1, 2, 'W'.
- **minimum** (`int` or `list` of `int`) – The minimum speed (allowed range 1 - 500k).
- **maximum** (`int` or `list` of `int`) – The maximum speed (allowed range 1 - 500k).
- **acceleration** (`int` or `list` of `int`) – The acceleration and deceleration time in milliseconds (allowed range 1 - 1000).

Raises

OptoSigmaError – If there was an error processing the command.

`set_speed_origin(stage, minimum, maximum, acceleration)`

Set the minimum, maximum and acceleration values when moving to the origin.

Examples:

- `set_speed_origin(1, 100, 1000, 50)`
 - set origin speed for stage 1 to a minimum speed of 100 PPS, maximum speed of 1000 PPS and a 50 ms acceleration/deceleration time.
- `set_speed_origin(2, 1000, 5000, 200)`
 - set origin speed for stage 2 to a minimum speed of 1000 PPS, maximum speed of 5000 PPS and a 200 ms acceleration/deceleration time.
- `set_speed_origin('W', [100,1000], [1000,5000], [50,200])`
 - set origin speed for stage 1 to a minimum speed of 100 PPS, maximum speed of 1000 PPS and a 50 ms acceleration/deceleration time.
 - set origin speed for stage 2 to a minimum speed of 1000 PPS, maximum speed of 5000 PPS and a 200 ms acceleration/deceleration time.

Parameters

- **stage** (`int` or `str`) – The stage(s) to set the setting for. Allowed values: 1, 2, 'W'.
- **minimum** (`int` or `list` of `int`) – The minimum origin speed (allowed range 1 - 500k).
- **maximum** (`int` or `list` of `int`) – The maximum origin speed (allowed range 1 - 500k).
- **acceleration** (`int` or `list` of `int`) – The origin acceleration and deceleration time in milliseconds (allowed range 1 - 1000).

Raises

OptoSigmaError – If there was an error processing the command.

set_steps(stage, num_steps)

Set the number of steps that the stage motor will use.

See the manual for more details – the S command.

Parameters

- **stage** (`int`) – The stage to set the steps of (must be 1 or 2).
- **num_steps** (`int`) – The number of steps that the motor should use (must be one of 1, 2, 4, 5, 8, 10, 20, 25, 40, 50, 80, 100, 125, 200, 250).

Raises

`OptoSigmaError` – If there was an error processing the command.

status()

Returns the current position and state of each stage.

Returns

- **pos1** (`int`) – The current position of stage 1.
- **pos2** (`int`) – The current position of stage 2.
- **state** (`str`) – The stopped state of the stage (one of 'L', 'M', 'W', 'K') – see the manual for more details.
- **is_moving** (`bool`) – Whether a stage is busy moving.

Raises

`OptoSigmaError` – If there was an error processing the command.

stop()

Immediately stop both stages from moving.

Raises

`OptoSigmaError` – If there was an error processing the command.

stop_slowly(stage)

Slowly bring the stage(s) to a stop.

Parameters

stage (`int` or `str`) – The stage(s) to slowly stop. Allowed values:

- 1 (slowly stop stage 1),
- 2 (slowly stop stage 2), or
- 'W' (slowly stop stages 1 and 2).

Raises

`OptoSigmaError` – If there was an error processing the command.

wait(callback=None, sleep=0.05)

Wait for the stages to finish moving.

This is a blocking call because it uses `time.sleep()`.

Parameters

- **callback** (`callable()`, optional) – A callable function. The function will receive 4 arguments – the returned values from `status()`

- **sleep** (`float`, optional) – The number of seconds to wait between calls to *callback*.

`msl.equipment.resources.optronic_laboratories` package

Resources for equipment from Optronic Laboratories.

Submodules

`msl.equipment.resources.optronic_laboratories.ol756ocx_32` module

Load the 32-bit OL756SDKActiveXCtrl library using `MSL_LoadLib`.

```
class msl.equipment.resources.optronic_laboratories.ol756ocx_32.OL756(host, port,  
**kwargs)
```

Bases: `Server32`

Communicates with the 32-bit OL756SDKActiveXCtrl library.

accumulate_signals(meas_type)

Function needs to be called after a measurement was performed.

This essentially accumulates the data together until the user is ready to average out the data. This function is used in combination with `reset_averaging()` and `do_averaging()`.

Parameters

`meas_type` (`int`) – The measurement type wanted.

- 0 - Irradiance
- 1 - Radiance
- 2 - Transmittance
- 3 - Irradiance Calibration
- 4 - Radiance Calibration
- 5 - Transmittance Calibration

connect_to.ol756(mode, com_port=1)

Desired mode to connect to OL756. If attempting to connect in RS232 or USB mode, and OL756 is not detected, then a dialog box will appear to prompt user to select either to retry, cancel or switch to DEMO.

Parameters

- `mode` (`int`) – Valid modes are:
 - -1: Disconnect. Call this before quitting the application.
 - 0: RS232
 - 1: USB
 - 2: DEMO mode

- **com_port** (`int`, optional) – If connecting through RS232 then *port* is the COM port number to use.

Returns

`int` – The mode that was actually used for the connection.

do_averaging(meas_type, num_to_average)

Function divides the accumulated signal by the number of scans performed. It then sets the array containing the data with the averaged data. This function is used in combination with `reset_averaging()` and `accumulate_signals()`.

Parameters

- **meas_type** (`int`) – The measurement type wanted.
 - 0 - Irradiance
 - 1 - Radiance
 - 2 - Transmittance
 - 3 - Irradiance Calibration
 - 4 - Radiance Calibration
 - 5 - Transmittance Calibration
- **num_to_average** (`int`) – The number of scans to average.

do_calculations(meas_type)

Function needs to be called after each measurement to update the calculations.

Parameters

meas_type (`int`) – The measurement type wanted.

- 0 - Irradiance
- 1 - Radiance
- 2 - Transmittance
- 3 - Irradiance Calibration
- 4 - Radiance Calibration
- 5 - Transmittance Calibration

enable_calibration_file(meas_type, enable)

Enables or disables the use of a calibration file.

Use this option to generate calibrated results. To open a standard file used to create a calibration, use `enable_standard_file()` instead.

The user should call `load_calibration_file()` first to load the calibration file before enabling it.

Parameters

- **meas_type** (`int`) – The measurement type wanted.
 - 0 - Irradiance
 - 1 - Radiance

- 2 - Transmittance
- **enable (bool)** – Whether to enable or disable the use of a calibration file.

enable_dark_current(enable)

Turn the dark current on or off.

Enable this feature if you want the dark current automatically acquired and subtracted before each measurement. If you wish to take a dark current manually, see the [get_dark_current\(\)](#) function.

The parameters for the dark current will need to be set using [set_dark_current_params\(\)](#).

Parameters

- enable (bool)** – Whether to turn the dark current on or off.

enable_pmt_protection_mode(enable)

Turn the PMT protection routines on or off.

Enable this feature if you want the PMT to be shielded while traveling through high intensity spikes. This feature will make the scan slower since the wavelength and filter drive will move asynchronously.

The PMT is still protected by the hardware. This function prevents exposure of the PMT while traveling.

Parameters

- enable (bool)** – Whether to turn the PMT protection routines on or off.

enable_standard_file(meas_type, enable)

Function enables standard files to be used.

To open a calibration file used to create a measurement, use [enable_calibration_file\(\)](#) instead.

The user should call [load_standard_file\(\)](#) first to load the standard file before enabling it.

Parameters

- **meas_type (int)** – The calibration measurement type wanted.
 - 3 - Irradiance Calibration
 - 4 - Radiance Calibration
 - 5 - Transmittance Calibration
- **enable (bool)** – Whether to turn the application of the standard file on or off.

export_config_file(file_path)

Exports the config file into a OL756 compatible configuration file.

Not all settings used will be applicable.

Parameters

- file_path (str)** – A valid path to save the file at.

export_registry()

Save data out to the Windows registry.

Make sure that a read was done at some point using [*import_registry\(\)*](#). Does not create a configuration file that can be loaded into another computer. For that particular function, call [*export_config_file\(\)*](#).

get_adaptive_int_time_index(*gain_index*)

Get the adaptive integration time index.

Parameters

gain_index ([int](#)) – The index of the gain to use to get the integration time.

- 0 - 1.0E-5
- 1 - 1.0E-6
- 2 - 1.0E-7
- 3 - 1.0E-8
- 4 - 1.0E-9
- 5 - 1.0E-10
- 6 - 1.0E-11

Returns

[int](#) – The adaptive integration time index.

get_cri(*meas_type*, *index*)

Get the color-rendering information.

The user should call [*do_calculations\(\)*](#) at least once before calling this function.

Parameters

- **meas_type** ([int](#)) – The measurement type wanted.
 - 0 - Irradiance
 - 1 - Radiance
 - 2 - Transmittance
 - 3 - Irradiance Calibration
 - 4 - Radiance Calibration
 - 5 - Transmittance Calibration
- **index** ([int](#)) – The color-rendering index.
 - 0 - General CRI
 - 1 - Light Greyish Red (CRI#1)
 - 2 - Dark Greyish Yellow (CRI#2)
 - 3 - Strong Yellow Green (CRI#3)
 - 4 - Moderate Yellowish Green (CRI#4)
 - 5 - Light Bluish Green (CRI#5)

- 6 - Light Blue (CRI#6)
- 7 - Light Violet (CRI#7)
- 8 - Light Reddish Purple (CRI#8)
- 9 - Strong Red (CRI#9)
- 10 - Strong Yellow (CRI#10)
- 11 - Strong Green (CRI#11)
- 12 - Strong Blue (CRI#12)
- 13 - Light Yellowish Pink (CRI#13)
- 14 - Moderate Olive Green (CRI#14)

Returns

`float` – The color-rendering information.

get_cal_array()

This method allows user to get the spectral data of a calibration after it is made. The data allows the user to take the data and create their own data files.

Returns

- `int` – A pointer to an array of signals.
- `int` – The number of points acquired.

get_cal_file_enabled(meas_type)

Checks to see if the calibration file is enabled.

The user should call `load_calibration_file()` first to load the calibration file before enabling it.

Parameters

`meas_type (int)` – The measurement type wanted.

- 0 - Irradiance
- 1 - Radiance
- 2 - Transmittance

Returns

`bool` – Whether the calibration file is enabled.

get_calculated_data(meas_type, index)

Gets data calculated from the intensities.

The user should call `do_calculations()` at least once before calling this function.

Parameters

- `meas_type (int)` – The measurement type wanted.
 - 0 - Irradiance
 - 1 - Radiance
 - 2 - Transmittance
 - 3 - Irradiance Calibration

- 4 - Radiance Calibration
 - 5 - Transmittance Calibration
- **index** (`int`) – The index to retrieve data of.
 - 0 - Color Temperature
 - 1 - Dominant Wavelength
 - 2 - LED Half Bandwidth
 - 3 - Left Half Bandwidth
 - 4 - Right Half Bandwidth
 - 5 - Peak Spectral Value
 - 6 - LEDPeakWavelength
 - 7 - Radiometric Value
 - 8 - Purity
 - 9 - Center Wavelength
 - 10 - Photometric Value

Returns

`float` – Pointer to a double to hold the data.

get_calibration_file(meas_type)

Get a calibration file that is loaded.

Parameters

meas_type (`int`) – The measurement type wanted.

- 0 - Irradiance
- 1 - Radiance
- 2 - Transmittance

Returns

`str` – String containing the name and path of the calibration file that is loaded for a particular measurement type.

get_chromaticity_data(meas_type, index)

Get the calculated chromaticity values requested.

Must have called `do_calculations()` at least once.

Parameters

- **meas_type** (`int`) – The measurement type wanted.
 - 0 - Irradiance
 - 1 - Radiance
 - 2 - Transmittance
 - 3 - Irradiance Calibration
 - 4 - Radiance Calibration

- 5 - Transmittance Calibration
- **index (int)** – The chromaticity index value [0..70]. See the SDK manual for more details.

Returns

float – Pointer to a double to hold the data.

get_dark_current(*use_compensation*)

Takes a manual dark current.

User will have to subtract from data array by retrieving this array via a [*get_cal_array\(\)*](#) or [*get_signal_array\(\)*](#). This is a special function and most users will want to use [*enable_dark_current\(\)*](#) instead because it automatically does the subtraction.

Function if called externally by user program will not have result saved out to data file. If the [*enable_dark_current\(\)*](#) was enabled, then this function need should not be called.

Parameters

use_compensation (int) – Adjusts dark current for more dynamic ranging using reverse current.

Returns

float – The dark current.

get_dark_current_enable()

Returns whether the dark-current mode is enabled.

Returns

bool – Whether the dark-current mode is enabled or disabled.

get_dark_current_mode()

Returns whether the dark current is taken at a wavelength or in shutter mode.

Returns

int – The dark-current mode

- 0 - Dark current in wavelength mode (taken at a particular wavelength designated by the user).
- 1 - Dark current in shutter mode

get_dark_current_wavelength()

Get the dark current wavelength.

Returns

float – Wavelength that the dark current will be taken at.

get_ending_wavelength()

Get the ending wavelength of the scan range.

Returns

float – The ending wavelength, in nanometers, of the scan range.

get_gain_index()

Get the index of the gain that will be applied when the parameters are to be sent down.

Applies to both quick scan and point to point scans.

Returns

`int` – The gain index.

- 0 - 1.0E-5
- 1 - 1.0E-6
- 2 - 1.0E-7
- 3 - 1.0E-8
- 4 - 1.0E-9
- 5 - 1.0E-10 (Point to Point mode only)
- 6 - 1.0E-11 (Point to Point mode only)
- 7 - Auto Gain Ranging (Point to Point mode only)

get_increment()

Get the wavelength increment that is used for a scan.

Returns

`float` – The wavelength increment, in nanometers.

get_increment_index()

Get the index of the wavelength increment that is used for a scan.

Applies to both quick scan and point to point scans.

Returns

`int` – Index of the wavelength increment of a scan.

- 0 - 0.025 nm
- 1 - 0.05 nm
- 2 - 0.1 nm
- 3 - 0.2 nm
- 4 - 0.5 nm
- 5 - 1.0 nm
- 6 - 2.0 nm
- 7 - 5.0 nm
- 8 - 10.0 nm

get_integration_time_index(*scan_mode*)

Get the index into the integration time array.

Applies to both quick scan and point to point scans. In quick scan, the speed will vary based on the scan range and increments.

Parameters

`scan_mode` (`int`) – The scan mode to use to get the index of.

Returns

`int` – Point to Point mode

- 0 - 1.000 sec
- 1 - 0.500 sec
- 2 - 0.200 sec
- 3 - 0.100 sec
- 4 - 0.050 sec
- 5 - 0.020 sec
- 6 - 0.010 sec
- 7 - 0.005 sec
- 8 - 0.002 sec
- 9 - 0.001 sec
- 10 - Adaptive (Point To Point mode only)

Quick Scan mode

- 0 - slowest
- 10 - fastest

get_ocx_version()

Get the version of the OL756 SDK ActiveX control.

Returns

`str` – The software version.

get_pmt_flux_overload()

Get the voltage of the photomultiplier tube flux overload.

Returns

`float` – Voltage that the PMT will determine to be at the overload point.

get_pmt_voltage()

Returns the voltage that will sent or has been sent down to the PMT.

Returns

`float` – Voltage value, in volts, of the photomultiplier tube.

get_quick_scan_rate()

Returns the rate at the quick scan index.

Returns

`float` – Rate of the quick scan at the current index in nm/s.

get_quick_scan_rate_index()

Returns the index of the quick scan rate.

Returns

`int` – Index of the quick scan rate.

get_scan_mode()

Get the mode the scan will be done in.

Returns

`int` – The scan mode

- 0 - Point to Point mode
- 1 - Quick Scan mode

get_settling_time()

Get the settling time.

Settling time is time where the wavelength drive pauses once it reaches its target wavelength.

Returns

`float` – Settling time, in seconds, to be sent down or has already been sent to the system.

get_signal_array()

Get the spectral data of a measurement after it is made.

Returns

`tuple` – The spectral data.

get_standard_file(meas_type)

Retrieves the name of standard file.

Parameters

`meas_type (int)` – The measurement type wanted.

- 3 - Irradiance calibration
- 4 - Radiance calibration
- 5 - Transmittance calibration

Returns

`str` – String containing the name and path of the standard file that is loaded for a particular calibration type.

get_start_wavelength()

Get the starting wavelength of a scan.

Applies to both quick scan and point to point scans.

Returns

`float` – The wavelength, in nanometers, that the scan will start from.

get_std_file_enabled(meas_type)

Checks to see if the standard file is enabled.

The user should call `load_standard_file()` first to load the standard file before enabling it.

Parameters

`meas_type (int)` – The calibration type wanted.

- 0 - Irradiance
- 1 - Radiance
- 2 - Transmittance

Returns

`bool` – Whether a standard file is enabled.

`import_config_file(path)`

The file is a standard OL756 configuration file.

Not all settings used will be applicable. Measurement type is not used because in the SDK, the `take_point_to_point_measurement()` function has as an input the measurement type. The user should select the type and not have it based on the configuration file.

Parameters

`path (str)` – A valid path to load the file at.

`import_registry()`

Loads data from the registry.

Loads default if no registry exists. To import the configuration from another computer, use `import_config_file()` instead.

Not all settings used will be applicable. Measurement type is not used because in the SDK, the `take_point_to_point_measurement()` function has as an input the measurement type. The user should select the type and not have it based on the configuration file.

`load_calibration_file(path, meas_type)`

Load a calibration file.

Parameters

- `path (str)` – The path of a calibration file.
- `meas_type (int)` – The measurement type.
 - 0 - Irradiance
 - 1 - Radiance
 - 2 - Transmittance

`load_standard_file(path, meas_type)`

Load a standard file.

Parameters

- `path (str)` – The path of a standard file.
- `meas_type (int)` – The measurement type.
 - 3 - Irradiance Calibration
 - 4 - Radiance Calibration
 - 5 - Transmittance Calibration

`manual_filter_drive_connect(connect)`

Used to connect or disconnect the filter drive.

Disconnecting essentially acquires scans without the filter.

Parameters

`connect (bool)` – Connect or disconnect the filter drive. Reconnecting will home the wavelength and filter drive.

manual_get_gain()

The index of the gain that will be applied when the parameters are to be sent down.

Returns

int – The gain index.

- 0 - 1.0E-5
- 1 - 1.0E-6
- 2 - 1.0E-7
- 3 - 1.0E-8
- 4 - 1.0E-9
- 5 - 1.0E-10 (Point to Point mode only)
- 6 - 1.0E-11 (Point to Point mode only)
- 7 - Auto Gain Ranging (Point to Point mode only)

manual_get_integration_time()

Returns the integration time set in the system.

Only applies to the integration time used for Point to Point scans.

Returns

float – The integration time in seconds.

manual_get_pmt_overload()

Returns the PMT overload voltage set in the system.

Returns

float – Overload voltage, in volts, of the photomultiplier tube.

manual_get_pmt_voltage()

Returns the PMT high voltage set in the system.

Returns

float – Voltage, in volts, of the photomultiplier tube.

manual_get_settling_time()

Returns the settling time of the instrument.

Returns

float – Settling time of the system in seconds.

manual_get_signal()

Returns the signal at the current position of the wavelength drive.

Returns

float – The signal, in amperes.

manual_home_ol756()

Homes the wavelength and filter drive.

Will reconnect the filter drive if it was disconnected

manual_move_to_wavelength(wavelength)

Moves the wavelength drive to a particular location.

Parameters

wavelength (`float`) – The wavelength to move the wavelength drive to.

manual_set_gain(gain_index, mode)

Set the gain.

Parameters

• **gain_index** (`int`) – The gain index.

- 0 - 1.0E-5
- 1 - 1.0E-6
- 2 - 1.0E-7
- 3 - 1.0E-8
- 4 - 1.0E-9
- 5 - 1.0E-10 (Point to Point mode only)
- 6 - 1.0E-11 (Point to Point mode only)
- 7 - Auto Gain Ranging (Point to Point mode only)

• **mode** (`int`) – The scan mode

- 0 - point to point
- 1 - quick scan

manual_set_integration_time(time)

Sets the integration time set in the system.

Only applies to the integration time used for Point to Point scans.

Parameters

time (`float`) – The integration time in seconds.

manual_set_pmt_overload(overload)

Sets the PMT overload voltage set in the system.

Parameters

overload (`float`) – Overload voltage, in volts, of the photomultiplier tube in Volts.

manual_set_pmt_voltage(voltage)

Sets the PMT high voltage set in the system.

Parameters

voltage (`float`) – Voltage, in volts, of the photomultiplier tube.

manual_set_settling_time(time)

Sets the settling time of the instrument.

Parameters

time (`float`) – Settling time of the system.

move_to_wavelength(wavelength)

Moves the wavelength drive to a particular location.

Parameters

wavelength (`float`) – The wavelength, in nanometers, to move the wavelength drive to.

read.ol756_flash_settings()

Reads the saved settings from the flash memory.

Reads the settings such as the grating alignment factor, filter skew and wavelength skew.
Loads these values into the ActiveX control memory.

reset_averaging(meas_type)

Resets the accumulated signal array for the specified measurement type.

This function is used in combination with `do_averaging()` and `accumulate_signals()`.

Parameters

meas_type (`int`) – The measurement type.

- 0 - Irradiance
- 1 - Radiance
- 2 - Transmittance
- 3 - Irradiance Calibration
- 4 - Radiance Calibration
- 5 - Transmittance Calibration

save_calibration_file(meas_type, path)

Create a OL756-compatible calibration file.

Parameters

- **meas_type** (`int`) – The measurement type.
 - 3 - Irradiance Calibration
 - 4 - Radiance Calibration
 - 5 - Transmittance Calibration
- **path** (`str`) – The path to save the calibration file to.

save_measurement_data(meas_type, path)

Save the measurement data to a OL756-compatible data file.

Parameters

- **meas_type** (`int`) – The measurement type.
 - 0 - Irradiance
 - 1 - Radiance
 - 2 - Transmittance
- **path** (`str`) – The path to save the data to.

send_down_parameters(*scan_mode*)

Sends down the parameters to the system.

This needs to be called whenever parameters dealing with the PMT or integration time and gain has changed. Needs to be called once before doing any measurements or other signal acquisition including dark current.

The following methods affect the parameters: *set_pmt_flux_overload_voltage()* *set_gain()* *set_integration_time()* *set_pmt_hi_voltage()* *set_settling_time()* *set_scan_range()* *set_adaptive_integration_time()*

Parameters

scan_mode (*int*) – The scan mode.

- 0 - Point to point
- 1 - Quick scan

set_adaptive_integration_time(*gain_index*, *speed_index*)

Sets the scan speed of the scan at a particular gain range.

Adaptive integration time is used solely for point to point scans in auto-gain ranging.

Parameters

- **gain_index** (*int*) – The index of the gain to use to set the integration time.
 - 0 - 1.0E-5
 - 1 - 1.0E-6
 - 2 - 1.0E-7
 - 3 - 1.0E-8
 - 4 - 1.0E-9
 - 5 - 1.0E-10
 - 6 - 1.0E-11
- **speed_index** (*int*) – The scan speed index [0..12] – 0=Slowest, 12=Fastest.

set_averaging_number_of_scan(*num_avg_scans*)

Set the number of scans to average.

Parameters

num_avg_scans (*int*) – The number of scans to average.

set_dark_current_params(*mode*, *wavelength*)

Sets the mode and the wavelength to use for a dark-current measurement.

Parameters

- **mode** (*int*) – The mode to use to acquire a dark-current measurement
 - 0 - wavelength
 - 1 - shutter

- **wavelength** (`float`) – The wavelength, in nanometers, to use for a dark-current measurement.

`set_gain(scan_mode, gain_index)`

Sets the index of the gain that will be applied when the parameters are to be sent down.

Applies to both quick scan and point to point scans.

Parameters

- **scan_mode** (`int`) – The scan mode
 - 0 - Point to Point
 - 1 - Quick Scan
- **gain_index** (`int`) – The gain index
 - 0 - 1.0E-5
 - 1 - 1.0E-6
 - 2 - 1.0E-7
 - 3 - 1.0E-8
 - 4 - 1.0E-9
 - 5 - 1.0E-10 (available only in Point to Point mode)
 - 6 - 1.0E-11 (available only in Point to Point mode)
 - 7 - Auto Gain Ranging (available only in Point to Point mode)

`set_integration_time(scan_mode, scan_speed)`

Sets the index of the scan speed used.

Applies to both quick scan and point to point scans. In quick scan, the speed will vary based on the scan range and increments.

Parameters

- **scan_mode** (`int`) – The scan mode
 - 0 - Point to Point
 - 1 - Quick Scan
- **scan_speed** (`int`) – Index to the integration time array

Point to Point mode

- 0 - 1.000 sec
- 1 - 0.500 sec
- 2 - 0.200 sec
- 3 - 0.100 sec
- 4 - 0.050 sec
- 5 - 0.020 sec
- 6 - 0.010 sec

- 7 - 0.005 sec
 - 8 - 0.002 sec
 - 9 - 0.001 sec
 - 10 - Adaptive (Point To Point mode only)
 - 11 - User defined (Point To Point mode only)
- Quick Scan mode
- 0 - slowest
 - 10 - fastest

set_pmt_flux_overload_voltage(*overload_voltage*)

Sets the value to use for the photomultiplier tube flux overload.

Parameters

overload_voltage (`float`) – Voltage that the PMT will determine to be at the overload point. Software only, because PMT has built-in protection also.

set_pmt_hi_voltage(*hi_voltage*)

Sets the value to be determined to be a flux overload by the software.

Parameters

hi_voltage (`float`) – Voltage, in volts, that the PMT will determine to be overload point.

set_reference_white_point(*white*, *user_def_x*, *user_def_y*)

Sets the value of the reference illuminant.

Parameters

- **white** (`int`) – The reference white point
 - 0 - Incandescent(A)
 - 1 - Direct Sunlight(B)
 - 2 - Indirect Sunlight(C)
 - 3 - Natural Daylight(D65)
 - 4 - Normalized Reference(E)
 - 5 - User Defined
- **user_def_x** (`float`) – User defined x on CIE chart.
- **user_def_y** (`float`) – User defined y on CIE chart.

set_scan_range(*start*, *end*, *inc_index*)

Sets the wavelength scan range.

Parameters

- **start** (`float`) – Starting wavelength, in nanometers.
- **end** (`float`) – Ending wavelength, in nanometers.
- **inc_index** (`int`) – Increment index, in nanometers.

set_settling_time(*time*)

Set the settling time.

Settling time is the time that the wavelength drive pauses once it reaches its target wavelength.

Parameters

time (`float`) – Settling Time in seconds to be sent down or has already been sent to the system.

set_tab_delimited_mode(*enable*)

Purpose of function is to set what mode to write the data files as.

Setting the tab delimited to true will write the data in a tab delimited format, else a false will write in a comma delimited format. Tab delimited files will not be compatible with some versions of the software. If you want data files to be compatible with v1.32 software and below, leave the mode to `False`.

Parameters

enable (`bool`) – Whether to use the new file format using TABs as a delimiter or the old file format compatible with v1.32 and below.

set_user_defined_integration_time(*time*)

Sets the user defined integration time to be used only in point to point scans and only if the user sets the integration time mode.

Parameters

time (`float`) – Integration time in seconds.

stop_measurement()

Stops a measurement.

Applies only to Point to Point measurements. Quick scans are done so quickly that there is no need to stop a measurement once it starts.

take_point_to_point_calibration(*meas_type*)

Takes a calibration in point to point mode.

Need to have called `send_down_parameters()` at least once before calling any of the measurement functions or data acquisition functions.

Parameters

meas_type (`int`) – The measurement type.

- 3 - Irradiance Calibration
- 4 - Radiance Calibration
- 5 - Transmittance Calibration

take_point_to_point_measurement(*meas_type*)

Takes a measurement in point to point mode.

Need to have called `send_down_parameters()` at least once before calling any of the measurement functions or data acquisition functions.

Parameters

meas_type (`int`) – The measurement type.

- 0 - Irradiance

- 1 - Radiance
- 2 - Transmittance

`take_quick_scan_calibration(meas_type)`

Takes a calibration in quick scan mode.

Need to have called `send_down_parameters()` at least once before calling any of the measurement functions or data acquisition functions.

Parameters

`meas_type` (`int`) – The measurement type.

- 3 - Irradiance Calibration
- 4 - Radiance Calibration
- 5 - Transmittance Calibration

`take_quick_scan_measurement(meas_type)`

Takes a measurement in quick scan mode.

Need to have called `send_down_parameters()` at least once before calling any of the measurement functions or data acquisition functions.

Parameters

`meas_type` (`int`) – The measurement type.

- 0 - Irradiance
- 1 - Radiance
- 2 - Transmittance

`msl.equipment.resources.optronic_laboratories.ol756ocx_64` module

Load the 32-bit OL756SDKActiveXCtrl library using `MSL_LoadLib`.

`class msl.equipment.resources.optronic_laboratories.ol756ocx_64.OL756(record=None)`

Bases: `Connection`

A wrapper around the OL756SDKActiveXCtrl library.

Attention: See the `ol756ocx_32.OL756` class for all available methods.

This class can be used with either a 32- or 64-bit Python interpreter to call the 32-bit functions in the OL756SDKActiveXCtrl library.

The `properties` for an OL756 connection supports the following key-value pairs in the `Connections Database`:

```
'mode': int, connection mode (0=RS232, 1=USB) [default: 1]  
'com_port': int, the COM port number (RS232 mode only) [default: 1]
```

Do not instantiate this class directly. Use the `connect()` method to connect to the equipment.

Parameters

record (*EquipmentRecord*) – A record from an *Equipment-Register Database*.

disconnect()

Disconnect from the OL756 spectroradiometer.

msl.equipment.resources.optronic_laboratories.ol_current_source module

Communicate with a DC current source from Optronic Laboratories.

class msl.equipment.resources.optronic_laboratories.ol_current_source.OLCurrentSource

Bases: `object`

Communicate with a DC current source from Optronic Laboratories.

Attention: The connection interface must be selected (using the buttons on the front panel) to be either RS-232 or IEEE-488 after the Current Source is initially powered on. Even if this is the default power-on interface, it must be manually re-selected before communication will work.

The *properties* for the connection supports the following key-value pairs in the *Connections Database*:

```
'address': int, the internal address of the device (RS-232 only)
↳ [default: 1]
'delay': float, the number of seconds to wait between a write-read
↳ transaction (RS-232 only) [default: 0.1]
```

as well as the key-value pairs supported by *ConnectionSerial* if using RS-232 as the interface or by *ConnectionGPIB* if using IEEE-488 as the interface.

Do not instantiate this class directly. Use the `connect()` method to connect to the equipment.

get_current() → float

Get the output current.

get_option(lamp: int, option: int) → str | float

Get the value of a lamp configuration option.

Parameters

- **lamp** – The lamp number (between 0 and 9, inclusive).
- **option** – The option type to read. Must be one of the following values
 - 40: Lamp Hours → `float`
 - 50: Recalibration interval (hours) → `float`
 - 60: Target units (A, V or W) → `str`
 - 70: Target value → `float`
 - 80: Current limit → `float`

- 90: Lamp description text → `str`
- 95: Wattage (L or H) → `str`

Returns

The value of the *option* that was requested.

`get_voltage()` → `float`

Get the output voltage.

`get_wattage()` → `float`

Get the output wattage.

`reset()` → `None`

Reset the communication buffers.

`select_lamp(lamp: int)` → `None`

Select a lamp.

Parameters

`lamp` – The lamp number (between 0 and 9, inclusive).

`set_current(amps: float)` → `float`

Set the target output current.

Parameters

`amps` – The target current, in Amps. If the value is above the target current limit for the presently selected lamp setup or if the value is less than the minimum supported current, the target current will not change.

Returns

The actual value of the output current after it was set.

`set_option(lamp: int, option: int, value: str | float)` → `None`

Set a value for one of the lamp configuration options.

Parameters

- `lamp` – The lamp number (between 0 and 9, inclusive).
- `option` – The option type to update. Must be one of the following values
 - 40: Lamp Hours
 - 50: Recalibration interval (hours)
 - 60: Target units (A, V or W)
 - 70: Target value
 - 80: Current limit
 - 90: Lamp description text
 - 95: Wattage (L or H)
- `value` – The value to write for *option*.

`set_voltage(volts: float)` → `float`

Set the target output voltage.

Parameters

volts – The target voltage, in Volts. If the value is above the target voltage limit for the presently selected lamp setup or if the value is less than the minimum supported voltage, the target voltage will not change.

Returns

The actual value of the output voltage after it was set.

set_wattage(watts: float) → float

Set the target output wattage.

Parameters

watts – The target wattage, in Watts. If the value is above the target wattage limit for the presently selected lamp setup or if the value is less than the minimum supported wattage, the target wattage will not change.

Returns

The actual value of the output wattage after it was set.

state() → bool

Returns whether the output is on or off.

property system_status_byte: int

The system status byte that is returned in every reply.

It is constructed as follows:

- bit 7: Busy flag (the device is performing a function)
- bit 6: Reserved
- bit 5: Reserved
- bit 4: Lamp status (0=off, 1=on)
- bit 3: Reserved
- bit 2: Reserved
- bit 1: Seeking current (1=current is ramping)
- bit 0: Reserved

target_info() → dict[str, int | float | str]

Get the target information of the currently-selected lamp.

Returns

The lamp number, target value and target unit. The key-value pairs are:

```
{'lamp': int, 'value': float, 'unit': str}
```

turn_off() → None

Turn the output off.

turn_on() → None

Turn the output on.

zero_voltage_monitor() → None

Zero the voltage monitor.

msl.equipment.resources.picotech package

Resources for equipment from Pico Technology.

Subpackages

msl.equipment.resources.picotech.errors module

Exceptions and error codes defined in the Pico Technology SDK.

```
class msl.equipment.resources.picotech.errors.PS2000Error(value, names=None,
                                                       *values, module=None,
                                                       qualname=None,
                                                       type=None, start=1,
                                                       boundary=None)
```

Bases: [IntEnum](#)

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.errors.PS3000Error(value, names=None,
                                                       *values, module=None,
                                                       qualname=None,
                                                       type=None, start=1,
                                                       boundary=None)
```

Bases: [IntEnum](#)

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_TO0_OLD = 8
```

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.callbacks.BlockReady

All BlockReady callbacks have the same function signature, so create a generic BlockReady callback.

msl.equipment.resources.picotech.picoscope.callbacks.ps2000aBlockReady

ps2000aBlockReady callback

msl.equipment.resources.picotech.picoscope.callbacks.ps3000aBlockReady

ps3000aBlockReady callback

msl.equipment.resources.picotech.picoscope.callbacks.ps4000BlockReady

ps4000BlockReady callback

msl.equipment.resources.picotech.picoscope.callbacks.ps4000aBlockReady

ps4000aBlockReady callback

msl.equipment.resources.picotech.picoscope.callbacks.ps5000BlockReady

ps5000BlockReady callback

msl.equipment.resources.picotech.picoscope.callbacks.ps5000aBlockReady

ps5000aBlockReady callback

msl.equipment.resources.picotech.picoscope.callbacks.ps6000BlockReady

ps6000BlockReady callback

msl.equipment.resources.picotech.picoscope.callbacks.ps2000aStreamingReady

ps2000aStreamingReady callback

msl.equipment.resources.picotech.picoscope.callbacks.ps3000aStreamingReady

ps3000aStreamingReady callback

msl.equipment.resources.picotech.picoscope.callbacks.ps4000StreamingReady

ps4000StreamingReady callback

```
msl.equipment.resources.picotech.picoscope.callbacks.ps4000aStreamingReady
    ps4000aStreamingReady callback

msl.equipment.resources.picotech.picoscope.callbacks.ps5000StreamingReady
    ps5000StreamingReady callback

msl.equipment.resources.picotech.picoscope.callbacks.ps5000aStreamingReady
    ps5000aStreamingReady callback

msl.equipment.resources.picotech.picoscope.callbacks.ps6000StreamingReady
    ps6000StreamingReady callback

msl.equipment.resources.picotech.picoscope.callbacks.ps2000aDataReady
    ps2000aDataReady callback

msl.equipment.resources.picotech.picoscope.callbacks.ps3000aDataReady
    ps3000aDataReady callback

msl.equipment.resources.picotech.picoscope.callbacks.ps4000aDataReady
    ps4000aDataReady callback

msl.equipment.resources.picotech.picoscope.callbacks.ps5000aDataReady
    ps5000aDataReady callback

msl.equipment.resources.picotech.picoscope.callbacks.ps5000aDataReady
    ps5000aDataReady callback

msl.equipment.resources.picotech.picoscope.callbacks.ps6000aDataReady
    ps6000aDataReady callback

msl.equipment.resources.picotech.picoscope.callbacks.PS3000_CALLBACK_FUNC
    PS3000_CALLBACK_FUNC callback

msl.equipment.resources.picotech.picoscope.callbacks.GetOverviewBuffersMaxMin
    GetOverviewBuffersMaxMin callback
```

[msl.equipment.resources.picotech.picoscope.channel module](#)

Contains the information about a PicoScope channel.

```
class msl.equipment.resources.picotech.picoscope.channel.PicoScopeChannel(channel,
    en-
    abled,
    cou-
    pling,
    volt-
    age_range,
    volt-
    age_offset,
    band-
    width,
    max_adu_value)
```

Bases: `object`

Contains the information about a PicoScope channel.

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

Parameters

- **channel** (`enum.IntEnum`) – The PSX000xChannel enum.
- **enabled** (`bool`) – Whether the channel is enabled.
- **coupling** (`enum.IntEnum`) – The PSX000xCoupling enum, 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 PSX000xBandwidthLimiter enum.
- **max_adu_value** (`int`) – The maximum analog-to-digital unit.

`property channel`

The PSX000xChannel enum.

Type

`enum.IntEnum`

`property enabled`

Whether the channel is enabled.

Type

`bool`

`property coupling`

The PSX000xCoupling enum, e.g. AC or DC.

Type

`enum.IntEnum`

`property voltage_range`

The voltage range, in Volts.

Type

`float`

`property voltage_offset`

The voltage offset, in Volts.

Type

`float`

`property bandwidth`

The PSX000xBandwidthLimiter enum.

Type

`enum.IntEnum` or `None`

property volts_per_adu

The conversion factor to convert ADU to volts

Type

`float`

property raw

The raw data, in ADU

Type

`numpy.ndarray`

property buffer

The raw data, in ADU

Type

`numpy.ndarray`

property volts

The data, in volts

Type

`numpy.ndarray`

property num_samples

The size of the data array.

Type

`int`

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(*value*,
names=None,
**values*,
mod-
ule=None,
qual-
name=None,
type=None,
start=1,
bound-
ary=None)**

Bases: `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(value,
    names=None,
    *values,
    module=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

```
A = 0
B = 1
C = 2
D = 3
EXT = 4
MAX_CHANNELS = 4
NONE = 5
```

```
class msl.equipment.resources.picotech.picoscope.enums.PS2000Range(value,
    names=None,
    *values,
    module=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *values,
    module=None,
    qual-name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

FS = 0

PS = 1

NS = 2

US = 3

MS = 4

S = 5

MAX = 6

```
class msl.equipment.resources.picotech.picoscope.enums.PS2000Info(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1, bound-
    ary=None)

Bases: IntEnum

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(value,
    names=None,
    *val-
    ues,
    mod-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)

Bases: IntEnum

RISING = 0

FALLING = 1

MAX = 2
```

```
class msl.equipment.resources.picotech.picoscope.enums.PS2000OpenProgress(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

FAIL = -1

PENDING = 0

COMPLETE = 1

```
class msl.equipment.resources.picotech.picoscope.enums.PS2000EtsMode(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

OFF = 0

FAST = 1

SLOW = 2

MAX = 3

```
class msl.equipment.resources.picotech.picoscope.enums.PS2000ButtonState(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

NO_PRESS = 0

SHORT_PRESS = 1

LONG_PRESS = 2

```
class msl.equipment.resources.picotech.picoscope.enums.PS2000SweepType(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

UP = 0

DOWN = 1

UPDOWN = 2

DOWNUP = 3

MAX = 4

```
class msl.equipment.resources.picotech.picoscope.enums.PS2000WaveType(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

LEVEL = 0

WINDOW = 1

```
class msl.equipment.resources.picotech.picoscope.enums.PS2000TriggerState(value,
                           names=None,
                           *val-
                           ues,
                           mod-
                           ule=None,
                           qual-
                           name=None,
                           type=None,
                           start=1,
                           bound-
                           ary=None)
```

Bases: [IntEnum](#)**DONT_CARE** = 0**TRUE** = 1**FALSE** = 2**MAX** = 3

```
class msl.equipment.resources.picotech.picoscope.enums.PS2000PulseWidthType(value,
                           names=None,
                           *val-
                           ues,
                           mod-
                           ule=None,
                           qual-
                           name=None,
                           type=None,
                           start=1,
                           bound-
                           ary=None)
```

Bases: [IntEnum](#)**NONE** = 0**LESS_THAN** = 1**GREATER_THAN** = 2**IN_RANGE** = 3**OUT_OF_RANGE** = 4

```
class msl.equipment.resources.picotech.picoscope.enums.PS2000AChannelBufferIndex(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

NONE = 0

OR = 1

AND = 2

THEN = 3

```
class msl.equipment.resources.picotech.picoscope.enums.PS2000DigitalPort(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

PORT0 = 128

PORT1 = 129

PORT2 = 130

PORT3 = 131

MAX = 4

```
class msl.equipment.resources.picotech.picoscope.enums.PS2000ADigitalChannel(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

```
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(value,
    names=None,
    *values, mod-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

```
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(value,
    names=None,
    *values,
    module=None,
    qualname=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

AC = 0

DC = 1

```
class msl.equipment.resources.picotech.picoscope.enums.PS2000AChannelInfo(value,
    names=None,
    *values,
    module=None,
    qualname=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

RANGES = 0

```
class msl.equipment.resources.picotech.picoscope.enums.PS2000AEtsMode(value,
    names=None,
    *values,
    module=None,
    qualname=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

OFF = 0

FAST = 1

SLOW = 2

MAX = 3

```
class msl.equipment.resources.picotech.picoscope.enums.PS2000ATimeUnits(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

FS = 0

PS = 1

NS = 2

US = 3

MS = 4

S = 5

MAX = 6

```
class msl.equipment.resources.picotech.picoscope.enums.PS2000ASweepType(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

UP = 0

DOWN = 1

UPDOWN = 2

DOWNUP = 3

MAX = 4

```
class msl.equipment.resources.picotech.picoscope.enums.PS2000AWaveType(value,
    names=None,
    *values,
    mod-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *val-
    ues,
    mod-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

OFF = 0

WHITENOISE = 1

PRBS = 2

```
class msl.equipment.resources.picotech.picoscope.enums.PS2000ASigGenTrigType(value,
    names=None,
    *val-
    ues,
    mode=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

RISING = 0

FALLING = 1

GATE_HIGH = 2

GATE_LOW = 3

```
class msl.equipment.resources.picotech.picoscope.enums.PS2000ASigGenTrigSource(value,
    names=None,
    *val-
    ues,
    mode=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

NONE = 0

SCOPE_TRIG = 1

AUX_IN = 2

EXT_IN = 3

SOFT_TRIG = 4

```
class msl.equipment.resources.picotech.picoscope.enums.PS2000AIndexMode(value,
    names=None,
    *values,
    mod-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

SINGLE = 0

DUAL = 1

QUAD = 2

MAX = 3

```
class msl.equipment.resources.picotech.picoscope.enums.PS2000A_ThresholdMode(value,
    names=None,
    *val-
    ues,
    mod-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

LEVEL = 0

WINDOW = 1

```
class msl.equipment.resources.picotech.picoscope.enums.PS2000AThresholdDirection(value,
    names=None,
    *val-
    ues,
    mod-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

```
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(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

```
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(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

DONT_CARE = 0

TRUE = 1

FALSE = 2

MAX = 3

```
class msl.equipment.resources.picotech.picoscope.enums.PS2000ARatioMode(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

NONE = 0

AGGREGATE = 1

DECIMATE = 2

AVERAGE = 4

```
class msl.equipment.resources.picotech.picoscope.enums.PS2000APulseWidthType(value,
    names=None,
    *values,
    module=None,
    qualname=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

NONE = 0

LESS_THAN = 1

GREATER_THAN = 2

IN_RANGE = 3

OUT_OF_RANGE = 4

```
class msl.equipment.resources.picotech.picoscope.enums.PS2000AHoldOffType(value,
    names=None,
    *values,
    module=None,
    qualname=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

TIME = 0

MAX = 1

```
class msl.equipment.resources.picotech.picoscope.enums.PS3000Channel(value,
    names=None,
    *values,
    module=None,
    qualname=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

```
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(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: `IntEnum`

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(value,
    names=None,
    *values,
    mod-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

SQUARE = 0

TRIANGLE = 1

SINE = 2

MAX = 3

```
class msl.equipment.resources.picotech.picoscope.enums.PS3000TimeUnits(value,
    names=None,
    *values,
    mod-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

FS = 0

PS = 1

NS = 2

US = 3

MS = 4

S = 5

MAX = 6

```
class msl.equipment.resources.picotech.picoscope.enums.PS3000Info(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1, bound-
    ary=None)
```

Bases: `IntEnum`

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(value,
    names=None,
    *val-
    ues,
    mod-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: `IntEnum`

RISING = 0

FALLING = 1

MAX = 2

```
class msl.equipment.resources.picotech.picoscope.enums.PS3000OpenProgress(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

FAIL = -1

PENDING = 0

COMPLETE = 1

```
class msl.equipment.resources.picotech.picoscope.enums.PS3000EtsMode(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

OFF = 0

FAST = 1

SLOW = 2

MAX = 3

```
class msl.equipment.resources.picotech.picoscope.enums.PS3000ThresholdDirection(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

```
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(value,
                           names=None,
                           *values,
                           mode=None,
                           quality=None,
                           name=None,
                           type=None,
                           start=1,
                           boundary=None)
```

Bases: [IntEnum](#)

```
LEVEL = 0
```

```
WINDOW = 1
```

```
class msl.equipment.resources.picotech.picoscope.enums.PS3000TriggerState(value,
                           names=None,
                           *values,
                           mode=None,
                           quality=None,
                           name=None,
                           type=None,
                           start=1,
                           boundary=None)
```

Bases: [IntEnum](#)

```
DONT_CARE = 0
```

TRUE = 1

FALSE = 2

MAX = 3

```
class msl.equipment.resources.picotech.picoscope.enums.PS3000PulseWidthType(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

NONE = 0

LESS_THAN = 1

GREATER_THAN = 2

IN_RANGE = 3

OUT_OF_RANGE = 4

```
class msl.equipment.resources.picotech.picoscope.enums.PS3000ABandwidthLimiter(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

BW_FULL = 0

BW_20MHZ = 1

```
class msl.equipment.resources.picotech.picoscope.enums.PS3000AChannelBufferIndex(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

PORT0 = 128

PORT1 = 129

PORT2 = 130

PORT3 = 131

MAX = 4

```
class msl.equipment.resources.picotech.picoscope.enums.PS3000ADigitalChannel(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *values, mod-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *values,
    mod-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

AC = 0

DC = 1

```
class msl.equipment.resources.picotech.picoscope.enums.PS3000AChannelInfo(value,
    names=None,
    *values,
    mode=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

RANGES = 0

```
class msl.equipment.resources.picotech.picoscope.enums.PS3000AEtsMode(value,
    names=None,
    *values,
    mode=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

OFF = 0

FAST = 1

SLOW = 2

MAX = 3

```
class msl.equipment.resources.picotech.picoscope.enums.PS3000ATimeUnits(value,
    names=None,
    *values,
    mode=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

FS = 0

PS = 1

NS = 2

US = 3

MS = 4

S = 5

MAX = 6

```
class msl.equipment.resources.picotech.picoscope.enums.PS3000ASweepType(value,
    names=None,
    *values,
    mode=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

UP = 0

DOWN = 1

UPDOWN = 2

DOWNUP = 3

MAX = 4

```
class msl.equipment.resources.picotech.picoscope.enums.PS3000AWaveType(value,
    names=None,
    *values,
    mode=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *values,
    module=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

OFF = 0

WHITENOISE = 1

PRBS = 2

```
class msl.equipment.resources.picotech.picoscope.enums.PS3000ASigGenTrigType(value,
    names=None,
    *values,
    module=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

RISING = 0

FALLING = 1

GATE_HIGH = 2

GATE_LOW = 3

```
class msl.equipment.resources.picotech.picoscope.enums.PS3000ASigGenTrigSource(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

NONE = 0

SCOPE_TRIG = 1

AUX_IN = 2

EXT_IN = 3

SOFT_TRIGGER = 4

```
class msl.equipment.resources.picotech.picoscope.enums.PS3000AIndexMode(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

SINGLE = 0

DUAL = 1

QUAD = 2

MAX = 3

```
class msl.equipment.resources.picotech.picoscope.enums.PS3000A_ThresholdMode(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

LEVEL = 0

WINDOW = 1

```
class msl.equipment.resources.picotech.picoscope.enums.PS3000AThresholdDirection(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

```
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(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

```
DONT_CARE = 0
```

```
TRUE = 1
```

FALSE = 2

MAX = 3

```
class msl.equipment.resources.picotech.picoscope.enums.PS3000ARatioMode(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

NONE = 0

AGGREGATE = 1

DECIMATE = 2

AVERAGE = 4

```
class msl.equipment.resources.picotech.picoscope.enums.PS3000APulseWidthType(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

NONE = 0

LESS_THAN = 1

GREATER_THAN = 2

IN_RANGE = 3

OUT_OF_RANGE = 4

```
class msl.equipment.resources.picotech.picoscope.enums.PS3000AHoldOffType(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)

Bases: IntEnum

TIME = 0

EVENT = 1

MAX = 2

class msl.equipment.resources.picotech.picoscope.enums.PS4000ChannelBufferIndex(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)

Bases: IntEnum

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(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *val-
    ues,
    mod-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

RANGES = 0

RESISTANCES = 1

ACCELEROMETER = 2

PROBES = 3

TEMPERATURES = 4

```
class msl.equipment.resources.picotech.picoscope.enums.PS4000EtsMode(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

OFF = 0

FAST = 1

SLOW = 2

MAX = 3

```
class msl.equipment.resources.picotech.picoscope.enums.PS4000TimeUnits(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

FS = 0

PS = 1

NS = 2

US = 3

MS = 4

S = 5

MAX = 6

```
class msl.equipment.resources.picotech.picoscope.enums.PS4000SweepType(value,
    names=None,
    *values,
    mod-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

UP = 0

DOWN = 1

UPDOWN = 2

DOWNUP = 3

MAX = 4

```
class msl.equipment.resources.picotech.picoscope.enums.PS4000OperationTypes(value,
    names=None,
    *val-
    ues,
    mod-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

NONE = 0

WHITENOISE = 1

PRBS = 2

```
class msl.equipment.resources.picotech.picoscope.enums.PS4000WaveType(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

```
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(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

```
RISING = 0
FALLING = 1
GATE_HIGH = 2
```

GATE_LOW = 3

```
class msl.equipment.resources.picotech.picoscope.enums.PS4000SigGenTrigSource(value,
                                                                           names=None,
                                                                           *values,
                                                                           mode=None,
                                                                           quality=None,
                                                                           name=None,
                                                                           type=None,
                                                                           start=1,
                                                                           boundary=None)
```

Bases: [IntEnum](#)

NONE = 0

SCOPE_TRIG = 1

AUX_IN = 2

EXT_IN = 3

SOFT_TRIGGER = 4

```
class msl.equipment.resources.picotech.picoscope.enums.PS4000IndexMode(value,
                                                                       names=None,
                                                                       *values,
                                                                       mode=None,
                                                                       quality=None,
                                                                       name=None,
                                                                       type=None,
                                                                       start=1,
                                                                       boundary=None)
```

Bases: [IntEnum](#)

SINGLE = 0

DUAL = 1

QUAD = 2

MAX = 3

```
class msl.equipment.resources.picotech.picoscope.enums.PS4000ThresholdMode(value,
    names=None,
    *values,
    mode=None,
    qualname=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

LEVEL = 0

WINDOW = 1

```
class msl.equipment.resources.picotech.picoscope.enums.PS4000ThresholdDirection(value,
    names=None,
    *values,
    mode=None,
    qualname=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *values,
    module=None,
    qual-name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

DONT_CARE = 0

TRUE = 1

FALSE = 2

MAX = 3

```
class msl.equipment.resources.picotech.picoscope.enums.PS4000RatioMode(value,
    names=None,
    *values,
    module=None,
    qual-name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

NONE = 0

AGGREGATE = 1

AVERAGE = 2

```
class msl.equipment.resources.picotech.picoscope.enums.PS4000PulseWidthType(value,
    names=None,
    *values,
    module=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

```
NONE = 0

LESS_THAN = 1

GREATER_THAN = 2

IN_RANGE = 3

OUT_OF_RANGE = 4
```

```
class msl.equipment.resources.picotech.picoscope.enums.PS4000Ps4000HoldOffType(value,
    names=None,
    *values,
    module=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

```
TIME = 0

MAX = 1
```

```
class msl.equipment.resources.picotech.picoscope.enums.PS4000FrequencyCounterRange(value,
    names=None,
    *values,
    module=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

FC_2K = 0

FC_20K = 1

FC_20 = 2

FC_200 = 3

FC_MAX = 4

```
class msl.equipment.resources.picotech.picoscope.enums.PS4000AExtraOperations(value,
                                                                           names=None,
                                                                           *values,
                                                                           module=None,
                                                                           qual-
                                                                           name=None,
                                                                           type=None,
                                                                           start=1,
                                                                           bound-
                                                                           ary=None)
```

Bases: [IntEnum](#)

OFF = 0

WHITENOISE = 1

PRBS = 2

```
class msl.equipment.resources.picotech.picoscope.enums.PS4000ABandwidthLimiter(value,
                                                                           names=None,
                                                                           *values,
                                                                           module=None,
                                                                           qual-
                                                                           name=None,
                                                                           type=None,
                                                                           start=1,
                                                                           bound-
                                                                           ary=None)
```

Bases: [IntEnum](#)

BW_FULL = 0

BW_20KHZ = 1

```
class msl.equipment.resources.picotech.picoscope.enums.PS4000ACoupling(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

AC = 0

DC = 1

```
class msl.equipment.resources.picotech.picoscope.enums.PS4000AChannel(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *values, mod-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *val-
    ues,
    mod-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

R_315K = 512

R_1100K = 513

R_10M = 514

R_MAX = 3

R_ADCV = 268435456

```
class msl.equipment.resources.picotech.picoscope.enums.PS4000AEtsMode(value,
    names=None,
    *values,
    module=None,
    qual-name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

OFF = 0

FAST = 1

SLOW = 2

MAX = 3

```
class msl.equipment.resources.picotech.picoscope.enums.PS4000ATimeUnits(value,
    names=None,
    *values,
    module=None,
    qual-name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

FS = 0

PS = 1

NS = 2

US = 3

MS = 4

S = 5

MAX = 6

```
class msl.equipment.resources.picotech.picoscope.enums.PS4000ASweepType(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

UP = 0

DOWN = 1

UPDOWN = 2

DOWNUP = 3

MAX = 4

```
class msl.equipment.resources.picotech.picoscope.enums.PS4000AWaveType(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *values,
    module=None,
    qual-name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

OFF = 0

RED = 1

GREEN = 2

```
class msl.equipment.resources.picotech.picoscope.enums.PS4000AMetaType(value,
    names=None,
    *values,
    module=None,
    qual-name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

UNIT_INFO = 0

DEVICE_CAPABILITY = 1

DEVICE_SETTINGS = 2

SIGNAL_GENERATOR_SETTINGS = 3

```
class msl.equipment.resources.picotech.picoscope.enums.PS4000AMetaOperation(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

READ = 0

WRITE = 1

```
class msl.equipment.resources.picotech.picoscope.enums.PS4000AMetaFormat(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

COMMA_SEPERATED = 0

XML = 1

```
class msl.equipment.resources.picotech.picoscope.enums.PS4000ASigGenTrigType(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

RISING = 0

FALLING = 1

GATE_HIGH = 2

GATE_LOW = 3

```
class msl.equipment.resources.picotech.picoscope.enums.PS4000ASigGenTrigSource(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

NONE = 0

SCOPE_TRIG = 1

AUX_IN = 2

EXT_IN = 3

SOFT_TRIG = 4

```
class msl.equipment.resources.picotech.picoscope.enums.PS4000AIndexMode(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

SINGLE = 0

DUAL = 1

QUAD = 2

MAX = 3

```
class msl.equipment.resources.picotech.picoscope.enums.PS4000AThresholdMode(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

LEVEL = 0

WINDOW = 1

```
class msl.equipment.resources.picotech.picoscope.enums.PS4000AThresholdDirection(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

DONT_CARE = 0

TRUE = 1

FALSE = 2

MAX = 3

```
class msl.equipment.resources.picotech.picoscope.enums.PS4000ASensorState(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

CONNECT_STATE_FLOATING = 0

SENSOR_STATE_CONNECTED = 1

```
class msl.equipment.resources.picotech.picoscope.enums.PS4000AFrequencyCounterRange(value,
    names=None,
    *values,
    module=None,
    qual-name=None,
    type=None,
    start=1,
    boundary=None)

Bases: IntEnum

FC_2K = 0
FC_20K = 1
FC_20 = 2
FC_200 = 3
FC_MAX = 4

class msl.equipment.resources.picotech.picoscope.enums.PS4000AConditionsInfo(value,
    names=None,
    *values,
    module=None,
    qual-name=None,
    type=None,
    start=1,
    boundary=None)

Bases: IntEnum

CLEAR = 1
ADD = 2

class msl.equipment.resources.picotech.picoscope.enums.PS4000ARatioMode(value,
    names=None,
    *values,
    module=None,
    qual-name=None,
    type=None,
    start=1,
    boundary=None)

Bases: IntEnum
```

NONE = 0

AGGREGATE = 1

DECIMATE = 2

AVERAGE = 4

DISTRIBUTION = 8

```
class msl.equipment.resources.picotech.picoscope.enums.PS4000APulseWidthType(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

NONE = 0

LESS_THAN = 1

GREATER_THAN = 2

IN_RANGE = 3

OUT_OF_RANGE = 4

```
class msl.equipment.resources.picotech.picoscope.enums.PS4000AChannelInfo(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

RANGES = 0

RESISTANCES = 1

```
class msl.equipment.resources.picotech.picoscope.enums.PS4000APicoStringValue(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

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_THRESHOLDMAJOR = 39
FREQUENCY_COUNTER_THRESHOLDMINOR = 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_REQUESTED_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(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
Bases: IntEnum
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(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

OFF = 0

FAST = 1

SLOW = 2

MAX = 3

```
class msl.equipment.resources.picotech.picoscope.enums.PS5000TimeUnits(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

FS = 0

PS = 1

NS = 2

US = 3

MS = 4

S = 5

MAX = 6

```
class msl.equipment.resources.picotech.picoscope.enums.PS5000SweepType(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

UP = 0

DOWN = 1

UPDOWN = 2

DOWNUP = 3

MAX = 4

```
class msl.equipment.resources.picotech.picoscope.enums.PS5000WaveType(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *val-
    ues,
    mod-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

RISING = 0

FALLING = 1

GATE_HIGH = 2

GATE_LOW = 3

```
class msl.equipment.resources.picotech.picoscope.enums.PS5000SigGenTrigSource(value,
    names=None,
    *val-
    ues,
    mod-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

NONE = 0

SCOPE_TRIG = 1

AUX_IN = 2

EXT_IN = 3

SOFT_TRIG = 4

```
class msl.equipment.resources.picotech.picoscope.enums.PS5000IndexMode(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

SINGLE = 0

DUAL = 1

QUAD = 2

MAX = 3

```
class msl.equipment.resources.picotech.picoscope.enums.PS5000ThresholdMode(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

LEVEL = 0

WINDOW = 1

```
class msl.equipment.resources.picotech.picoscope.enums.PS5000ThresholdDirection(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

```
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(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: `IntEnum`

```
DONT_CARE = 0
TRUE = 1
FALSE = 2
MAX = 3
```

```
class msl.equipment.resources.picotech.picoscope.enums.PS5000RatioMode(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: `IntEnum`

NONE = 0

AGGREGATE = 1

DECIMATE = 2

AVERAGE = 4

DISTRIBUTION = 8

```
class msl.equipment.resources.picotech.picoscope.enums.PS5000PulseWidthType(value,
    names=None,
    *val-
    ues,
    mode-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

NONE = 0

LESS_THAN = 1

GREATER_THAN = 2

IN_RANGE = 3

OUT_OF_RANGE = 4

```
class msl.equipment.resources.picotech.picoscope.enums.PS5000ChannelInfo(value,
    names=None,
    *val-
    ues,
    mode-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

RANGES = 0

```
class msl.equipment.resources.picotech.picoscope.enums.PS5000ADeviceResolution(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

RES_8BIT = 0

RES_12BIT = 1

RES_14BIT = 2

RES_15BIT = 3

RES_16BIT = 4

```
class msl.equipment.resources.picotech.picoscope.enums.PS5000AExtraOperations(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

OFF = 0

WHITENOISE = 1

PRBS = 2

```
class msl.equipment.resources.picotech.picoscope.enums.PS5000ABandwidthLimiter(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

BW_FULL = 0

BW_20MHZ = 1

```
class msl.equipment.resources.picotech.picoscope.enums.PS5000ACoupling(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

AC = 0

DC = 1

```
class msl.equipment.resources.picotech.picoscope.enums.PS5000AChannel(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

OFF = 0

FAST = 1

SLOW = 2

MAX = 3

```
class msl.equipment.resources.picotech.picoscope.enums.PS5000ATimeUnits(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

FS = 0

PS = 1

NS = 2

US = 3

MS = 4

S = 5

MAX = 6

```
class msl.equipment.resources.picotech.picoscope.enums.PS5000ASweepType(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

UP = 0

DOWN = 1

UPDOWN = 2

DOWNUP = 3

MAX = 4

```
class msl.equipment.resources.picotech.picoscope.enums.PS5000AWaveType(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

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(value,
                           names=None,
                           *values,
                           module=None,
                           quality=None,
                           name=None,
                           type=None,
                           start=1,
                           boundary=None)
```

Bases: [IntEnum](#)

```
RISING = 0
```

```
FALLING = 1
```

```
GATE_HIGH = 2
```

```
GATE_LOW = 3
```

```
class msl.equipment.resources.picotech.picoscope.enums.PS5000ASigGenTrigSource(value,
                           names=None,
                           *values,
                           module=None,
                           quality=None,
                           name=None,
                           type=None,
                           start=1,
                           boundary=None)
```

Bases: [IntEnum](#)

NONE = 0

SCOPE_TRIG = 1

AUX_IN = 2

EXT_IN = 3

SOFT_TRIGGER = 4

```
class msl.equipment.resources.picotech.picoscope.enums.PS5000AIndexMode(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

SINGLE = 0

DUAL = 1

QUAD = 2

MAX = 3

```
class msl.equipment.resources.picotech.picoscope.enums.PS5000AThresholdMode(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

LEVEL = 0

WINDOW = 1

```
class msl.equipment.resources.picotech.picoscope.enums.PS5000AThresholdDirection(value,
    names=None,
    *values,
    module=None,
    qualname=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: `IntEnum`

`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(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

DONT_CARE = 0

TRUE = 1

FALSE = 2

MAX = 3

```
class msl.equipment.resources.picotech.picoscope.enums.PS5000ATriggerWithinPreTrigger(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

DISABLE = 0

ARM = 1

```
class msl.equipment.resources.picotech.picoscope.enums.PS5000ARatioMode(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

NONE = 0

AGGREGATE = 1

DECIMATE = 2

AVERAGE = 4

DISTRIBUTION = 8

```
class msl.equipment.resources.picotech.picoscope.enums.PS5000APulseWidthType(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

NONE = 0

LESS_THAN = 1

GREATER_THAN = 2

IN_RANGE = 3

OUT_OF_RANGE = 4

```
class msl.equipment.resources.picotech.picoscope.enums.PS5000AChannelInfo(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

RANGES = 0

```
class msl.equipment.resources.picotech.picoscope.enums.PS6000ExternalFrequency(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

BW_FULL = 0

BW_20MHZ = 1

BW_25MHZ = 2

```
class msl.equipment.resources.picotech.picoscope.enums.PS6000Channel(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *val-
    ues,
    module=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *values,
    mod-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

AC = 0

DC_1M = 1

DC_50R = 2

```
class msl.equipment.resources.picotech.picoscope.enums.PS6000EtsMode(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

OFF = 0

FAST = 1

SLOW = 2

MAX = 3

```
class msl.equipment.resources.picotech.picoscope.enums.PS6000TimeUnits(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

FS = 0

PS = 1

NS = 2

US = 3

MS = 4

S = 5

MAX = 6

```
class msl.equipment.resources.picotech.picoscope.enums.PS6000SweepType(value,
    names=None,
    *values,
    mod-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

UP = 0

DOWN = 1

UPDOWN = 2

DOWNUP = 3

MAX = 4

```
class msl.equipment.resources.picotech.picoscope.enums.PS6000WaveType(value,
    names=None,
    *values,
    mod-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

OFF = 0

WHITENOISE = 1

PRBS = 2

```
class msl.equipment.resources.picotech.picoscope.enums.PS6000SigGenTrigType(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

RISING = 0

FALLING = 1

GATE_HIGH = 2

GATE_LOW = 3

```
class msl.equipment.resources.picotech.picoscope.enums.PS6000SigGenTrigSource(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

SINGLE = 0

DUAL = 1

QUAD = 2

MAX = 3

```
class msl.equipment.resources.picotech.picoscope.enums.PS6000ThresholdMode(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

LEVEL = 0

WINDOW = 1

```
class msl.equipment.resources.picotech.picoscope.enums.PS6000ThresholdDirection(value,
    names=None,
    *values,
    module=None,
    qualname=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: `IntEnum`

`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(value,
                           names=None,
                           *values,
                           mode=None,
                           quality=None,
                           name=None,
                           type=None,
                           start=1,
                           boundary=None)
```

Bases: [IntEnum](#)

DONT_CARE = 0

TRUE = 1

FALSE = 2

MAX = 3

```
class msl.equipment.resources.picotech.picoscope.enums.PS6000RatioMode(value,
                           names=None,
                           *values,
                           mode=None,
                           quality=None,
                           name=None,
                           type=None,
                           start=1,
                           boundary=None)
```

Bases: [IntEnum](#)

NONE = 0

AGGREGATE = 1

AVERAGE = 2

DECIMATE = 4

DISTRIBUTION = 8

```
class msl.equipment.resources.picotech.picoscope.enums.PS6000PulseWidthType(value,
    names=None,
    *values,
    module=None,
    qualname=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: `IntEnum`

`NONE = 0`

`LESS_THAN = 1`

`GREATER_THAN = 2`

`IN_RANGE = 3`

`OUT_OF_RANGE = 4`

`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(path)`

Parse a PicoScope header file.

Parameters

`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_dict)`

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(header_dict,
pi-
costa-
tus_h_path)`

Creates the _picoscopeEnums.py file

`msl.equipment.resources.picotech.picoscope.helper.create_picoscope_structs_file(header_dict)`

Creates the _picoscopeStructs.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 _picoscopeCallbacks.py file

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

`msl.equipment.resources.picotech.picoscope.helper.create_picoscope_functions_file(header_dict)`

Create the _picoscopeFunctions.py file. The lists in this file are used for creating ctypes._FuncPtr objects

`msl.equipment.resources.picotech.picoscope.helper.print_class_def_signatures(header_dict)`

`msl.equipment.resources.picotech.picoscope.helper.print_common_functions(header_dict,
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: 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(record,
func_ptrs)`

Bases: `ConnectionSDK`

Use the PicoScope SDK to communicate with the oscilloscope.

The `properties` for a PicoScope connection supports the following key-value pairs in the *Connections Database*:

```
'open': bool, Whether to open the connection synchronously [default: True]
'open_async': bool, Whether to open the connection asynchronously
↳ [default: False]
'auto_select_power': bool, For devices that can be powered by an AC
↳ adaptor or a USB cable [default: True]
'resolution': str, Only valid for ps5000a [default: '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*

Do not instantiate this class directly. Use the `connect()` method to connect to the equipment.

Parameters

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

`property handle`

Returns the handle to the SDK library.

Type
`int`

`property channel`

The information about each channel.

Type
`dict` of `PicoScopeChannel`

property dt

The time between voltage samples (i.e., delta t).

Type

`float`

property pre_trigger

The number of seconds that data was acquired for before the trigger event.

Type

`float`

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** (*PicoScopeInfoApi, PS2000Info* or *PS3000Info*, optional) – An enum value, or if `None` then request all information from the PicoScope. The enum depends on the model number of the PicoScope that you are connected to.
- **include_name** (`bool`, optional) – If `True` then includes the enum member name as a prefix. For example, returns '`CAL_DATE: 09Aug16`' if `include_name` is `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`, optional) – The number of seconds before the trigger event to start acquiring data.
- **segment_index** (`int`, optional) – Specifies which memory segment to save the data to (see manual).
- **callback** (`BlockReady`, optional) – 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`, optional) – The number of seconds before the trigger event to start acquiring data.
- **auto_stop** (`bool`, optional) – A flag that specifies if the streaming should stop when all of samples have been captured.
- **factor** (`int`, optional) – The down-sampling factor that will be applied to the raw data.
- **ratio_mode** (`enum.IntEnum`, optional) – 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`, optional) – The impedance and coupling type.
- **scale** (`enum.IntEnum`, optional) – The input voltage range.
- **offset** (`float`, optional) – 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`, optional) – The bandwidth limiter to use.
- **enabled** (`bool`, optional) – 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`, optional) – Which memory segment to save the data to.
- **oversample** (`int`, optional) – The amount of over-sample required.

Returns

- `float` – The sampling interval, i.e. dt.
- `int` – The number of samples that will be acquired.

Raises

`PicoTechError` – 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`, optional) – The time, in seconds, between the trigger occurring and the first sample.
- **direction** (`enum.IntEnum`, optional) – The direction in which the signal must move to cause a trigger.
- **timeout** (`float`, optional) – 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`, optional) – 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, which are found in the *structs* module.

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

Base class for the ps2000 and ps3000 PicoScopes.

```
class msl.equipment.resources.picotech.picoscope.picoscope_2k3k(record,  
                                         func_ptrs)
```

Bases: *PicoScope*

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

Do not instantiate this class directly. Use the [*connect\(\)*](#) method to connect to the equipment.

Parameters

- **record** (*EquipmentRecord*) – A 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\(\)](#).

**run_streaming_ns(*sample_interval*, *time_units*, *max_samples*, *auto_stop*,
no_of_samples_per_aggregate, *overview_buffer_size*)**

This function tells the scope unit to start collecting data in fast streaming mode . The function returns immediately without waiting for data to be captured. After calling this function, you should next call [get_streaming_last_values\(\)](#) to copy the data to your application's buffer.

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 `TriggerConditions` structures, which are found in the `structs` module. 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: `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 `connect()` method to connect to the equipment.

Parameters

- `record (EquipmentRecord)` – A 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`, optional) – A `ChannelInfo` enum value or enum member name.

get_max_down_sample_ratio(*num_unaggregated_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 *StreamingReady 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`, optional) – The number of samples required. If `None` then automatically determine the number of samples to retrieve.
- **start_index** (`int`, optional) – 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`, optional) – The down-sampling factor that will be applied to the raw data.
- **ratio_mode** (`enum.IntEnum`, optional) – Which down-sampling mode to use. A `RatioMode` enum.
- **segment_index** (`int`, optional) – 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** (`callback`) – A `DataReady callback` function.
- **num_samples** (`int`, optional) – The number of samples required. If `None` then automatically determine the number of samples to retrieve.
- **start_index** (`int`, optional) – 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`, optional) – The downsampling factor that will be applied to the raw data.
- **ratio_mode** (`enum.IntEnum`, optional) – Which down-sampling mode to use. A `RatioMode` enum.
- **segment_index** (`int`, optional) – 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`, optional) – A int16, numpy array. If `None` then use a pre-allocated array.
- **mode** (`enum.IntEnum`, optional) – An enum value or member name from `RatioMode`.
- **segment_index** (`int`, optional) – 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.c_longlong`) – An array of 64-bit words (`ctypes.c_int64`), 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, repetition_rate=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.

Parameters

- **waveform** (`numpy.ndarray`) – The arbitrary waveform, in volts.
- **repetition_rate** (`float`, optional) – The requested repetition rate (frequency) of the entire arbitrary waveform. The actual repetition rate that is used may be different based on the specifications of the AWG. If specified then the `sig_gen_frequency_to_phase()` method is called to determine the value of `start_delta_phase`.
- **offset_voltage** (`float`, optional) – The voltage offset, in volts, to be applied to the waveform.
- **pk_to_pk** (`float`, optional) – The peak-to-peak voltage, in volts, of the waveform signal. If `None` then uses the maximum value of the waveform to determine the peak-to-peak voltage.
- **start_delta_phase** (`int`, optional) – The initial value added to the phase accumulator as the generator begins to step through the waveform buffer.
- **stop_delta_phase** (`int`, optional) – The final value added to the phase accumulator before the generator restarts or reverses the sweep. When frequency sweeping is not required, set equal to `start_delta_phase`.
- **delta_phase_increment** (`int`, optional) – The amount added to the delta phase value every time the `dwell_count` period expires. This determines the amount by which the generator sweeps the output frequency in each dwell period. When frequency sweeping is not required, set to zero.
- **dwell_count** (`int`, optional) – The time, in units of `dacPeriod`, between successive additions of `delta_phase_increment` to the delta phase accumulator. This determines the rate at which the generator sweeps the output frequency.
- **sweep_type** (`enum.IntEnum`, optional) – 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`, optional) – The type of waveform to be produced, specified by one of the following enumerated types (B models only): OFF, WHITENOISE, PRBS
- **index_mode** (`enum.IntEnum`, optional) – Specifies how the signal will be formed from the arbitrary waveform data. Possible values are SINGLE or DUAL.
- **shots** (`int`, optional) – If `None` then start and run continuously after trigger occurs.
- **sweeps** (`int`, optional) – If `None` then start a sweep and continue after trigger occurs.
- **trigger_type** (`enum.IntEnum`, optional) – The type of trigger that will be applied to the signal generator. One of: RISING, FALLING, GATE_HIGH, GATE_LOW.
- **trigger_source** (`enum.IntEnum`, optional) – The source that will trigger the signal generator. If `None` then run without waiting for trigger.
- **ext_in_threshold** (`int`, optional) – Used to set trigger level for external trigger.

Returns

`numpy.ndarray` – The arbitrary waveform, in ADU.

Raises

`PicoTechError` – If the value of an input parameter is invalid.

```
set_sig_gen_builtin(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. Call `set_sig_gen_builtin_v2()` instead, which uses double-precision arguments.

```
set_sig_gen_builtin_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_builtin()` with double-precision frequency arguments for more precise control at low frequencies.

This function is invalid for ps4000 and ps4000a.

Parameters

- **offset_voltage** (`float`, optional) – The voltage offset, in volts, to be applied to the waveform.
- **pk_to_pk** (`float`, optional) – The peak-to-peak voltage, in volts, of the waveform signal.
- **wave_type** (`enum.IntEnum`, optional) – The type of waveform to be generated. A WaveType enum.

- **start_frequency** (`float`, optional) – The frequency that the signal generator will initially produce.
- **stop_frequency** (`float`, optional) – The frequency at which the sweep reverses direction or returns to the initial frequency.
- **increment** (`float`, optional) – The amount of frequency increase or decrease in sweep mode.
- **dwell_time** (`float`, optional) – The time, in seconds, for which the sweep stays at each frequency.
- **sweep_type** (`enum.IntEnum`, optional) – 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`, optional) – The type of waveform to be produced, specified by one of the following enumerated types (B models only): OFF, WHITENOISE, PRBS
- **shots** (`int`, optional) – If `None` then start and run continuously after trigger occurs.
- **sweeps** (`int`, optional) – If `None` then start a sweep and continue after trigger occurs.
- **trigger_type** (`enum.IntEnum`, optional) – The type of trigger that will be applied to the signal generator. One of: RISING, FALLING, GATE_HIGH, GATE_LOW.
- **trigger_source** (`enum.IntEnum`, optional) – The source that will trigger the signal generator. If `None` then run without waiting for trigger.
- **ext_in_threshold** (`int`, optional) – 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_builtin(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_sig_gen_arbitrary\(\)](#) for setting up the arbitrary waveform generator (AWG).

sig_gen_frequency_to_phase(*repetition_rate, 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

- **repetition_rate** (`float`) – The requested repetition rate (frequency) of the entire arbitrary waveform.
- **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.

Raises

`PicoTechError` – If the value of an input parameter is invalid.

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_TRIGGER.

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 `TriggerConditions` structures, which are found in the [structs](#) module, 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 `TriggerChannelProperties structs`) – A list of `TriggerChannelProperties` structures describing the requested properties.
- **timeout** (`float`, optional) – 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`, optional) – 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: PicoScope2k3k
```

A wrapper around the PicoScope ps2000 SDK.

Do not instantiate this class directly. Use the `connect()` method to connect to the equipment.

Parameters

record (`EquipmentRecord`) – A 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: *PicoScopeApi*

A wrapper around the PicoScope ps2000a SDK.

Do not instantiate this class directly. Use the `connect()` method to connect to the equipment.

Parameters

record (*EquipmentRecord*) – A 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: *PicoScope2k3k*

A wrapper around the PicoScope ps3000 SDK.

Do not instantiate this class directly. Use the *connect()* method to connect to the equipment.

Parameters

record (*EquipmentRecord*) – A 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: *PicoScopeApi*

A wrapper around the PicoScope ps3000a SDK.

Do not instantiate this class directly. Use the `connect()` method to connect to the equipment.

Parameters

`record` (*EquipmentRecord*) – A 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.

Returns

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: [PicoScopeApi](#)

A wrapper around the PicoScope ps4000 SDK.

Do not instantiate this class directly. Use the [connect\(\)](#) method to connect to the equipment.

Parameters

record ([EquipmentRecord](#)) – A 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: [PicoScopeApi](#)

A wrapper around the PicoScope ps4000a SDK.

Do not instantiate this class directly. Use the [connect\(\)](#) method to connect to the equipment.

Parameters

record ([EquipmentRecord](#)) – A 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 qualifer 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 qualifer.

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: [*PicoScopeApi*](#)

A wrapper around the PicoScope ps5000 SDK.

Do not instantiate this class directly. Use the [*connect\(\)*](#) method to connect to the equipment.

Parameters

record ([*EquipmentRecord*](#)) – A 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: *PicoScopeApi*

A wrapper around the PicoScope ps5000a SDK.

Do not instantiate this class directly. Use the `connect()` method to connect to the equipment.

Parameters

`record` (*EquipmentRecord*) – A 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

`PS5000ADeviceResolution` – 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: `PicoScopeApi`

A wrapper around the PicoScope ps6000 SDK.

Do not instantiate this class directly. Use the `connect()` method to connect to the equipment.

Parameters

`record` (`EquipmentRecord`) – A 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.
PS2000TriggerChannelProperties`

Bases: 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: 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: 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: 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: 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.  
PS2000ADigitalChannelDirections
```

Bases: Structure

channel

Structure/Union member

direction

Structure/Union member

```
class msl.equipment.resources.picotech.picoscope structs.  
PS2000ATriggerChannelProperties
```

Bases: 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.  
PS3000TriggerChannelProperties
```

Bases: 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: 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: 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: 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.PS3000ATriggerConditionsV2
    Bases: 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: 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: 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.PS3000ADigitalChannelDirections
    Bases: Structure

    channel
        Structure/Union member

    direction
        Structure/Union member

class msl.equipment.resources.picotech.picoscope.structs.PS3000ATriggerChannelProperties
    Bases: 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: 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: 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: 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.

PS4000TriggerChannelProperties

Bases: 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: Structure

channel

Structure/Union member

state

Structure/Union member

class msl.equipment.resources.picotech.picoscope structs. **PS4000ADirection**

Bases: Structure

channel

Structure/Union member

```
direction
    Structure/Union member

class msl.equipment.resources.picotech.picoscope structs.PS4000ACCondition
    Bases: Structure

condition
    Structure/Union member

source
    Structure/Union member

class msl.equipment.resources.picotech.picoscope structs.
PS4000ATriggerChannelProperties
    Bases: 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: Structure

channel
    Structure/Union member

state
    Structure/Union member

class
    msl.equipment.resources.picotech.picoscope structs.PS5000TriggerConditions
    Bases: 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: 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.
PS5000TriggerChannelProperties
Bases: 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: 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: 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: 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.
PS5000ATriggerChannelProperties

    Bases: 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: 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
```


Bases: [IntEnum](#)

The allowed data types for a PT-104 Data Logger.

OFF = 0

PT100 = 1

PT1000 = 2

RESISTANCE_TO_375R = 3

RESISTANCE_TO_10K = 4

DIFFERENTIAL_TO_115MV = 5

DIFFERENTIAL_TO_2500MV = 6

SINGLE_ENDED_TO_115MV = 7

SINGLE_ENDED_TO_2500MV = 8

MAX_DATA_TYPES = 9

```
msl.equipment.resources.picotech.pt104.enumerate_units(comm_type='all')
```

Find PT-104 Platinum Resistance Data Logger's.

This routine returns a list of all the attached PT-104 devices of the specified communication type.

Note: You cannot call this function after you have opened a connection to a Data Logger.

Parameters

comm_type ([str](#), optional) – The communication type used by the PT-104. Can be any of the following values: 'usb', 'ethernet', 'enet', 'all'

Returns

[list](#) of [str](#) – A list of serial numbers of the PT-104 Data Loggers that were found.

```
class msl.equipment.resources.picotech.pt104.PT104(record)
```

Bases: [ConnectionSDK](#)

Uses the PicoTech SDK to communicate with a PT-104 Platinum Resistance Data Logger.

The [properties](#) for a PT-104 connection supports the following key-value pairs in the [Connections Database](#):

<code>'ip_address': str</code> , The IP address and port number of the PT-104 (e.g., →'192.168.1.201:1234')
<code>'open_via_ip': bool</code> , Whether to connect to the PT-104 by Ethernet. →Default is to connect by USB.

Do not instantiate this class directly. Use the [connect\(\)](#) method to connect to the equipment.

Parameters

record (*EquipmentRecord*) – A record from an *Equipment-Register Database*.

MIN_WIRES = 2

MAX_WIRES = 4

DataType

alias of *Pt104DataType*

disconnect()

Disconnect from the PT-104 Data Logger.

get_ip_details()

Reads the IP details of the PT-104 Data Logger.

Returns

dict – The IP details.

get_unit_info(*info=None*, *include_name=True*)

Retrieves information about the PT-104 Data Logger.

If the device fails to open, or no device is opened only the driver version is available.

Parameters

- **info** (*PicoScopeInfoApi*, optional) – An enum value or member name. If *None* then request all information from the PT-104.
- **include_name** (*bool*, optional) – If *True* then includes the enum member name as a prefix. For example, returns 'CAL_DATE: 09Aug16' if *include_name* is *True* else '09Aug16'.

Returns

str – The requested information from the PT-104 Data Logger.

get_value(*channel*, *filtered=False*)

Get the most recent reading for the specified channel.

Once you open the driver and define some channels, the driver begins to take continuous readings from the PT-104 Data Logger.

The scaling of measurements is as follows:

Range	Scaling
Temperature	value * 1/1000 deg C
Voltage (0 to 2.5 V)	value * 10 nV
Voltage (0 to 115 mV)	value * 1 nV
Resistance	value * 1 mOhm

Parameters

- **channel** (*int*) – The number of the channel to read, from 1 to 4 in differential mode or 1 to 8 in single-ended mode.

- **filtered** (`bool`, optional) – If set to `True`, the driver returns a low-pass filtered value of the temperature. The time constant of the filter depends on the channel parameters as set by `set_channel()`, and on how many channels are active.

Returns

`float` – The latest reading for the specified channel.

`open()`

Open the connection to the PT-104 via USB.

`open_via_ip(address=None)`

Open the connection to the PT-104 via ETHERNET.

Parameters

`address` (`str`, optional) – The IP address and port number to use to connect to the PT-104. For example, '`192.168.1.201:1234`'. If `None` then uses the '`ip_address`' value of the `properties`

`set_channel(channel, data_type, num_wires)`

Configure a single channel of the PT-104 Data Logger.

The fewer channels selected, the more frequently they will be updated. Measurement takes about 1 second per active channel.

If a call to the `set_channel()` method has a data type of single-ended, then the specified channel's 'sister' channel is also enabled. For example, enabling 3 also enables 7.

Parameters

- **channel** (`int`) – The channel you want to set the details for. It should be between 1 and 4 if using single-ended inputs in voltage mode.
- **data_type** (`DataType`) – The type of reading you require. Can be an enum value or member name.
- **num_wires** (`int`) – The number of wires the PT100 or PT1000 sensor has (2, 3 or 4)

`set_ip_details(enabled, ip_address=None, port=None)`

Writes the IP details to the device.

Parameters

- **enabled** (`bool`) – Whether to enable or disable Ethernet communication for this device.
- **ip_address** (`str`, optional) – The new IP address. If `None` then do not change the IP address.
- **port** (`int`, optional) – The new port number. If `None` then do not change the port number.

`set_mains(hertz)`

Inform the driver of the local mains (line) frequency.

This helps the driver to filter out electrical noise.

Parameters

hertz (`int`) – Either 50 or 60.

`msl.equipment.resources.princeton_instruments package`

Wrapper around the `ARC_Instrument.dll` SDK from Princeton Instruments.

Submodules

`msl.equipment.resources.princeton_instruments.arc_instrument module`

Wrapper around the `ARC_Instrument.dll` SDK from Princeton Instruments.

The wrapper was written using v2.0.3 of the SDK.

`class msl.equipment.resources.princeton_instruments.arc_instrument.PrincetonInstruments(record)`

Bases: `Connection`

Wrapper around the `ARC_Instrument.dll` SDK from Princeton Instruments.

The `properties` for a ARCInstrument connection supports the following key-value pairs in the `Connections Database`:

```
'sdk_path': str, The path to the SDK [default: 'ARC_Instrument_x64.dll']  
'open': bool, Whether to automatically open the connection [default: True]
```

Do not instantiate this class directly. Use the `connect()` method to connect to the equipment.

Parameters

`record` (`EquipmentRecord`) – A record from an `Equipment-Register Database`.

disconnect()

Close all open connections.

static close_enum(enum)

Function to close an open enumeration.

Parameters

`enum` (`int`) – A handle defined in `ARC_Open_xxxx` by which the instrument is to be addressed.

det_read(det_num)

Readout a single detector.

Parameters

`det_num` (`int`) – The detector to be addressed.

Returns

`float` – Reading value for the selected detector.

det_readall()

Readout all detectors.

Returns

`tuple` of `float` – The reading of each detector.

det_start_nonblock_read(*det_num*)

Start to readout a single detector, but return immediately (non-blocking read).

Parameters

`det_num` (`int`) – The detector to be addressed.

det_nonblock_read_done(*det_num*)

Returns the detector value.

Parameters

`det_num` (`int`) – The detector to be addressed.

Returns

`float` – The value of the detector.

filter_home()

Homes the filter wheel.

mono_filter_home()

Homes the filter wheel.

mono_grating_calc_gadjust(*grating, wave, ref_wave*)

Calculate a new grating GAdjust.

Parameters

- `grating` (`int`) – The number of the grating that is to be addressed.
- `wave` (`float`) – The wavelength, in nm, on which a peak is currently falling.
- `ref_wave` (`float`) – The wavelength, in nm, on which the peak should fall.

Returns

`int` – The calculated grating GAdjust. Note - this value is specific to a specific instrument-grating combination and not transferable between instruments.

mono_grating_calc_offset(*grating, wave, ref_wave*)

Calculate a new grating offset.

Parameters

- `grating` (`int`) – The number of the grating that is to be addressed.
- `wave` (`float`) – The wavelength, in nm, on which a peak is currently falling.
- `ref_wave` (`float`) – The wavelength, in nm, on which the peak should fall.

Returns

`int` – The calculated grating offset. Note - this value is specific to a specific instrument-grating combination and not transferable between instruments.

mono_grating_install(*grating*, *density*, *blaze*, *nm_blaze*, *um_blaze*, *hol_blaze*, *mirror*, *reboot*)

Install a new grating.

Parameters

- **grating** (`int`) – The number of the grating that is to be addressed.
- **density** (`int`) – The groove density in grooves per mm of the grating.
- **blaze** (`str`) – The Blaze string.
- **nm_blaze** (`bool`) – Whether the grating has a nm blaze.
- **um_blaze** (`bool`) – Whether the grating has a um blaze.
- **hol_blaze** (`bool`) – Whether the grating has a holographic blaze.
- **mirror** (`bool`) – Whether the grating position is a mirror.
- **reboot** (`bool`) – Whether to reboot the monochromator after installing.

mono_grating_uninstall(*grating*, *reboot*)

Uninstall a grating.

Parameters

- **grating** (`int`) – The number of the grating that is to be uninstalled.
- **reboot** (`bool`) – Whether to reboot the monochromator after uninstalling.

mono_move_steps(*num_steps*)

Move the grating a set number of steps.

Parameters

num_steps (`int`) – Number of steps to move the wavelength drive.

mono_reset()

Reset the monochromator.

mono_restore_factory_settings()

Restore the instrument factory settings.

mono_scan_done()

Check if a scan has completed.

Returns

- `bool` – Whether the scan finished (the scan ended or the grating wavelength limit was reached).
- `float` – The current wavelength, in nm, of the grating.

mono_slit_home(*slit_num*)

Homes a motorized slit.

Parameters

slit_num (`int`) – The slit to be homed.

- 1 - Side entrance slit.

- 2 - Front entrance slit.
- 3 - Front exit slit.
- 4 - Side exit slit.
- 5 - Side entrance slit on a double slave unit.
- 6 - Front entrance slit on a double slave unit.
- 7 - Front exit slit on a double slave unit.
- 8 - Side exit slit on a double slave unit.

static mono_slit_name(slit_num)

Returns a text description of a slit position.

Parameters

slit_num (`int`) – The slit to be addressed.

Returns

`str` – The description of the slit port location.

mono_start_jog(jog_max_rate, jog_forwards)

Start jogging the wavelength of the monochromator.

Parameters

- **jog_max_rate** (`bool`) – Whether to jog at the max scan rate or at the current scan rate.
 - `False` - Current Scan Rate.
 - `True` - Max Scan Rate.
- **jog_forwards** (`bool`) – Whether to jog forward (increasing nm) or backwards (decreasing nm).
 - `False` - Jog Backwards (decreasing nm).
 - `True` - Jog Forwards (increasing nm).

mono_start_scan_to_nm(wavelength_nm)

Start a wavelength scan.

Parameters

wavelength_nm (`float`) – Wavelength in nm we are to scan to. Note - the value should not be lower than the current wavelength and should not exceed the current grating wavelength limit.

mono_stop_jog()

End a wavelength jog.

calc_mono_slit_bandpass(slit_num, width)

Calculates the band pass for the provided slit width.

Parameters

- **slit_num** (`int`) – The slit number.
 - 1 - Side entrance slit.
 - 2 - Front entrance slit.

- 3 - Front exit slit.
 - 4 - Side exit slit.
 - 5 - Side entrance slit on a double slave unit.
 - 6 - Front entrance slit on a double slave unit.
 - 7 - Front exit slit on a double slave unit.
 - 8 - Side exit slit on a double slave unit.
- **width (float)** – The slit width that the band pass is being calculated for.

Returns

float – The calculated band pass for the slit, in nm.

ncl_filter_home()

Home the filter wheel.

open_filter(enum_num)

Function to open a Filter Wheel.

Parameters

enum_num (int) – Number in the enumeration list to open.

open_mono(enum_num)

Function to open a Monochromator.

Parameters

enum_num (int) – Number in the enumeration list to open.

open_mono_port(port)

Function to open a Monochromator on a specific COM Port.

Parameters

port (int or str) – The COM port (e.g., 5 or 'COM5').

open_mono_serial(serial)

Function to open a Monochromator with a specific serial number.

Parameters

serial (class:int or str) – The serial number of the Monochromator.

open_filter_port(port)

Function to open a Filter Wheel on a specific COM Port.

Parameters

port (int or str) – The COM port (e.g., 5 or 'COM5').

open_filter_serial(serial)

Function to open a Filter Wheel with a specific serial number.

Parameters

serial (class:int or str) – The serial number of the Filter Wheel.

open_readout(port_num)

Function to open a Readout System (NCL/NCL-Lite).

Parameters

port_num (`int`) – Number in the enumeration list to open.

Returns

- `int` – If a mono is attached to an NCL on port 1, then this is the enum by which to address the first mono.
- `int` – If a mono is attached to an NCL on port 2, then this is the enum by which to address the second mono.

open_readout_port(*port*)

Function to open a Readout System (NCL/NCL-Lite) on a specific COM Port.

Parameters

port (class:`int` or `str`) – The COM port (e.g., 5 or 'COM5').

Returns

- `int` – If a mono is attached to an NCL on port 1, then this is the enum by which to address the first mono.
- `int` – If a mono is attached to an NCL on port 2, then this is the enum by which to address the second mono.

static search_for_inst()

Function used to search for all attached Acton Research Instruments.

Returns

`int` – The number of Acton Research Instruments found and enumerated. Enumeration list starts with zero and ends with the “number found” minus one.

valid_det_num(*det_num*)

Check if the detector number is valid on the Readout System.

Parameters

det_num (`int`) – The detector number to check.

Returns

`bool` – Whether the detector number is valid on the Readout System.

static valid_enum(*enum*)

Check if an enumeration is valid and the instrument is open.

Parameters

enum (`int`) – The enumeration to check.

Returns

`bool` – Whether the enumeration is valid and the instrument is open.

static valid_mono_enum(*mono_enum*)

Check if a monochromator enumeration is valid and the instrument is open.

All functions call this function to verify that the requested action is valid.

Parameters

mono_enum (`int`) – The enumeration to check.

Returns

`bool` – Whether the enumeration is valid and the instrument is open.

static valid_readout_enum(readout_enum)

Check if a Readout System (NCL/NCL-Lite) enumeration is valid and the instrument is open.

Parameters

`readout_enum (int)` – The enumeration to check.

Returns

`bool` – Whether the enumeration is valid and the instrument is open.

static valid_filter_enum(filter_enum)

Check if a Filter Wheel enumeration is valid and the instrument is open.

Parameters

`filter_enum (int)` – The enumeration to check.

Returns

`bool` – Whether the enumeration is valid and the instrument is open.

static ver()

Get the DLL version number.

It is the only function that can be called at any time.

Returns

- `int` – The major version number.
- `int` – The minor version number.
- `int` – The build version number.

get_det_bipolar(det_num)

Return if a detector takes bipolar (+/-) readings.

Parameters

`det_num (int)` – The detector to be addressed.

Returns

`bool` – `True` if the detector is bipolar (+/-).

get_det_bipolar_str(det_num)

Return the description of the detector polarity.

Parameters

`det_num (int)` – The detector to be addressed.

Returns

`str` – A description of whether the detector is uni-polar or bipolar.

get_det_hv_volts(det_num)

Return the high voltage volts setting.

Parameters

`det_num (int)` – The detector to be addressed.

Returns

`int` – High voltage volts that the detector is set to.

get_det_hv_on(*det_num*)

Return if the high voltage for a detector is turned on.

Parameters

det_num (`int`) – The detector to be addressed.

Returns

`bool` – Whether the high voltage is turned on.

get_det_num_avg_read()

Return the number of readings that are averaged.

Returns

`int` – Number of readings averaged into a single reading.

get_det_range(*det_num*)

Return the detector range factor.

Parameters

det_num (`int`) – The detector to be addressed.

Returns

`int` – The detector gain range.

- Detector Range 0 - 1x
- Detector Range 1 - 2x
- Detector Range 2 - 4x
- Detector Range 3 - 50x
- Detector Range 4 - 200x

get_det_range_factor(*det_num*)

Return the detector range multiplier.

Parameters

det_num (`int`) – The detector to be addressed.

Returns

`int` – The detector range multiplier.

get_det_type(*det_num*)

Return the detector readout type (Current, Voltage, Photon Counting).

Parameters

det_num (`int`) – The detector to be addressed.

Returns

`int` –

The detector readout type.

- Detector Type 1 - Current
- Detector Type 2 - Voltage
- Detector Type 3 - Photon Counting

`get_det_type_str(det_num)`

Return a description of the detector readout type.

Parameters

`det_num` (`int`) – The detector to be addressed.

Returns

`str` – The detector readout type (Current, Voltage, Photon Counting).

`get_filter_max_pos()`

Returns the maximum filter position.

Returns

`int` – Returns the maximum position possible with the filter wheel.

`get_filter_min_pos()`

Returns the minimum filter position.

Returns

`int` – Returns the minimum position possible with the filter wheel.

`get_filter_model()`

Returns the model string from the instrument.

Returns

`str` – The model string of the instrument.

`get_filter_position()`

Returns the current filter position.

Returns

`int` – The current filter wheel position.

`get_filter_present()`

Returns if the instrument has a filter wheel.

Returns

`int` – Whether an integrated filter wheel is present on the filter wheel.

`get_filter_serial()`

Returns the serial number of the instrument.

Returns

`str` – The serial number of the instrument.

`static get_filter_preopen_model(filter_enum)`

Returns the model string of an instrument not yet opened.

Note, `search_for_inst()` needs to be called prior to this call. In the case of multiple Filter Wheel/Spectrographs attached, it allows the user to sort, which instruments are to be opened before opening them.

Parameters

`filter_enum` (`int`) – A filter enumeration value.

Returns

`str` – The model string of the unopened filter wheel.

static get_enum_preopen_model(enum)

Returns the model number of an instrument not yet opened.

Note, [search_for_inst\(\)](#) needs to be called prior to this call.

Parameters

enum (int) – The enumeration for the unopened instrument.

Returns

str – A Princeton Instruments model number.

static get_enum_preopen_serial(enum)

Returns the serial number of an instrument not yet opened.

Note, [search_for_inst\(\)](#) needs to be called prior to this call.

Parameters

enum (int) – The enumeration for the unopened instrument.

Returns

str – A Princeton Instruments serial number.

static get_enum_preopen_com(enum)

Returns the COM port number of an instrument not yet opened.

Note, [search_for_inst\(\)](#) needs to be called prior to this call.

Parameters

enum (int) – The enumeration for the unopened instrument.

Returns

int – The COM port number.

get_mono_backlash_steps()

Returns the number of backlash steps used when reversing the wavelength drive.

Returns

int – The number of steps the instrument backlash corrects. Not valid on older instruments.

get_mono_detector_angle()

Returns the default detector angle of the instrument in radians.

Returns

float – The default detector angle of the instrument in radians.

get_mono_diverter_pos(diverter_num)

Returns the slit that the diverter mirror is pointing to.

Parameters

diverter_num (int) – The diverter to be queried.

- Mirror Position 1 - motorized entrance diverter mirror.
- Mirror Position 2 - motorized exit diverter mirror.
- Mirror Position 3 - motorized entrance diverter on a double slave unit.
- Mirror Position 4 - motorized exit diverter on a double slave unit.

Returns

`int` –

The slit port that the diverter mirror is currently pointing at.

- Slit Port 1 - Side entrance slit.
- Slit Port 2 - Front entrance slit.
- Slit Port 3 - Front exit slit.
- Slit Port 4 - Side exit slit.
- Slit Port 5 - Side entrance slit on a double slave unit.
- Slit Port 6 - Front entrance slit on a double slave unit.
- Slit Port 7 - Front exit slit on a double slave unit.
- Slit Port 8 - Side exit slit on a double slave unit.

get_mono_diverter_pos_str(*diverter_num*)

Returns a string describing the port the mirror is pointing to.

Parameters

`diverter_num` (`int`) – The diverter to be queried.

- Mirror Position 1 - motorized entrance diverter mirror.
- Mirror Position 2 - motorized exit diverter mirror.
- Mirror Position 3 - motorized entrance diverter on a double slave unit.
- Mirror Position 4 - motorized exit diverter on a double slave unit.

Returns

`str` – A string describing which port the diverter is pointing at.

get_mono_diverter_valid(*diverter_num*)

Returns if a motorized diverter position is valid for an instrument.

Parameters

`diverter_num` (`int`) – The diverter to be queried.

- Mirror Position 1 - motorized entrance diverter mirror.
- Mirror Position 2 - motorized exit diverter mirror.
- Mirror Position 3 - motorized entrance diverter on a double slave unit.
- Mirror Position 4 - motorized exit diverter on a double slave unit.

Returns

`bool` – Whether the diverter mirror is valid.

get_mono_double()

Returns if the instrument is a double monochromator.

Returns

`bool` – Whether the instrument is a double monochromator.

get_mono_double_subtractive()

Returns if a double monochromator is subtractive instead of additive.

Returns

`bool` – Whether the double monochromator is subtractive.

get_mono_double_intermediate_slit()

If a monochromator is a double, return the intermediate slit position.

Returns

`slit_pos (int)` – The intermediate slit of the double monochromator

- Slit Port 1 - Side entrance slit.
- Slit Port 2 - Front entrance slit.
- Slit Port 3 - Front exit slit.
- Slit Port 4 - Side exit slit.
- Slit Port 5 - Side entrance slit on a double slave unit.
- Slit Port 6 - Front entrance slit on a double slave unit.
- Slit Port 7 - Front exit slit on a double slave unit.
- Slit Port 8 - Side exit slit on a double slave unit.

get_mono_exit_slit()

Return the exit slit position for a monochromator.

Function works with single and double monochromators.

Returns

`slit_pos (int)` – The intermediate slit of the double monochromator

- Slit Port 1 - Side entrance slit.
- Slit Port 2 - Front entrance slit.
- Slit Port 3 - Front exit slit.
- Slit Port 4 - Side exit slit.
- Slit Port 5 - Side entrance slit on a double slave unit.
- Slit Port 6 - Front entrance slit on a double slave unit.
- Slit Port 7 - Front exit slit on a double slave unit.
- Slit Port 8 - Side exit slit on a double slave unit.

get_mono_filter_max_pos()

Returns the maximum filter position.

Returns

`int` – The maximum position possible with the filter wheel.

get_mono_filter_min_pos()

Returns the minimum filter position.

Returns

`int` – The minimum position possible with the filter wheel.

get_mono_filter_position()

Returns the current filter position.

Returns

`int` – The current filter wheel position.

get_mono_filter_present()

Returns if the instrument has an integrated filter wheel.

Note, is a new option and not available on most instruments. All filter functions call this function before proceeding.

Returns

`bool` – Whether an integrated filter wheel is present on the monochromator.

get_mono_focal_length()

Returns the default focal length of the instrument.

Returns

`float` – The focal length of the instrument in millimeters.

get_mono_gear_steps()

Returns the number of steps in a set of sine drive gears.

Returns

- `int` – The number of steps per rev on the minor gear of a sine wavelength drive.
- `int` – The number of steps per rev on the major gear of a sine wavelength drive.

get_mono_grating()

Returns the current grating.

Returns

`int` – The current grating. This assumes the correct turret has been inserted in the instrument.

get_mono_grating_blaze(*grating*)

Returns the blaze of a given grating.

Parameters

grating (`int`) – Which grating to request the information about. Validates the request by calling `get_mono_grating_installed()`.

Returns

`str` – The blaze of the grating.

get_mono_grating_density(*grating*)

Returns the groove density (grooves per millimeter) of a given grating.

Parameters

grating (`int`) – Which grating to request the information about. Validates the request by calling `get_mono_grating_installed()`.

Returns

`int` – The groove density of the grating in grooves per millimeter. For a mirror, this function will return 1200 grooves per MM.

get_mono_grating_gadjust(*grating*)

Returns the GAdjust of a grating.

Parameters

`grating (int)` – Which grating to request the information about.

Returns

`int` – The grating GAdjust. Note, this value is specific to a specific instrument grating combination and not transferable between instruments.

get_mono_grating_installed(*grating*)

Returns if a grating is installed.

Parameters

`grating (int)` – Which grating we are requesting the information about.

Returns

`bool` – Whether the grating requested is installed.

get_mono_grating_max()

Get the maximum grating position installed.

This is usually the number of gratings installed.

Returns

`int` – The number of gratings installed.

get_mono_grating_offset(*grating*)

Returns the offset of a grating.

Parameters

`grating (int c_long)` – The number of the grating that is to be addressed.

Returns

`int` – The grating offset. Note, this value is specific to a specific instrument grating combination and not transferable between instruments.

get_mono_half_angle()

Returns the default half angle of the instrument in radians.

:param `float`: The half angle of the instrument in radians.

get_mono_init_grating()

Returns the initial grating (on instrument reboot).

Returns

`int` – The power-up grating number.

get_mono_init_scan_rate_nm()

Returns the initial wavelength scan rate (on instrument reboot).

Returns

`float` – The power-up scan rate in nm / minute.

get_mono_init_wave_nm()

Returns the initial wavelength (on instrument reboot).

Returns

`float` – The power-up wavelength in nm.

get_mono_int_led_on()

Checks if the interrupter LED is on.

Returns

`bool` – Whether the interrupter LED is on.

get_mono_model()

Returns the model number of the instrument.

Returns

`str` – The model number of the instrument.

get_mono_motor_int()

Read the motor gear interrupter.

Returns

`bool` – Whether the motor gear was interrupted.

get_mono_precision()

Returns the nm-decimal precision of the wavelength drive.

Note, this is independent of the wavelength step resolution whose coarseness is defined by the grating being used.

Returns

`int` – The number of digits after the decimal point the instrument uses. Note, the true precision is limited by the density of the grating and can be much less than the instruments wavelength precision.

get_mono_scan_rate_nm_min()

Return the current wavelength scan rate.

Returns

`float` – Scan rate in nm per minute.

get_mono_serial()

Returns the serial number of the instrument.

Returns

`str` – The serial number of the instrument as a string.

get_mono_shutter_open()

Returns if the integrated shutter is open.

Returns

`bool` – Whether the shutter is open.

get_mono_shutter_valid()

Returns if the instrument has an integrated shutter.

Returns

`bool` – Whether a shutter is present.

get_mono_sine_drive()

Returns if the gearing system has sine instead of a linear drive system.

Returns

`bool` – Whether the gearing is sine based verses linear gearing.

get_mono_slit_type(*slit_pos*)

Returns the slit type for a given slit.

Parameters

`slit_pos (int)` – The slit to be addressed.

- Slit Port 1 - Side entrance slit.
- Slit Port 2 - Front entrance slit.
- Slit Port 3 - Front exit slit.
- Slit Port 4 - Side exit slit.
- Slit Port 5 - Side entrance slit on a double slave unit.
- Slit Port 6 - Front entrance slit on a double slave unit.
- Slit Port 7 - Front exit slit on a double slave unit.
- Slit Port 8 - Side exit slit on a double slave unit.

Returns

`int` – The type of slit attached.

- Slit Type 0 - No slit, older instruments that do will only return a slit type of zero.
- Slit Type 1 - Manual Slit.
- Slit Type 2 - Fixed Slit Width.
- Slit Type 3 - Focal Plane adapter.
- Slit Type 4 - Continuous Motor.
- Slit Type 5 - Fiber Optic.
- Slit Type 6 - Index able Slit.

get_mono_slit_type_str(*slit_pos*)

Returns a string descriptor of a given slit.

Parameters

`slit_pos (int)` – The slit to be addressed.

- Slit Port 1 - Side entrance slit.
- Slit Port 2 - Front entrance slit.
- Slit Port 3 - Front exit slit.
- Slit Port 4 - Side exit slit.
- Slit Port 5 - Side entrance slit on a double slave unit.
- Slit Port 6 - Front entrance slit on a double slave unit.

- Slit Port 7 - Front exit slit on a double slave unit.
- Slit Port 8 - Side exit slit on a double slave unit.

Returns

`str` – The description of the slit.

get_mono_slit_width(*slit_pos*)

Returns the slit width of a motorized slit.

Parameters

`slit_pos (int)` – The slit to be addressed.

- Slit Port 1 - Side entrance slit.
- Slit Port 2 - Front entrance slit.
- Slit Port 3 - Front exit slit.
- Slit Port 4 - Side exit slit.
- Slit Port 5 - Side entrance slit on a double slave unit.
- Slit Port 6 - Front entrance slit on a double slave unit.
- Slit Port 7 - Front exit slit on a double slave unit.
- Slit Port 8 - Side exit slit on a double slave unit.

Returns

`int` – The width of the slit, if the slit is motorized.

get_mono_slit_width_max(*slit_pos*)

Returns the maximum width of a motorized slit.

Parameters

`slit_pos (int)` – The slit to be addressed.

- Slit Port 1 - Side entrance slit.
- Slit Port 2 - Front entrance slit.
- Slit Port 3 - Front exit slit.
- Slit Port 4 - Side exit slit.
- Slit Port 5 - Side entrance slit on a double slave unit.
- Slit Port 6 - Front entrance slit on a double slave unit.
- Slit Port 7 - Front exit slit on a double slave unit.
- Slit Port 8 - Side exit slit on a double slave unit.

Returns

`int` – The maximum width of the slit, if the slit is motorized.

get_mono_turret()

Returns the current grating turret number.

Returns

`int` – The current turret number.

get_mono_turret_max()

Returns the number of turrets installed.

Returns

`int` – The number of turrets installed.

get_mono_turret_gratings()

Returns the number of gratings per turret.

Returns

`int` – The number of gratings that can be placed on a single turret. This number can be one, two or three depending on the monochromator model.

get_mono_wavelength_cutoff_nm()

Returns, in nm, the max wavelength achievable by the instrument using the current grating.

Returns

`float` – The maximum center wavelength in nanometers for the current grating. On SpectraPro's this value equals $1400 \text{ nm} * \text{grating density} / 1200 \text{ g/mm}$. On AM/VM products, this value is limited by the sine bar and will vary.

get_mono_wavelength_min_nm()

Returns, in nm, the min wavelength achievable by the instrument using the current grating.

Returns

`float` – The minimum center wavelength in nanometers for the current grating. On SpectraPro's it is usually -10 nm, on linear AM/VM instruments this value will vary.

get_mono_wavelength_abs_cm()

Returns the current center wavelength of instrument in absolute wavenumber.

Returns

`float` – The current wavelength of the instrument in absolute wavenumber.

get_mono_wavelength_ang()

Returns the current center wavelength of instrument in Angstroms.

Returns

`float` – The current wavelength of the instrument in Angstroms.

get_mono_wavelength_ev()

Returns the current center wavelength of instrument in electron volts.

Returns

`float` – The current wavelength of the instrument in electron volts.

get_mono_wavelength_micron()

Returns the current center wavelength of instrument in microns.

Returns

`float` – The current wavelength of the instrument in microns.

get_mono_wavelength_nm()

Returns the current center wavelength of instrument in nm.

Returns

`float` – The current wavelength of the instrument in nanometers.

get_mono_wavelength_rel_cm(*center_nm*)

Returns the current center wavelength of instrument in relative wavenumber.

Parameters

`center_nm (int)` – The wavelength in nanometers that 0 relative wavenumber is centered around.

Returns

`float` – The current wavelength of the instrument in relative wavenumber.

get_mono_wheel_int()

Read the wheel gear interrupter.

Returns

`bool` – Whether the wheel gear was interrupted.

get_mono_nm_rev_ratio()

Returns the number of stepper steps per rev of the wavelength drive motor.

Returns

`float` – The ratio of nm per rev in a linear wavelength drive.

static get_mono_preopen_model(*mono_enum*)

Returns the model of an instrument not yet opened.

Note, `search_for_inst()` needs to be called prior to this call. In the case of multiple Monochromator/Spectrographs attached, it allows the user to sort, which instruments are to be opened before opening them.

Parameters

`mono_enum (int)` – A monochromator enumeration.

Returns

`str` – The model of the unopened monochromator.

get_ncl_filter_max_pos()

Returns the maximum filter position.

Returns

`int` – The maximum filter position.

get_ncl_filter_min_pos()

Returns the minimum filter position.

Returns

`int` – The minimum filter position.

get_ncl_filter_position()

Return the current filter position.

Returns

`int` – The current filter position.

get_ncl_filter_present()

Returns if the instrument has an integrated filter wheel setup.

Returns

`bool` – Whether a NCL filter wheel is setup and present.

get_ncl_mono_enum(*mono_num*)

Return the enumeration of the attached monochromator.

Parameters

`mono_num` (`int`) – The readout system monochromator port being addressed.

Returns

`int` – The enumeration of the monochromator. Check the value returned with `valid_mono_enum()`.

get_ncl_mono_setup(*mono_num*)

Return if the open NCL port has a Mnnochromator attached.

Parameters

`mono_num` (`int`) – The readout system monochromator port being addressed.

Returns

`bool` – Whether a monochromator is attached to the port and it is set up.

get_ncl_shutter_open(*shutter_num*)

Return if a NCL Readout shutter is open.

Parameters

`shutter_num` (`int`) – The shutter being addressed (1 or 2 on an NCL).

Returns

`bool` – Whether the shutter is in the open position.

get_ncl_shutter_valid(*shutter_num*)

Return if a NCL Readout shutter number is valid.

Parameters

`shutter_num` (`int`) – The shutter being addressed (1 or 2 on an NCL).

Returns

`bool` – Whether the shutter number is valid.

get_ncl_ttl_in(*ttl_line*)

Return if an input TTL line is being pulled on by an outside connection.

Parameters

`ttl_line` (`int`) – The TTL line number being addressed.

Returns

`bool` – Whether *ttl_line* is being pulled to TTL high.

get_ncl_ttl_out(*ttl_line*)

Return if an output TTL line is on.

Parameters

`ttl_line` (`int`) – The TTL line number being addressed.

Returns

`bool` – Whether the output *ttl_line* is set to TTL high.

get_ncl_ttl_valid(*ttl_line*)

Return if a TTL line is a valid line.

Parameters

ttl_line (`int`) – The TTL line number being addressed.

Returns

`bool` – Whether *ttl_line* is a valid line number.

get_num_det()

Return the number of single point detectors in the Readout System.

Returns

`int` – Number of single point detectors the readout system supports. (1 NCL-Lite, 2,3 NCL).

static get_num_found_inst_ports()

Returns the value last returned by `search_for_inst()`.

Can be called multiple times.

Returns

`int` – The number of instruments found.

get_readout_itime_ms()

Return the integration time used for taking a reading.

Returns

`int` – The integration time used for taking a reading, in milliseconds.

get_readout_model()

Returns the model string from the instrument.

:param `str`: The model string of the instrument.

get_readout_serial()

Returns the serial number from the instrument.

Returns

`str` – The serial number of the instrument as a string.

static get_readout_preopen_model(*enum*)

Returns the model string of an instrument not yet opened.

Note, “meth:`search_for_inst` needs to be called prior to this call. In the case of multiple Readout Systems attached, it allows the user to sort, which instruments are to be opened before opening them.

Parameters

`enum` (`int`) – The instrument enumeration.

Returns

`str` – The model string of the unopened instrument.

set_det_bipolar(*det_num*)

Set the detector readout to bipolar (+/-).

Parameters

`det_num` (`int`) – The detector to be addressed.

set_det_current(*det_num*)

Set the detector readout type to Current.

Parameters

• **det_num** (`int`) – The detector to be addressed.

set_det_hv_volts(*det_num*, *hv_volts*)

Set the voltage value of the high voltage.

Parameters

- **det_num** (`int`) – The detector to be addressed.
- **hv_volts** (`int`) – The voltage value.

set_det_hv_off(*det_num*)

Set the detector high voltage to off.

Parameters

• **det_num** (`int` c_long) – The detector to be addressed.

Returns

`bool` – Whether the high voltage has been turned off.

set_det_hv_on(*det_num*)

Set the detector high voltage to on.

Parameters

• **det_num** (`int`) – The detector to be addressed.

Returns

`bool` – Whether the high voltage has been turned on.

set_det_num_avg_read(*num_reads_avg*)

Set the number of readings to average.

Parameters

• **num_reads_avg** (`int`) – The number reads that will be required to acquire a single read.

set_det_photon(*det_num*)

Set the detector readout type to Photon Counter.

Parameters

• **det_num** (`int`) – The detector to be addressed.

set_det_range(*det_num*, *gain_range*)

Set the detector gain-range factor.

Parameters

- **det_num** (`int`) – The detector to be addressed.
- **gain_range** (`int`) – The detector gain range.
 - Detector Range 0 - 1x
 - Detector Range 1 - 2x
 - Detector Range 2 - 4x

- Detector Range 3 - 50x
- Detector Range 4 - 200x

set_det_type(*det_num*, *det_type*)

Set the detector readout type.

Parameters

- **det_num** (`int`) – The detector to be addressed.
- **det_type** (`int`) – The detector readout type.
 - Detector Type 1 - Current
 - Detector Type 2 - Voltage
 - Detector Type 3 - Photon Counting

set_det_unipolar(*det_num*)

Set the detector readout to unipolar (-).

Parameters

det_num (`int`) – The detector to be addressed.

set_det_voltage(*det_num*)

Set the detector readout type to voltage.

Parameters

det_num (`int`) – The detector to be addressed.

set_filter_position(*position*)

Sets the current filter position.

Parameters

position (`int`) – Position the filter is to be set to.

set_mono_diverter_pos(*diverter_num*, *diverter_pos*)

Sets the port that the unit is to point to.

Parameters

- **diverter_num** (`int`) – The diverter to be modified.
 - Mirror Position 1 - motorized entrance diverter mirror.
 - Mirror Position 2 - motorized exit diverter mirror.
 - Mirror Position 3 - motorized entrance diverter on a double slave unit.
 - Mirror Position 4 - motorized exit diverter on a double slave unit.
- **diverter_pos** (`int`) – The slit port that the diverter mirror is to be set to. Not all ports are valid for all diverters.
 - Slit Port 1 - Side entrance slit.
 - Slit Port 2 - Front entrance slit. Slit Port 3 - Front exit slit.
 - Slit Port 4 - Side exit slit.
 - Slit Port 5 - Side entrance slit on a double slave unit.
 - Slit Port 6 - Front entrance slit on a double slave unit.

- Slit Port 7 - Front exit slit on a double slave unit.
- Slit Port 8 - Side exit slit on a double slave unit.

set_mono_filter_position(*position*)

Sets the current filter position.

Parameters

position (`int`) – Position the filter is to be set to.

set_mono_grating(*grating*)

Sets the current grating.

Parameters

grating (`int`) – Set the current grating. This assumes the correct turret is inserted in the instrument and grating selected is a valid grating. This function will change to current turret in the firmware, but needs user interaction to install the correct turret.

set_mono_grating_gadjust(*grating, gadjust*)

Sets the grating GAdjust.

Parameters

- **grating** (`int`) – The number of the grating that is to be addressed.
- **gadjust** (`int`) – The grating GAdjust. Note, this value is specific to a specific instrument-grating combination and not transferable between instruments.

set_mono_grating_offset(*grating, offset*)

Sets the grating offset.

Parameters

- **grating** (`int`) – The number of the grating that is to be addressed.
- **offset** (`int`) – The grating offset. Note, this value is specific to a specific instrument-grating combination and not transferable between instruments.

set_mono_init_grating(*init_grating*)

Sets the initial grating (on instrument reboot).

Parameters

init_grating (`int`) – The power-up grating number.

set_mono_init_scan_rate_nm(*init_scan_rate*)

Sets the initial wavelength scan rate (on instrument reboot).

Parameters

init_scan_rate (`float`) – The power-up scan rate in nm / minute.

set_mono_init_wave_nm(*init_wave*)

Sets the initial wavelength (on instrument reboot).

Parameters

init_wave (`float`) – The power-up wavelength in nm.

set_mono_int_led(*led_state*)

Turns on and off the interrupter LED.

Parameters

led_state (`bool`) – Indicates whether the LED should be on or off.

set_mono_scan_rate_nm_min(*scan_rate*)

Set the wavelength scan rate.

Parameters

scan_rate (`float`) – The rate to scan the wavelength in nm/minute.

set_mono_shutter_closed()

Sets the integrated shutter position to closed.

Returns

`bool` – Whether the shutter has been closed.

set_mono_shutter_open()

Sets the integrated shutter position to open.

Returns

`bool` – Whether the shutter has been opened.

set_mono_slit_width(*slit_pos*, *slit_width*)

Sets the slit width of a motorized slit (Types 4, 6 and 9).

Parameters

- **slit_pos** (`int`) – The slit to be addressed.
 - Slit Port 1 - Side entrance slit.
 - Slit Port 2 - Front entrance slit.
 - Slit Port 3 - Front exit slit.
 - Slit Port 4 - Side exit slit.
 - Slit Port 5 - Side entrance slit on a double slave unit.
 - Slit Port 6 - Front entrance slit on a double slave unit.
 - Slit Port 7 - Front exit slit on a double slave unit.
 - Slit Port 8 - Side exit slit on a double slave unit.
- **slit_width** (`int`) – The width to which the slit is to be set to. Note valid ranges are 10 to 3000 microns in one-micron increments for normal continuous motor slits (type 4 and 9), 10 to 12000 microns in five-micron increments for large continuous motor slits (type 4) and 25, 50, 100, 250, 500, 1000 microns for indexable slits (Type 6).

set_mono_turret(*turret*)

Sets the current grating turret.

Parameters

turret (`int`) – Set the current turret. This will change the grating to Turret number minus one times gratings per turret plus the one plus the current grating number minus one mod gratings per turret. The user must insert the correct turret into the instrument when using this function.

set_mono_wavelength_abs_cm(*wavelength*)

Sets the center wavelength of the instrument in absolute wavenumbers.

Parameters

wavelength (`float`) – The wavelength in absolute wavenumber that the instrument is to be moved to.

set_mono_wavelength_ang(*wavelength*)

Sets the center wavelength of the instrument in angstroms.

Parameters

wavelength (`float`) – The wavelength in Angstroms that the instrument is to be moved to.

set_mono_wavelength_ev(*wavelength*)

Sets the center wavelength of the instrument in Electron Volts.

Parameters

wavelength (`float`) – The wavelength in Electron Volts that the instrument is to be moved to.

set_mono_wavelength_micron(*wavelength*)

Sets the center wavelength of the instrument in microns.

Parameters

wavelength (`float`) – The wavelength in microns that the instrument is to be moved to.

set_mono_wavelength_nm(*wavelength*)

Sets the center wavelength of the instrument in nm.

Parameters

wavelength (`float`) – The wavelength in nm that the instrument is to be moved to.

set_mono_wavelength_rel_cm(*center_nm*, *wavelength*)

Sets the center wavelength of the instrument in relative Wavenumbers.

Parameters

- **center_nm** (`float`) – The wavelength in nanometers that 0 relative Wavenumber is centered around.
- **wavelength** (`float`) – The wavelength in relative Wavenumber that the instrument is to be moved to.

set_ncl_filter_position(*position*)

Set the current filter position.

Parameters

position (`int`) – The current filter position.

set_ncl_filter_present(*min_filter*, *max_filter*)

Set the instrument filter wheel to active (NCL only).

Parameters

- **min_filter** (`int`) – Min Filter Position (on an NCL with the standard filter wheel 1).

- **max_filter(int)** – Max Filter Position (on an NCL with the standard filter wheel 6).

set_ncl_shutter_closed(shutter_num)

Set a NCL Readout shutter to a closed state.

Parameters

shutter_num(int) – The shutter being addressed (1 or 2 on an NCL).

set_ncl_shutter_open(shutter_num)

Set a NCL Readout shutter to a closed state.

Parameters

shutter_num(int) – The shutter being addressed (1 or 2 on an NCL).

set_ncl_ttl_out_off(ttl_line)

Turn a TTL line off.

Parameters

ttl_line(int) – The TTL line number being addressed.

set_ncl_ttl_out_on(ttl_line)

Turn a TTL line on.

Parameters

ttl_line(int) – The TTL line number being addressed.

set_readout_itime_ms(itime_ms)

Set the integration time used for taking a reading.

Parameters

itime_ms(int) – The integration time in milliseconds to be used when reading out a detector.

static init(path='ARC_Instrument_x64.dll')

Initialize the SDK.

Parameters

path(str) – The path to the SDK.

static is_inited()

Check if the `init()` method was called.

Returns

bool – Whether the `init()` method was called.

static error_to_english(error_code)

Convert an error code into a message.

Parameters

error_code(int) – The error code.

Returns

str – The error message.

msl.equipment.resources.raicol package

Resources for equipment from Raicol Crystals.

Submodules

msl.equipment.resources.raicol.raicol_tec module

Control a TEC (Peltier-based) oven from Raicol Crystals.

class msl.equipment.resources.raicol.raicol_tec.RaicolTEC(*record*)

Bases: *ConnectionSerial*

Control a TEC (Peltier-based) oven from Raicol Crystals.

Do not instantiate this class directly. Use the `connect()` method to connect to the equipment.

Parameters

record (*EquipmentRecord*) – A record from an *Equipment-Register Database*.

get_setpoint()

Get the setpoint temperature.

Returns

float – The setpoint temperature, in Celsius.

off()

Turn the TEC off.

on()

Turn the TEC on.

set_setpoint(*temperature*)

Set the setpoint temperature.

Parameters

temperature (**float**) – The setpoint temperature, in Celsius. Must be in the range [20.1, 60.0].

temperature()

Returns the current temperature of the oven.

The temperature is measured by a PT1000-Platinum resistor temperature sensor that is located near the crystal in the metallic mount.

Returns

float – The temperature of the oven, in Celsius.

msl.equipment.resources.thorlabs package

Resources for equipment 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 Thorlabs Kinesis v1.14.10

msl.equipment.resources.thorlabs.kinesis.benchtop_stepper_motor module

This module provides all the functionality required to control a **Benchtop Stepper Motor** including:

- BSC101
- BSC102
- BSC103
- BSC201
- BSC202
- BSC203

`class msl.equipment.resources.thorlabs.kinesis.benchtop_stepper_motor.BenchtopStepperMotor(`

Bases: *MotionControl*

A wrapper around Thorlabs.MotionControl.Benchtop.StepperMotor.dll.

The *properties* for a BenchtopStepperMotor connection supports the following key-value pairs in the *Connections Database*:

```
'device_name': str, the device name found in ThorlabsDefaultSettings.xml  
↳ [default: None]
```

Do not instantiate this class directly. Use the `connect()` method to connect to the equipment.

Parameters

`record (EquipmentRecord)` – A record from an *Equipment-Register Database*.

can_home(*channel*)

Can the device perform a `home()`?

Parameters

channel (`int`) – The channel number (1 to n).

Returns

`bool` – `True` if the device can perform a home.

can_move_without_homing_first(*channel*)

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

Parameters

channel (`int`) – The channel number (1 to n).

Returns

`bool` – `True` if the device does not need to be `home()`'d before a move can be commanded.

check_connection()

Check connection.

Returns

`bool` – `True` if the USB is listed by the FTDI controller.

clear_message_queue(*channel*)

Clears the device message queue.

Parameters

channel (`int`) – The channel number (1 to n).

Raises

`ThorlabsError` – If not successful.

close()

Disconnect and close the device.

Raises

`ThorlabsError` – If not successful.

disable_channel(*channel*)

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

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

Parameters

channel (`int`) – The channel number (1 to n).

Raises

`ThorlabsError` – If not successful.

enable_channel(*channel*)

Enable channel for computer control.

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

Parameters

channel (`int`) – The channel number (1 to n).

Raises

ThorlabsError – If not successful.

enable_last_msg_timer(*channel*, *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

- **channel** (`int`) – The channel number (1 to n).
- **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(*channel*)

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

See `get_real_value_from_device_unit()` for converting from a `DeviceUnit` to a `RealValue`.

Parameters

channel (`int`) – The channel number (1 to n).

Returns

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

get_bow_index(*channel*)

Gets the stepper motor bow index.

Parameters

channel (`int`) – The channel number (1 to n).

Returns

`int` – The bow index.

get_calibration_file(*channel*)

Get the calibration file for this motor.

Parameters

channel (`int`) – The channel number (1 to n).

Returns

`str` – The filename of the calibration file.

Raises

ThorlabsError – If not successful.

get_device_unit_from_real_value(*channel*, *real_value*, *unit_type*)

Converts a real-world value to a device value.

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

Parameters

- **channel** (`int`) – The channel number (1 to n).

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

Returns

`int` – The device value.

Raises

`ThorlabsError` – If not successful.

get_digital_outputs(*channel*)

Gets the digital output bits.

Parameters

`channel` (`int`) – The channel number (1 to n).

Returns

`int` – Bit mask of states of the 4 digital output pins.

get_encoder_counter(*channel*)

Get the Encoder Counter.

For devices that have an encoder, the current encoder position can be read.

Parameters

`channel` (`int`) – The channel number (1 to n).

Returns

`int` – Encoder count of encoder units.

get_firmware_version(*channel*)

Gets the version number of the device firmware.

Returns

`str` – The firmware version.

get_hardware_info(*channel*)

Gets the hardware information from the device.

Parameters

`channel` (`int`) – The channel number (1 to n).

Returns

`structs.TLI_HardwareInformation` – The hardware information.

Raises

`ThorlabsError` – If not successful.

get_hardware_info_block(*channel*)

Gets the hardware information in a block.

Parameters

`channel` (`int`) – The channel number (1 to n).

Returns

`structs.TLI_HardwareInformation` – The hardware information.

Raises

`ThorlabsError` – If not successful.

get_homing_params_block(channel)

Get the homing parameters.

Parameters

channel (`int`) – The channel number (1 to n).

Returns

`structs.MOT_HomingParameters` – The homing parameters.

Raises

`ThorlabsError` – If not successful.

get_homing_velocity(channel)

Gets the homing velocity.

See `get_real_value_from_device_unit()` for converting from a `DeviceUnit` to a `RealValue`.

Parameters

channel (`int`) – The channel number (1 to n).

Returns

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

get_input_voltage(channel)

Gets the analogue input voltage reading.

Parameters

channel (`int`) – The channel number (1 to n).

Returns

`int` – The input voltage 0-32768 corresponding to 0-5V.

get_jog_mode(channel)

Gets the jog mode.

Parameters

channel (`int`) – The channel number (1 to n).

Returns

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

Raises

`ThorlabsError` – If not successful.

get_jog_params_block(channel)

Get the jog parameters.

Parameters

channel (`int`) – The channel number (1 to n).

Returns

`structs.MOT_JogParameters` – The jog parameters.

Raises

`ThorlabsError` – If not successful.

get_jog_step_size(*channel*)

Gets the distance to move when jogging.

See [*get_real_value_from_device_unit\(\)*](#) for converting from a DeviceUnit to a RealValue.

Parameters

channel (*int*) – The channel number (1 to n).

Returns

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

get_jog_vel_params(*channel*)

Gets the jog velocity parameters.

See [*get_real_value_from_device_unit\(\)*](#) for converting from a DeviceUnit to a RealValue.

Parameters

channel (*int*) – The channel number (1 to n).

Returns

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

Raises

ThorlabsError – If not successful.

get_joystick_params(*channel*)

Gets the joystick parameters.

Parameters

channel (*int*) – The channel number (1 to n).

Returns

structs.MOT_JoystickParameters – The joystick parameters.

Raises

ThorlabsError – If not successful.

get_limit_switch_params(*channel*)

Gets the limit switch parameters.

Parameters

channel (*int*) – The channel number (1 to n).

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

`ThorlabsError` – If not successful.

get_limit_switch_params_block(channel)

Get the limit switch parameters.

Parameters

`channel (int)` – The channel number (1 to n).

Returns

`structs.MOT_LimitSwitchParameters` – The limit switch parameters.

Raises

`ThorlabsError` – If not successful.

get_motor_params(channel)

Gets the motor stage parameters.

Deprecated: calls `get_motor_params_ext()`

get_motor_params_ext(channel)

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`.

Parameters

`channel (int)` – The channel number (1 to n).

Returns

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

Raises

`ThorlabsError` – If not successful.

get_motor_travel_limits(channel)

Gets the motor stage min and max position.

These define the range of travel for the stage.

Parameters

`channel (int)` – The channel number (1 to n).

Returns

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

Raises

`ThorlabsError` – If not successful.

get_motor_travel_mode(*channel*)

Get the motor travel mode.

Parameters

channel (`int`) – The channel number (1 to n).

Returns

`enums.MOT_TravelModes` – The travel mode.

get_motor_velocity_limits(*channel*)

Gets the motor stage maximum velocity and acceleration.

Parameters

channel (`int`) – The channel number (1 to n).

Returns

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

Raises

`ThorlabsError` – If not successful.

get_move_absolute_position(*channel*)

Gets the move absolute position.

See `get_real_value_from_device_unit()` for converting from a DeviceUnit to a RealValue.

Parameters

channel (`int`) – The channel number (1 to n).

Returns

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

get_move_relative_distance(*channel*)

Gets the move relative distance.

See `get_real_value_from_device_unit()` for converting from a DeviceUnit to a RealValue.

Parameters

channel (`int`) – The channel number (1 to n).

Returns

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

get_next_message(*channel*)

Get the next Message Queue item, if it is available. See `messages`.

Parameters

channel (`int`) – The channel number (1 to n).

Returns

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

- `int` – The message data.

Raises

`ThorlabsError` – If not successful.

get_num_channels()

Gets the number of channels in the device.

Returns

`int` – The number of channels.

get_number_positions(*channel*)

Get the 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.

Parameters

`channel (int)` – The channel number (1 to n).

Returns

`int` – The number of positions.

get_pid_loop_encoder_coeff(*channel*)

Gets the Encoder PID loop encoder coefficient.

Parameters

`channel (int)` – The channel number (1 to n).

Returns

`float` – The Encoder PID loop encoder coefficient.

get_pid_loop_encoder_params(*channel*)

Gets the Encoder PID loop parameters.

Parameters

`channel (int)` – The channel number (1 to n).

Returns

`structs.MOT_PIDLoopEncoderParams` – The parameters used to define the Encoder PID Loop.

Raises

`ThorlabsError` – If not successful.

get_position(*channel*)

Get the current position.

The current position is the last recorded position. The current position is updated either by the polling mechanism or by calling `request_position()`.

See `get_real_value_from_device_unit()` for converting from a `DeviceUnit` to a `RealValue`.

Parameters

`channel (int)` – The channel number (1 to n).

Returns

`index (int)` – The current position in `DeviceUnits` (see manual).

get_position_counter(*channel*)

Get the position counter.

The position counter is identical to the position parameter. The position counter is set to zero when homing is complete. The position counter can also be set using [*set_position_counter\(\)*](#) if homing is not to be performed.

See [*get_real_value_from_device_unit\(\)*](#) for converting from a DeviceUnit to a RealValue.

Parameters

channel (`int`) – The channel number (1 to n).

Returns

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

get_power_params(*channel*)

Gets the power parameters for the stepper motor.

Parameters

channel (`int`) – The channel number (1 to n).

Returns

`structs.MOT_PowerParameters` – The power parameters.

Raises

`ThorlabsError` – If not successful.

get_rack_digital_outputs()

Gets the rack digital output bits.

Returns

`int` – Bit mask of states of the 4 digital output pins.

get_rack_status_bits()

Gets the Rack status bits.

Returns

`int` – The status bits including 4 with one per electronic input pin.

get_real_value_from_device_unit(*channel*, *device_value*, *unit_type*)

Converts a device value to a real-world value.

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

Parameters

- **channel** (`int`) – The channel number (1 to n).
- **device_value** (`int`) – The device value.
- **unit_type** (`enums.UnitType`) – The unit of the device value.

Returns

`float` – The real-world value.

Raises

`ThorlabsError` – If not successful.

get_soft_limit_mode(*channel*)

Gets the software limits mode.

Parameters

channel (`int`) – The channel number (1 to n).

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(*channel*)

Gets the Stepper Motor maximum stage position.

See `get_real_value_from_device_unit()` for converting from a DeviceUnit to a RealValue.

Parameters

channel (`int`) – The channel number (1 to n).

Returns

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

get_stage_axis_min_pos(*channel*)

Gets the Stepper Motor minimum stage position.

See `get_real_value_from_device_unit()` for converting from a DeviceUnit to a RealValue.

Parameters

channel (`int`) – The channel number (1 to n).

Returns

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

get_status_bits(*channel*)

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 the polling function, `start_polling()`.

Parameters

channel (`int`) – The channel number (1 to n).

Returns

`int` – The status bits from the device.

get_trigger_switches(*channel*)

Gets the trigger switch parameter.

Parameters

channel (`int`) – The channel number (1 to n).

Returns

`int` – Trigger mask where:

- Bit 0 - Input trigger enabled.
- Bit 1 - Output trigger enabled.
- Bit 2 - Output Passthrough mode enabled where Output Trigger mirrors Input Trigger.
- Bit 3 - Output trigger high when moving.
- Bit 4 - Performs relative move when input trigger goes high.
- Bit 5 - Performs absolute move when input trigger goes high.
- Bit 6 - Perfgoms home when input trigger goes high.
- Bit 7 - Output triggers when motor moved by software command.

get_vel_params(*channel*)

Gets the move velocity parameters.

See `get_real_value_from_device_unit()` for converting from a DeviceUnit to a RealValue.

Parameters

`channel (int)` – The channel number (1 to n).

Returns

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

Raises

`ThorlabsError` – If not successful.

get_vel_params_block(*channel*)

Get the move velocity parameters.

Parameters

`channel (int)` – The channel number (1 to n).

Returns

`structs.MOT_VelocityParameters` – The velocity parameters.

Raises

`ThorlabsError` – If not successful.

has_last_msg_timer_overrun(*channel*)

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.

Parameters

`channel (int)` – The channel number (1 to n).

Returns

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

home(*channel*)

Home the device.

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

Parameters

`channel (int)` – The channel number (1 to n).

Raises

`ThorlabsError` – If not successful.

identify(*channel*)

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

Parameters

`channel (int)` – The channel number (1 to n).

Raises

`ThorlabsError` – If not successful.

is_calibration_active(*channel*)

Is a calibration file active for this motor?

Parameters

`channel (int)` – The channel number (1 to n).

Returns

`bool` – Whether a calibration file is active.

is_channel_valid(*channel*)

Verifies that the specified channel is valid.

Parameters

`channel (int)` – The requested channel number (1 to n).

Returns

`bool` – Whether the channel is valid.

load_settings(*channel*)

Update device with stored settings.

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

Parameters

`channel (int)` – The channel number (1 to n).

Raises

`ThorlabsError` – If not successful.

load_named_settings(*channel*, *settings_name*)

Update device with named settings.

Parameters

- `channel (int)` – The channel number (1 to n).

- **settings_name** (`str`) – The name of the device to load the settings for. Examples for the value of `setting_name` can be found in `ThorlabsDefaultSettings.xml`, which gets created when the Kinesis software is installed.

Raises

`ThorlabsError` – If not successful.

max_channel_count()

Gets the number of channels available to this device.

This function returns the number of available bays, not the number of bays filled.

Returns

`int` – The number of channels available on this device.

message_queue_size(channel)

Gets the size of the message queue.

Parameters

`channel` (`int`) – The channel number (1 to n).

Returns

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

move_absolute(channel)

Moves the device to the position defined in the `set_move_absolute_position()` command.

Parameters

`channel` (`int`) – The channel number (1 to n).

Raises

`ThorlabsError` – If not successful.

move_at_velocity(channel, direction)

Start moving at the current velocity in the specified direction.

Parameters

- `channel` (`int`) – The channel number (1 to n).
- `direction` (`enums.MOT_TravelDirection`) – The required direction of travel as a `enums.MOT_TravelDirection` enum value or member name.

Raises

`ThorlabsError` – If not successful.

move_jog(channel, jog_direction)

Perform a jog.

Parameters

- `channel` (`int`) – The channel number (1 to n).
- `jog_direction` (`enums.MOT_TravelDirection`) – The jog direction as a `enums.MOT_TravelDirection` enum value or member name.

Raises

ThorlabsError – If not successful.

move_relative(*channel*, *displacement*)

Move the motor by a relative amount.

See [get_device_unit_from_real_value\(\)](#) for converting from a RealValue to a DeviceUnit.

Parameters

- **channel** ([int](#)) – The channel number (1 to n).
- **displacement** ([int](#)) – Signed displacement in DeviceUnits (see manual).

Raises

ThorlabsError – If not successful.

move_relative_distance(*channel*)

Moves the device by a relative distance defined by [set_move_relative_distance\(\)](#).

Parameters

channel ([int](#)) – The channel number (1 to n).

Raises

ThorlabsError – If not successful.

move_to_position(*channel*, *index*)

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 RealValue to a DeviceUnit.

Parameters

- **channel** ([int](#)) – The channel number (1 to n).
- **index** ([int](#)) – The position in DeviceUnits (see manual).

Raises

ThorlabsError – If not successful.

needs_homing(*channel*)

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

ThorlabsError – If not successful.

persist_settings(*channel*)

Persist device settings to device.

Parameters

channel (`int`) – The channel number (1 to n).

polling_duration(*channel*)

Gets the polling loop duration.

Parameters

channel (`int`) – The channel number (1 to n).

Returns

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

register_message_callback(*channel*, *callback*)

Registers a callback on the message queue.

Parameters

- **channel** (`int`) – The channel number (1 to n).
- **callback** (`MotionControlCallback`) – A function to be called whenever messages are received.

Raises

`ThorlabsError` – If not successful.

request_backlash(*channel*)

Requests the backlash.

Parameters

channel (`int`) – The channel number (1 to n).

Raises

`ThorlabsError` – If not successful.

request_bow_index(*channel*)

Requests the stepper motor bow index.

Parameters

channel (`int`) – The channel number (1 to n).

Raises

`ThorlabsError` – If not successful.

request_digital_outputs(*channel*)

Requests the digital output bits.

Parameters

channel (`int`) – The channel number (1 to n).

Raises

`ThorlabsError` – If not successful.

request_encoder_counter(*channel*)

Requests the encoder counter.

For devices that have an encoder, the current encoder position can be read.

Parameters

channel (`int`) – The channel number (1 to n).

Raises

`ThorlabsError` – If not successful.

request_homing_params(*channel*)

Requests the homing parameters.

Parameters

channel (`int`) – The channel number (1 to n).

Raises

`ThorlabsError` – If not successful.

request_input_voltage(*channel*)

Requests the analogue input voltage reading.

Parameters

channel (`int`) – The channel number (1 to n).

Raises

`ThorlabsError` – If not successful.

request_jog_params(*channel*)

Requests the jog parameters.

Parameters

channel (`int`) – The channel number (1 to n).

Raises

`ThorlabsError` – If not successful.

request_joystick_params(*channel*)

Requests the joystick parameters.

Parameters

channel (`int`) – The channel number (1 to n).

Raises

`ThorlabsError` – If not successful.

request_limit_switch_params(*channel*)

Requests the limit switch parameters.

Parameters

channel (`int`) – The channel number (1 to n).

Raises

`ThorlabsError` – If not successful.

request_move_absolute_position(*channel*)

Requests the position of next absolute move.

Parameters

channel (`int`) – The channel number (1 to n).

Raises

`ThorlabsError` – If not successful.

request_move_relative_distance(*channel*)

Requests the relative move distance.

Parameters

channel (`int`) – The channel number (1 to n).

Raises

`ThorlabsError` – If not successful.

request_pid_loop_encoder_params(*channel*)

Requests the Encoder PID loop parameters.

Parameters

channel (`int`) – The channel number (1 to n).

Raises

`ThorlabsError` – If not successful.

request_position(*channel*)

Requests the current position.

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

Parameters

channel (`int`) – The channel number (1 to n).

Raises

`ThorlabsError` – If not successful.

request_power_params(*channel*)

Requests the power parameters.

Parameters

channel (`int`) – The channel number (1 to n).

Raises

`ThorlabsError` – If not successful.

request_rack_digital_outputs()

Requests the rack digital output bits.

Raises

`ThorlabsError` – If not successful.

request_rack_status_bits()

Requests the Rack status bits be downloaded.

Raises

`ThorlabsError` – If not successful.

request_settings(*channel*)

Requests that all settings are downloaded from the device.

This function requests that the device upload all its settings to the DLL.

Parameters

channel (`int`) – The channel number (1 to n).

Raises

ThorlabsError – If not successful.

request_status_bits(*channel*)

Request the status bits which identify the current motor state.

This needs to be called to get the device to send its current status bits. Note, this is called automatically if Polling is enabled for the device using *start_polling()*.

Parameters

channel (*int*) – The channel number (1 to n).

Raises

ThorlabsError – If not successful.

request_trigger_switches(*channel*)

Requests the trigger switch parameter.

Warning: This function is currently not in the DLL, as of v1.14.8, but it is in the header file.

Parameters

channel (*int*) – The channel number (1 to n).

Raises

ThorlabsError – If not successful.

request_vel_params(*channel*)

Requests the velocity parameters.

Parameters

channel (*int*) – The channel number (1 to n).

Raises

ThorlabsError – If not successful.

reset_rotation_modes(*channel*)

Reset the rotation modes for a rotational device.

Parameters

channel (*int*) – The channel number (1 to n).

Raises

ThorlabsError – If not successful.

resume_move_messages(*channel*)

Resume suspended move messages.

Parameters

channel (*int*) – The channel number (1 to n).

Raises

ThorlabsError – If not successful.

set_backlash(*channel*, *distance*)

Sets the backlash distance (used to control hysteresis).

See [`get_device_unit_from_real_value\(\)`](#) for converting from a `RealValue` to a `DeviceUnit`.

Parameters

- **channel** (`int`) – The channel number (1 to n).
- **distance** (`int`) – The backlash distance in `DeviceUnits` (see manual).

Raises

`ThorlabsError` – If not successful.

set_bow_index(*channel*, *bow_index*)

Sets the stepper motor bow index.

Parameters

- **channel** (`int`) – The channel number (1 to n).
- **bow_index** (`int`) – The bow index.

Raises

`ThorlabsError` – If not successful.

set_calibration_file(*channel*, *path*, *enabled*)

Set the calibration file for this motor.

Parameters

- **channel** (`int`) – The channel number (1 to n).
- **path** (`str`) – The path to a calibration file to load.
- **enabled** (`bool`) – `True` to enable, `False` to disable.

Raises

`OSError` – If the *path* does not exist.

set_digital_outputs(*channel*, *outputs_bits*)

Sets the digital output bits.

Parameters

- **channel** (`int`) – The channel number (1 to n).
- **outputs_bits** (`int`) – Bit mask to set states of the 4 digital output pins.

Raises

`ThorlabsError` – If not successful.

set_direction(*channel*, *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

- **channel** (`int`) – The channel number (1 to n).

- **reverse** (`bool`) – If `True` then directions will be swapped on these moves.

Raises

`ThorlabsError` – If not successful.

set_encoder_counter(*channel*, *count*)

Set the Encoder Counter values.

Setting the encoder counter to zero, effectively defines a home position on the encoder strip. Setting this value does not move the device.

Parameters

- **channel** (`int`) – The channel number (1 to n).
- **count** (`int`) – The encoder count in encoder units.

Raises

`ThorlabsError` – If not successful.

set_homing_params_block(*channel*, *direction*, *limit*, *velocity*, *offset*)

Set the homing parameters.

Parameters

- **channel** (`int`) – The channel number (1 to n).
- **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

`ThorlabsError` – If not successful.

set_homing_velocity(*channel*, *velocity*)

Sets the homing velocity.

See `get_device_unit_from_real_value()` for converting from a `RealValue` to a `DeviceUnit`.

Parameters

- **channel** (`int`) – The channel number (1 to n).
- **velocity** (`int`) – The homing velocity in `DeviceUnits` (see manual).

Raises

`ThorlabsError` – If not successful.

set_jog_mode(*channel*, *mode*, *stop_mode*)

Sets the jog mode.

Parameters

- **channel** (`int`) – The channel number (1 to n).
- **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

`ThorlabsError` – If not successful.

set_jog_params_block(*channel*, *jog_params*)

Set the jog parameters.

Parameters

- **channel** (`int`) – The channel number (1 to n).
- **jog_params** (`structs.MOT_JogParameters`) – The jog parameters.

Raises

`ThorlabsError` – If not successful.

set_jog_step_size(*channel*, *step_size*)

Sets the distance to move on jogging.

See `get_device_unit_from_real_value()` for converting from a `RealValue` to a `DeviceUnit`.

Parameters

- **channel** (`int`) – The channel number (1 to n).
- **step_size** (`int`) – The step size in `DeviceUnits` (see manual).

Raises

`ThorlabsError` – If not successful.

set_jog_vel_params(*channel*, *max_velocity*, *acceleration*)

Sets jog velocity parameters.

See `get_device_unit_from_real_value()` for converting from a `RealValue` to a `DeviceUnit`.

Parameters

- **channel** (`int`) – The channel number (1 to n).
- **max_velocity** (`int`) – The maximum velocity in `DeviceUnits` (see manual).
- **acceleration** (`int`) – The acceleration in `DeviceUnits` (see manual).

Raises

`ThorlabsError` – If not successful.

set_joystick_params(*channel*, *joystick_params*)

Sets the joystick parameters.

Parameters

- **channel** (`int`) – The channel number (1 to n).
- **joystick_params** (`structs.MOT_JoystickParameters`) – The joystick parameters.

Raises

`ThorlabsError` – If not successful.

set_limit_switch_params(*channel*, *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 `RealValue` to a `DeviceUnit`.

Parameters

- **channel** (`int`) – The channel number (1 to n).
- **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

`ThorlabsError` – If not successful.

set_limit_switch_params_block(*channel*, *params*)

Set the limit switch parameters.

Parameters

- **channel** (`int`) – The channel number (1 to n).
- **params** (`structs.MOT_LimitSwitchParameters`) – The new limit switch parameters.

Raises

`ThorlabsError` – If not successful.

set_limits_software_approach_policy(*channel*, *policy*)

Sets the software limits policy.

Parameters

- **channel** (`int`) – The channel number (1 to n).

- **policy** (`enums.MOT_LimitsSoftwareApproachPolicy`) – The soft limit mode as a `enums.MOT_LimitsSoftwareApproachPolicy` enum value or member name.

set_motor_params(*channel*, *steps_per_rev*, *gear_box_ratio*, *pitch*)

Sets the motor stage parameters.

Deprecated: calls `set_motor_params_ext()`

set_motor_params_ext(*channel*, *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 DeviceUnit to a RealValue.

Parameters

- **channel** (`int`) – The channel number (1 to n).
- **steps_per_rev** (`float`) – The steps per revolution.
- **gear_box_ratio** (`float`) – The gear box ratio.
- **pitch** (`float`) – The pitch.

Raises

ThorlabsError – If not successful.

set_motor_travel_limits(*channel*, *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 DeviceUnit to a RealValue.

Parameters

- **channel** (`int`) – The channel number (1 to n).
- **min_position** (`float`) – The minimum position in RealWorldUnits [millimeters or degrees].
- **max_position** (`float`) – The maximum position in RealWorldUnits [millimeters or degrees].

Raises

ThorlabsError – If not successful.

set_motor_travel_mode(*channel*, *travel_mode*)

Set the motor travel mode.

Parameters

- **channel** (`int`) – The channel number (1 to n).
- **travel_mode** (`enums.MOT_TravelModes`) – The travel mode as a `enums.MOT_TravelModes` enum value or member name.

Raises

ThorlabsError – If not successful.

set_motor_velocity_limits(*channel*, *max_velocity*, *max_acceleration*)

Sets the motor stage maximum velocity and acceleration.

See [*get_real_value_from_device_unit\(\)*](#) for converting from a DeviceUnit to a RealValue.

Parameters

- **channel** (`int`) – The channel number (1 to n).
- **max_velocity** (`float`) – The maximum velocity in RealWorldUnits [millimeters or degrees].
- **max_acceleration** (`float`) – The maximum acceleration in RealWorldUnits [millimeters or degrees].

Raises

ThorlabsError – If not successful.

set_move_absolute_position(*channel*, *position*)

Sets the move absolute position.

See [*get_device_unit_from_real_value\(\)*](#) for converting from a RealValue to a DeviceUnit.

Parameters

- **channel** (`int`) – The channel number (1 to n).
- **position** (`int`) – The absolute position in DeviceUnits (see manual).

Raises

ThorlabsError – If not successful.

set_move_relative_distance(*channel*, *distance*)

Sets the move relative distance.

See [*get_device_unit_from_real_value\(\)*](#) for converting from a RealValue to a DeviceUnit.

Parameters

- **channel** (`int`) – The channel number (1 to n).
- **distance** (`int`) – The relative position in DeviceUnits (see manual).

Raises

ThorlabsError – If not successful.

set_pid_loop_encoder_coeff(*channel*, *coeff*)

Sets the Encoder PID loop encoder coefficient.

Parameters

- **channel** (`int`) – The channel number (1 to n).
- **coeff** (`float`) – The Encoder PID loop encoder coefficient. Set to 0.0 to disable the encoder or if no encoder is present otherwise the positive encoder coefficient.

Raises

ThorlabsError – If not successful.

set_pid_loop_encoder_params(*channel*, *mode*, *prop_gain*, *int_gain*, *diff_gain*, *limit*, *tol*)

Sets the Encoder PID loop parameters.

Parameters

- **channel** (*int*) – The channel number (1 to n).
- **mode** (*enums.MOT_PID_LoopMode*) – The Encoder PID loop mode as a *enums.MOT_PID_LoopMode* enum value or member name.
- **prop_gain** (*int*) – The Encoder PID Loop proportional gain. Range 0 to 2^{24} .
- **int_gain** (*int*) – The Encoder PID Loop integral gain. Range 0 to 2^{24} .
- **diff_gain** (*int*) – The Encoder PID Loop differential gain. Range 0 to 2^{24} .
- **limit** (*int*) – The Encoder PID Loop output limit. Range 0 to 2^{15} .
- **tol** (*int*) – The Encoder PID Loop tolerance. Range 0 to 2^{15} .

Raises

ThorlabsError – If not successful.

set_position_counter(*channel*, *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 *RealValue* to a *DeviceUnit*.

Parameters

- **channel** (*int*) – The channel number (1 to n).
- **count** (*int*) – The position counter in *DeviceUnits* (see manual).

Raises

ThorlabsError – If not successful.

set_power_params(*channel*, *rest*, *move*)

Sets the power parameters for the stepper motor.

Parameters

- **channel** (*int*) – The channel number (1 to n).
- **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

ThorlabsError – If not successful.

set_rack_digital_outputs(*outputs_bits*)

Sets the rack digital output bits.

Parameters

outputs_bits (`int`) – Bit mask to set states of the 4 digital output pins.

Raises

`ThorlabsError` – If not successful.

set_rotation_modes(*channel, mode, direction*)

Set the rotation modes for a rotational device.

Parameters

- **channel** (`int`) – The channel number (1 to n).
- **mode** (`enums.MOT_MovementModes`) – The travel mode as a `enums.MOT_MovementModes` enum value or member name.
- **direction** (`enums.MOT_MovementDirections`) – The travel mode as a `enums.MOT_MovementDirections` enum value or member name.

Raises

`ThorlabsError` – If not successful.

set_stage_axis_limits(*channel, min_position, max_position*)

Sets the stage axis position limits.

See `get_device_unit_from_real_value()` for converting from a `RealValue` to a `DeviceUnit`.

Parameters

- **channel** (`int`) – The channel number (1 to n).
- **min_position** (`int`) – The minimum position in `DeviceUnits` (see manual).
- **max_position** (`int`) – The maximum position in `DeviceUnits` (see manual).

Raises

`ThorlabsError` – If not successful.

set_trigger_switches(*channel, indicator_bits*)

Sets the trigger switch bits.

Parameters

- **channel** (`int`) – The channel number (1 to n).
- **indicator_bits** (`int`) – Sets the 8 bits indicating action on trigger input and events to trigger electronic output.
 - Bit 0 - Input trigger enabled.
 - Bit 1 - Output trigger enabled.
 - Bit 2 - Output pass-through mode enabled where Output Trigger mirrors Input Trigger.
 - Bit 3 - Output trigger high when moving.

- Bit 4 - Performs relative move when input trigger goes high.
- Bit 5 - Performs absolute move when input trigger goes high.
- Bit 6 - Performs home when input trigger goes high.
- Bit 7 - Output triggers when motor moved by software command.

Raises

ThorlabsError – If not successful.

set_vel_params(*channel*, *max_velocity*, *acceleration*)

Sets the move velocity parameters.

See [get_device_unit_from_real_value\(\)](#) for converting from a RealValue to a DeviceUnit.

Parameters

- **channel** (*int*) – The channel number (1 to n).
- **max_velocity** (*int*) – The maximum velocity in DeviceUnits (see manual).
- **acceleration** (*int*) – The acceleration in DeviceUnits (see manual).

Raises

ThorlabsError – If not successful.

set_vel_params_block(*channel*, *min_velocity*, *max_velocity*, *acceleration*)

Set the move velocity parameters.

See [get_device_unit_from_real_value\(\)](#) for converting from a RealValue to a DeviceUnit.

Parameters

- **channel** (*int*) – The channel number (1 to n).
- **min_velocity** (*int*) – The minimum velocity in DeviceUnits (see manual).
- **max_velocity** (*int*) – The maximum velocity in DeviceUnits (see manual)..
- **acceleration** (*int*) – The acceleration in DeviceUnits (see manual)..

Raises

ThorlabsError – If not successful.

start_polling(*channel*, *milliseconds*)

Starts the internal polling loop.

This function continuously requests position and status messages.

Parameters

- **channel** (*int*) – The channel number (1 to n).
- **milliseconds** (*int*) – The polling rate, in milliseconds.

Raises

ThorlabsError – If not successful.

stop_immediate(*channel*)

Stop the current move immediately (with the risk of losing track of the position).

Parameters

channel (`int`) – The channel number (1 to n).

Raises

ThorlabsError – If not successful.

stop_polling(*channel*)

Stops the internal polling loop.

Parameters

channel (`int`) – The channel number (1 to n).

stop_profiled(*channel*)

Stop the current move using the current velocity profile.

Parameters

channel (`int`) – The channel number (1 to n).

Raises

ThorlabsError – If not successful.

suspend_move_messages(*channel*)

Suspend automatic messages at ends of moves.

Useful to speed up part of real-time system with lots of short moves.

Parameters

channel (`int`) – The channel number (1 to n).

Raises

ThorlabsError – If not successful.

time_since_last_msg_received(*channel*)

Gets the time, in milliseconds, since the last message was received.

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

Parameters

channel (`int`) – The channel number (1 to n).

Returns

- `int` – The time, in milliseconds, since the last message was received.
- `bool` – `True` if monitoring is enabled otherwise `False`.

uses_pid_loop_encoding(*channel*)

Determines if we can uses PID loop encoding.

This is true if the stage supports PID Loop Encoding. Requires `get_pid_loop_encoder_coeff()` to have a positive non-zero coefficient, see also `set_pid_loop_encoder_coeff()`.

Parameters

channel (`int`) – The channel number (1 to n).

wait_for_message(*channel*)

Wait for next Message Queue item if it is available. See [messages](#).

Parameters

channel (`int`) – The channel number (1 to n).

Returns

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

Raises

ThorlabsError – If not successful.

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

A callback to register for a [MotionControl](#) message queue.

```
from msl.equipment import Config
from msl.equipment.resources.thorlabs import MotionControlCallback

@MotionControlCallback
def msg_callback():
    print('MotionControlCallback: ', flipper.convert_message(*flipper.get_
→next_message()))

# The "example2.xml" configuration file contains the following element:
# <equipment alias="filter_flipper" manufacturer="Thorlabs" model="MFF101/M"/>

db = Config('config.xml').database()

flipper = db.equipment['filter_flipper'].connect()
flipper.register_message_callback(msg_callback)

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

msl.equipment.resources.thorlabs.kinesis.callbacks.MotionControlCallback

A callback to register for a [MotionControl](#) message queue.

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

Enums defined in Thorlabs Kinesis v1.14.10

```
class msl.equipment.resources.thorlabs.kinesis.enums.FT_Status(value, names=None,
                                                               *values,
                                                               module=None,
                                                               qualname=None,
                                                               type=None, start=1,
                                                               boundary=None)
```

Bases: [IntEnum](#)

```
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(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
Bases: IntEnum
MOT_NotMotor = 0
MOT_DCMotor = 1
MOT_StepperMotor = 2
MOT_BrushlessMotor = 3
MOT_CustomMotor = 100

class msl.equipment.resources.thorlabs.kinesis.enums.MOT_TravelModes(value,
    names=None,
    *values,
    mod-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
Bases: IntEnum
MOT_TravelModeUndefined = 0
```

```
MOT_Linear = 1
```

```
MOT_Rotational = 2
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.MOT_TravelDirection(value,
                           names=None,
                           *values,
                           mode=None,
                           quality=None,
                           name=None,
                           type=None,
                           start=1,
                           boundary=None)
```

Bases: [IntEnum](#)

```
MOT_TravelDirectionDisabled = 0
```

```
MOT_Forwards = 1
```

```
MOT_Reverse = 2
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.MOT_DirectionSense(value,
                           names=None,
                           *values,
                           mode=None,
                           quality=None,
                           name=None,
                           type=None,
                           start=1,
                           boundary=None)
```

Bases: [IntEnum](#)

```
MOT_Normal = 0
```

```
MOT_Backwards = 1
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.MOT_HomeLimitSwitchDirection(value,
                                     names=None,
                                     *values,
                                     mode=None,
                                     quality=None,
                                     name=None,
                                     type=None,
                                     start=1,
                                     boundary=None)
```

Bases: [IntEnum](#)

MOT_LimitSwitchDirectionUndefined = 0

MOT_ReverseLimitSwitch = 1

MOT_ForwardLimitSwitch = 4

```
class msl.equipment.resources.thorlabs.kinesis.enums.MOT_JogModes(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1, bound-
    ary=None)
```

Bases: [IntEnum](#)

MOT_JogModeUndefined = 0

MOT_Continuous = 1

MOT_SingleStep = 2

```
class msl.equipment.resources.thorlabs.kinesis.enums.MOT_StopModes(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

MOT_StopModeUndefined = 0

MOT_Immediate = 1

MOT_Profiled = 2

```
class msl.equipment.resources.thorlabs.kinesis.enums.MOT_ButtonModes(value,
    names=None,
    *values,
    mod-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

```
MOT_ButtonModeUndefined = 0
MOT_JogMode = 1
MOT_Preset = 2

class msl.equipment.resources.thorlabs.kinesis.enums.MOT_VelocityProfileModes(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)

Bases: IntEnum
MOT_Trapezoidal = 0
MOT_SCurve = 2

class msl.equipment.resources.thorlabs.kinesis.enums.MOT_LimitSwitchModes(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)

Bases: IntEnum
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(value,
    names=None,
    *values,
    module=None,
    qualname=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

```
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_LimitsSoftwareApproachPolicy(value,
    names=None,
    *values,
    module=None,
    qualname=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

```
DisallowIllegalMoves = 0
```

```
AllowPartialMoves = 1
```

```
AllowAllMoves = 2
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.MOT_CurrentLoopPhases(value,
                           names=None,
                           *values,
                           module=None,
                           qual-
                           name=None,
                           type=None,
                           start=1,
                           bound-
                           ary=None)
```

Bases: [IntEnum](#)

MOT_PhaseA = 0

MOT_PhaseB = 1

MOT_PhaseAB = 2

```
class msl.equipment.resources.thorlabs.kinesis.enums.MOT_MovementModes(value,
                           names=None,
                           *values,
                           module=None,
                           qual-
                           name=None,
                           type=None,
                           start=1,
                           bound-
                           ary=None)
```

Bases: [IntEnum](#)

LinearRange = 0

RotationalUnlimited = 1

RotationalWrapping = 2

```
class msl.equipment.resources.thorlabs.kinesis.enums.MOT_MovementDirections(value,
                           names=None,
                           *values,
                           module=None,
                           qual-
                           name=None,
                           type=None,
                           start=1,
                           bound-
                           ary=None)
```

Bases: [IntEnum](#)

Quickest = 0

Forwards = 1

Reverse = 2

```
class msl.equipment.resources.thorlabs.kinesis.enums.MOT_PID_LoopMode(value,
    names=None,
    *values,
    module=None,
    qualname=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

MOT_PIDLoopModeDisabled = 0

MOT_PIDOpenLoopMode = 1

MOT_PIDClosedLoopMode = 2

```
class msl.equipment.resources.thorlabs.kinesis.enums.NT_SignalState(value,
    names=None,
    *values, module=None,
    qualname=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

NT_BadSignal = 0

NT_GoodSignal = 1

```
class msl.equipment.resources.thorlabs.kinesis.enums.NT_Mode(value, names=None,
    *values,
    module=None,
    qualname=None,
    type=None, start=1,
    boundary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)

Bases: IntEnum
NT_ControlModeUndefined = 0
NT_OpenLoop = 1
NT_ClosedLoop = 2
NT_OpenLoopSmoothed = 3
NT_ClosedLoopSmoothed = 4

class msl.equipment.resources.thorlabs.kinesis.enums.NT_FeedbackSource(value,
    names=None,
    *values,
    mod-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)

Bases: IntEnum
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(value,
                                                               names=None,
                                                               *values,
                                                               module=None,
                                                               qualname=None,
                                                               type=None,
                                                               start=1,
                                                               boundary=None)
```

Bases: [IntEnum](#)

```
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(value,
                                                               names=None,
                                                               *values,
                                                               module=None,
                                                               qual-
                                                               name=None,
                                                               type=None,
                                                               start=1, bound-
                                                               ary=None)
```

Bases: [IntEnum](#)

```
NT_OddAndEven = 1
NT_Odd = 2
NT_Even = 3
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.NT_UnderOrOver(value,
    names=None,
    *values, module=None,
    qual-name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

```
NT_InRange = 1
NT_UnderRange = 2
NT_OverRange = 3
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.NT_CircleDiameterMode(value,
    names=None,
    *values, module=None,
    qual-name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

```
NT_ParameterCircleMode = 1
NT_AbsPowerCircleMode = 2
NT_LUTCircleMode = 3
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.NT_CircleAdjustment(value,
    names=None,
    *values, module=None,
    qual-name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

```
NT_LinearCircleAdjustment = 1
```

NT_LogCircleAdjustment = 2

NT_SquareCircleAdjustment = 3

NT_CubeCircleAdjustment = 4

```
class msl.equipment.resources.thorlabs.kinesis.enums.NT_TIARangeMode(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

NT_TIARangeModeUndefined = 0

NT_AutoRangeAtSelected = 1

NT_ManualRangeAtSelected = 2

NT_ManualRangeAtParameter = 3

NT_AutoRangeAtParameter = 4

```
class msl.equipment.resources.thorlabs.kinesis.enums.NT_LowPassFrequency(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

```
NT_VoltageRangeUndefined = 0
NT_VoltageRange_5v = 1
NT_VoltageRange_10v = 2
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.NT_OutputVoltageRoute(value,
    names=None,
    *val-
    ues,
    mod-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

```
NT_SMAOnly = 1
NT_HubOrSMA = 2
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.NT_PowerInputUnits(value,
    names=None,
    *values,
    mod-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

```
NT_Amps = 0
NT_Watts = 1
```

NT_Db = 2

```
class msl.equipment.resources.thorlabs.kinesis.enums.NT_SMA_Units(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1, bound-
    ary=None)
```

Bases: [IntEnum](#)

NT_Voltage = 0

NT_FullRange = 1

NT_UserDefined = 2

```
class msl.equipment.resources.thorlabs.kinesis.enums.BNT_CurrentLimit(value,
    names=None,
    *values,
    mod-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

NT_CurrentLimit_100mA = 0

NT_CurrentLimit_250mA = 1

NT_CurrentLimit_500mA = 2

```
class msl.equipment.resources.thorlabs.kinesis.enums.BNT_OutputLowPassFilter(value,
    names=None,
    *val-
    ues,
    mod-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

```
NT_FeedbackSignalDC = 0
```

```
NT_FeedbackSignalAC = 65535
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.BNT_BNCTriggerModes(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

```
NT_BNCModeTrigger = 0
```

```
NT_BNCModeLVOut = 65535
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.PZ_ControlModeTypes(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

PZ_Undefined = 0

PZ_OpenLoop = 1

PZ_CloseLoop = 2

PZ_OpenLoopSmooth = 3

PZ_CloseLoopSmooth = 4

```
class msl.equipment.resources.thorlabs.kinesis.enums.PZ_InputSourceFlags(value,
                           names=None,
                           *values,
                           module=None,
                           qualname=None,
                           type=None,
                           start=1,
                           boundary=None)
```

Bases: [IntEnum](#)

PZ_SoftwareOnly = 0

PZ_ExternalSignal = 1

PZ_Potentiometer = 2

PZ_All = 3

```
class msl.equipment.resources.thorlabs.kinesis.enums.PZ_OutputLUTModes(value,
                           names=None,
                           *values,
                           module=None,
                           qualname=None,
                           type=None,
                           start=1,
                           boundary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *values,
    module=None,
    qualname=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

```
DerivFilterOn = 1
```

```
DerivFilterOff = 2
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.PPC_NotchFilterState(value,
    names=None,
    *values,
    module=None,
    qualname=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

```
NotchFilterOn = 1
```

```
NotchFilterOff = 2
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.PPC_NotchFilterChannel(value,
    names=None,
    *values,
    mode=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

NotchFilter1 = 1

NotchFilter2 = 2

NotchFilterBoth = 3

```
class msl.equipment.resources.thorlabs.kinesis.enums.PPC_IoControlMode(value,
    names=None,
    *values,
    mode=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

SWOnly = 0

ExtBNC = 1

Joystick = 2

JoystickBnc = 3

```
class msl.equipment.resources.thorlabs.kinesis.enums.PPC_IoOutputMode(value,
    names=None,
    *values,
    mode=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

HV = 1

PosRaw = 2

PosCorrected = 3

```
class msl.equipment.resources.thorlabs.kinesis.enums.PPC_I0OutputBandwidth(value,
                           names=None,
                           *values,
                           module=None,
                           qualname=None,
                           type=None,
                           start=1,
                           boundary=None)
```

Bases: [IntEnum](#)

OP_Unfiltered = 1

OP_200Hz = 2

```
class msl.equipment.resources.thorlabs.kinesis.enums.PPC_I0FeedbackSourceDefinition(value,
                                    names=None,
                                    *values,
                                    module=None,
                                    qualname=None,
                                    type=None,
                                    start=1,
                                    boundary=None)
```

Bases: [IntEnum](#)

StrainGauge = 1

Capacitive = 2

Optical = 3

```
class msl.equipment.resources.thorlabs.kinesis.enums.PPC_DisplayIntensity(value,
                           names=None,
                           *values,
                           module=None,
                           qualname=None,
                           type=None,
                           start=1,
                           boundary=None)
```

Bases: [IntEnum](#)

Bright = 1

Dim = 2

Off = 3

```
class msl.equipment.resources.thorlabs.kinesis.enums.FF_Positions(value,
                     names=None,
                     *values,
                     module=None,
                     qualname=None,
                     type=None,
                     start=1, boundary=None)
```

Bases: [IntEnum](#)

FF_PositionError = 0

Position1 = 1

Position2 = 2

```
class msl.equipment.resources.thorlabs.kinesis.enums.FF_IOModes(value,
                     names=None,
                     *values,
                     module=None,
                     qualname=None,
                     type=None,
                     start=1,
                     boundary=None)
```

Bases: [IntEnum](#)

FF_ToggleOnPositiveEdge = 1

FF_SetPositionOnPositiveEdge = 2

FF_OutputHighAtsetPosition = 4

FF_OutputHighWhenMoving = 8

```
class msl.equipment.resources.thorlabs.kinesis.enums.FF_SignalModes(value,
    names=None,
    *values, module=None,
    qual-name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

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_WheelDirectionSense(value,
    names=None,
    *values, module=None,
    qual-name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

KMOT_WM_Positive = 1

KMOT_WM_Negative = 2

```
class msl.equipment.resources.thorlabs.kinesis.enums.KMOT_WheelMode(value,
    names=None,
    *values, module=None,
    qual-name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

KMOT_WM_Velocity = 1

KMOT_WM_Jog = 2

KMOT_WM_MoveAbsolute = 3

```
class msl.equipment.resources.thorlabs.kinesis.enums.KMOT_TriggerPortMode(value,
                           names=None,
                           *values,
                           mode=None,
                           quality=None,
                           name=None,
                           type=None,
                           start=1,
                           boundary=None)
```

Bases: [IntEnum](#)

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(value,
                           names=None,
                           *values,
                           mode=None,
                           quality=None,
                           name=None,
                           type=None,
                           start=1,
                           boundary=None)
```

Bases: [IntEnum](#)

KMOT_TrigPolarityHigh = 1

```
KMOT_TrigPolarityLow = 2
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.KIM_Channels(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1, bound-
    ary=None)
```

Bases: [IntEnum](#)

```
Channel1 = 1
```

```
Channel2 = 2
```

```
Channel3 = 3
```

```
Channel4 = 4
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.KIM_JogMode(value,
    names=None,
    *values,
    module=None,
    qualname=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

```
JogContinuous = 1
```

```
JogStep = 2
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.KIM_TravelDirection(value,
    names=None,
    *val-
    ues,
    mod-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

```
Forward = 1
```

```
Reverse = 2
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.KIM_FBSignalMode(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

```
FB_LimitSwitch = 1

FB_Encoder = 2
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.KIM_LimitSwitchModes(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

```
Ignore = 1

SwitchMakes = 2

SwitchBreaks = 3

SwitchMakes_HomeOnly = 4

SwitchBreaks_HomeOnly = 5
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.KIM_DirectionSense(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

```
Dir_Disabled = 0
Dir_Forward = 1
Dir_Reverse = 2

class msl.equipment.resources.thorlabs.kinesis.enums.KIM_TrigModes(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)

Bases: IntEnum

Trig_Disabled = 0
Trig_In_GPI = 1
Trig_InRelativeMove = 2
Trig_InAbsoluteMove = 3
Trig_InResetCount = 4
Trig_Out_GP0 = 10
Trig_Out_InMotion = 11
Trig_Out_AtMaxVelocity = 12
Trig_Out_PosStepFwd = 13
Trig_Out_PosStepRev = 14
Trig_Out_PosStepBoth = 15
Trig_Out_AtFwdLimit = 16
Trig_Out_AtRevLimit = 17
Trig_Out_AtEitherLimit = 18

class msl.equipment.resources.thorlabs.kinesis.enums.KIM_TrigPolarities(value,
    names=None,
    *values,
    mod-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

Trig_High = 1

Trig_Low = 2

```
class msl.equipment.resources.thorlabs.kinesis.enums.KIM_JoysticModes(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

JS_Velocity = 1

JS_Jog = 2

JS_GotoPosition = 3

```
class msl.equipment.resources.thorlabs.kinesis.enums.ChannelEnableModes(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

NONE = 0

Channel10only = 1

Channel12only = 2

Channel13only = 3

Channel14only = 4

Channels1and2 = 5

Channels3and4 = 6

```
class msl.equipment.resources.thorlabs.kinesis.enums.LD_InputSourceFlags(value,
    names=None,
    *values,
    module=None,
    qualname=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

LD_SoftwareOnly = 1

LD_ExternalSignal = 2

LD_Potentiometer = 4

```
class msl.equipment.resources.thorlabs.kinesis.enums.LD_DisplayUnits(value,
    names=None,
    *values,
    module=None,
    qualname=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

LD_ILim = 1

LD_ILD = 2

LD_IPD = 3

LD_PLD = 4

```
class msl.equipment.resources.thorlabs.kinesis.enums.LD_TIA_RANGES(value,
    names=None,
    *values,
    module=None,
    qualname=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

```
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(value,
    names=None,
    *values,
    module=None,
    qualname=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

```
LD_CathodeGrounded = 1
```

```
LD_AnodeGrounded = 2
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.KLD_TriggerMode(value,
    names=None,
    *values,
    module=None,
    qualname=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

```
KLD_Disabled = 0
```

```
KLD_Input = 1
```

```
KLD_Output = 10
```

```
KLD_LaserOn = 11
```

```
KLD_InterlockEnabled = 12
```

```
KLD_SetPointChange = 13
```

```
KLD_HighStability = 14
```

```
KLD_LowStability = 15
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.KLD_TrigPolarity(value,
    names=None,
    *values,
    module=None,
    qualname=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

```
KLD_TrigPol_High = 1
KLD_TrigPol_Low = 2
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.LS_InputSourceFlags(value,
    names=None,
    *values,
    module=None,
    qualname=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

```
LS_SoftwareOnly = 0
LS_ExternalSignal = 1
LS_Potentiometer = 4
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.KLS_OpMode(value,
    names=None,
    *values,
    module=None,
    qualname=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

```
KLS_ConstantPower = 0
KLS_ConstantCurrent = 1
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.KLS_TriggerMode(value,
    names=None,
    *values,
    mod-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

KLS_Disabled = 0

KLS_Input = 1

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(value,
    names=None,
    *values,
    mod-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

KLS_TrigPol_High = 1

KLS_TrigPol_Low = 2

```
class msl.equipment.resources.thorlabs.kinesis.enums.KNA_FeedbackSource(value,
                           names=None,
                           *values,
                           module=None,
                           qual-name=None,
                           type=None,
                           start=1,
                           boundary=None)
```

Bases: [IntEnum](#)

```
NT_FeedbackSourceUndefined = 0
NT_TIA = 1
NT_I01_5v = 4
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.KNA_TIARange(value,
                     names=None,
                     *values,
                     module=None,
                     qual-name=None,
                     type=None,
                     start=1, boundary=None)
```

Bases: [IntEnum](#)

```
KNA_TIARange1_5nA = 3
KNA_TIARange2_16_6nA = 4
KNA_TIARange3_50nA = 5
KNA_TIARange4_166nA = 6
KNA_TIARange5_500nA = 7
KNA_TIARange6_1_66uA = 8
KNA_TIARange7_5uA = 9
KNA_TIARange8_16_6uA = 10
KNA_TIARange9_50uA = 11
KNA_TIARange10_166uA = 12
KNA_TIARange11_500uA = 13
KNA_TIARange12_1_66mA = 14
KNA_TIARange13_5mA = 15
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.KNA_LowVoltageRange(value,
                           names=None,
                           *values,
                           mode=None,
                           quality=None,
                           name=None,
                           type=None,
                           start=1,
                           boundary=None)
```

Bases: [IntEnum](#)

KNA_VoltageRange_10v = 2

```
class msl.equipment.resources.thorlabs.kinesis.enums.KNA_LowOutputVoltageRoute(value,
                           names=None,
                           *values,
                           mode=None,
                           quality=None,
                           name=None,
                           type=None,
                           start=1,
                           boundary=None)
```

Bases: [IntEnum](#)

KNA_I010only = 1

```
class msl.equipment.resources.thorlabs.kinesis.enums.KNA_HighVoltageRange(value,
                           names=None,
                           *values,
                           mode=None,
                           quality=None,
                           name=None,
                           type=None,
                           start=1,
                           boundary=None)
```

Bases: [IntEnum](#)

KNA_Default_Range = 0

KNA_VoltageRange_CH1_75v = 0

KNA_VoltageRange_CH1_150v = 1

KNA_VoltageRange_CH2_75v = 0

KNA_VoltageRange_CH2_150v = 16

```
class msl.equipment.resources.thorlabs.kinesis.enums.KNA_HighOutputVoltageRoute(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

KNA_Default_Route = 0

KNA_ExtIn_PIN = 0

KNA_ExtIn_IO1 = 1

KNA_ExtOut_Dis = 0

KNA_ExtOut_IO2 = 16

KNA_EnableInputBoost = 256

```
class msl.equipment.resources.thorlabs.kinesis.enums.NT_IO1_Units(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1, bound-
    ary=None)
```

Bases: [IntEnum](#)

NT_Voltage = 0

NT_FullRange = 1

NT_UserDefined = 2

```
class msl.equipment.resources.thorlabs.kinesis.enums.KNA_WheelAdjustRate(value,
                           names=None,
                           *values,
                           mode=None,
                           qual-
                           name=None,
                           type=None,
                           start=1,
                           bound-
                           ary=None)
```

Bases: [IntEnum](#)

KNA_WM_Low = 0

KNA_WM_Medium = 1

KNA_WM_High = 2

```
class msl.equipment.resources.thorlabs.kinesis.enums.KNA_TriggerPortMode(value,
                           names=None,
                           *values,
                           mode=None,
                           qual-
                           name=None,
                           type=None,
                           start=1,
                           bound-
                           ary=None)
```

Bases: [IntEnum](#)

KNA_TrigDisabled = 0

KNA_TrigIn_GPI = 1

KNA_TrigIn_VoltageStepUp = 2

KNA_TrigIn_VoltageStepDown = 3

KNA_TrigOut_GPO = 10

```
class msl.equipment.resources.thorlabs.kinesis.enums.KNA_TriggerPortPolarity(value,
    names=None,
    *values,
    module=None,
    qual-name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

KNA_TrigPolarityHigh = 1

KNA_TrigPolarityLow = 2

```
class msl.equipment.resources.thorlabs.kinesis.enums.KNA_Channels(value,
    names=None,
    *values,
    module=None,
    qual-name=None,
    type=None,
    start=1, boundary=None)
```

Bases: [IntEnum](#)

KNA_ChannelUndefined = 0

KNA_Channel1 = 1

KNA_Channel2 = 2

```
class msl.equipment.resources.thorlabs.kinesis.enums.KNA_FeedbackModeTypes(value,
    names=None,
    *values,
    module=None,
    qual-name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

PZ_ControlModeUndefined = 0

PZ_OpenLoop = 1

```
PZ_CloseLoop = 2
```

```
PZ_OpenLoopSmooth = 3
```

```
PZ_CloseLoopSmooth = 4
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.KPZ_WheelDirectionSense(value,
    names=None,
    *values,
    module=None,
    qual-name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

```
KPZ_WM_Positive = 1
```

```
KPZ_WM_Negative = 2
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.KPZ_WheelMode(value,
    names=None,
    *values,
    module=None,
    qual-name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

```
KPZ_WM_MoveAtVoltage = 1
```

```
KPZ_WM_JogVoltage = 2
```

```
KPZ_WM_SetVoltage = 3
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.KPZ_WheelChangeRate(value,
    names=None,
    *values,
    module=None,
    qual-name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

KPZ_WM_High = 1

KPZ_WM_Medium = 2

KPZ_WM_Low = 3

```
class msl.equipment.resources.thorlabs.kinesis.enums.KPZ_TriggerPortMode(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

KPZ_TrigPolarityHigh = 1

KPZ_TrigPolarityLow = 2

```
class msl.equipment.resources.thorlabs.kinesis.enums.HubAnalogueModes(value,
    names=None,
    *values,
    module=None,
    qual-name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

AnalogueCh1 = 1

AnalogueCh2 = 2

ExtSignalSMA = 3

```
class msl.equipment.resources.thorlabs.kinesis.enums.QD_OperatingMode(value,
    names=None,
    *values,
    module=None,
    qual-name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

QD_ModeUndefined = 0

QD_Monitor = 1

QD_OpenLoop = 2

QD_ClosedLoop = 3

QD_AutoOpenClosedLoop = 4

```
class msl.equipment.resources.thorlabs.kinesis.enums.QD_LowVoltageRoute(value,
    names=None,
    *values,
    module=None,
    qual-name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

```
QD_RouteUndefined = 0
```

```
QD_SMAOnly = 1
```

```
QD_HubAndSMA = 2
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.QD_OpenLoopHoldValues(value,
                           names=None,
                           *values,
                           module=None,
                           qualname=None,
                           type=None,
                           start=1,
                           boundary=None)
```

Bases: [IntEnum](#)

```
QD_HoldOnZero = 1
```

```
QD_HoldOnLastValue = 2
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.QD_FilterEnable(value,
                       names=None,
                       *values,
                       module=None,
                       qualname=None,
                       type=None,
                       start=1,
                       boundary=None)
```

Bases: [IntEnum](#)

```
QD_Undefined = 0
```

```
QD_Enabled = 1
```

```
QD_Disabled = 2
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.QD_KPA_TrigModes(value,
                       names=None,
                       *values,
                       module=None,
                       qualname=None,
                       type=None,
                       start=1,
                       boundary=None)
```

Bases: [IntEnum](#)

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(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

GD_Trig_High = 1

GD_Trig_Low = 2

```
class msl.equipment.resources.thorlabs.kinesis.enums.SC_OperatingModes(value,
    names=None,
    *values,
    mode=None,
    quality=None,
    name=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

SC_Manual = 1

SC_Single = 2

SC_Auto = 3

SC_Triggered = 4

```
class msl.equipment.resources.thorlabs.kinesis.enums.SC_OperatingStates(value,
                           names=None,
                           *values,
                           mode=None,
                           quality=None,
                           name=None,
                           type=None,
                           start=1,
                           boundary=None)
```

Bases: [IntEnum](#)

SC_Unknown = 0

SC_Active = 1

SC_Inactive = 2

```
class msl.equipment.resources.thorlabs.kinesis.enums.SC_SolenoidStates(value,
                           names=None,
                           *values,
                           mode=None,
                           quality=None,
                           name=None,
                           type=None,
                           start=1,
                           boundary=None)
```

Bases: [IntEnum](#)

SC_SolenoidOpen = 1

SC_SolenoidClosed = 2

```
class msl.equipment.resources.thorlabs.kinesis.enums.KSC_TriggerPortMode(value,
                           names=None,
                           *values,
                           mode=None,
                           quality=None,
                           name=None,
                           type=None,
                           start=1,
                           boundary=None)
```

Bases: [IntEnum](#)

KSC_TrigDisabled = 0

KSC_TrigIn_GPI = 1

KSC_TrigOut_GPO = 10

```
class msl.equipment.resources.thorlabs.kinesis.enums.KSC_TriggerPortPolarity(value,
                                                                           names=None,
                                                                           *values,
                                                                           module=None,
                                                                           qualname=None,
                                                                           type=None,
                                                                           start=1,
                                                                           boundary=None)
```

Bases: [IntEnum](#)

KSC_TrigPolarityHigh = 1

KSC_TrigPolarityLow = 2

```
class msl.equipment.resources.thorlabs.kinesis.enums.KST_Stages(value,
                                                                names=None,
                                                                *values,
                                                                module=None,
                                                                qualname=None,
                                                                type=None,
                                                                start=1,
                                                                boundary=None)
```

Bases: [IntEnum](#)

ZST6 = 32

ZST13 = 33

ZST25 = 34

ZST206 = 48

ZST213 = 49

ZST225 = 50

ZFS206 = 64

ZFS213 = 65

ZFS225 = 66

DRV013_25MM = 80

DRV014_50MM = 81

NR360 = 112

PLS_X25MM = 114

```
PLS_X25MM_HiRes = 115
```

```
FW103 = 117
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.TSG_Hub_Analogue_Modes(value,  

    names=None,  

    *values,  

    mod-  

    ule=None,  

    qual-  

    name=None,  

    type=None,  

    start=1,  

    bound-  

    ary=None)
```

Bases: [IntEnum](#)

```
TSG_HubChannel1 = 1
```

```
TSG_HubChannel2 = 2
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.TSG_Display_Modes(value,  

    names=None,  

    *values,  

    mod-  

    ule=None,  

    qual-  

    name=None,  

    type=None,  

    start=1,  

    bound-  

    ary=None)
```

Bases: [IntEnum](#)

```
TSG_Undefined = 0
```

```
TSG_Position = 1
```

```
TSG_Voltage = 2
```

```
TSG_Force = 3
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.KSG_TriggerPortMode(value,  

    names=None,  

    *values,  

    mod-  

    ule=None,  

    qual-  

    name=None,  

    type=None,  

    start=1,  

    bound-  

    ary=None)
```

Bases: [IntEnum](#)

KSG_TrigDisabled = 0

KSG_TrigIn_GPI = 1

KSG_TrigOut_GPO = 10

KSG_TrigOut_LessThanLowerLimit = 11

KSG_TrigOut_MoreThanLowerLimit = 12

KSG_TrigOut_LessThanUpperLimit = 13

KSG_TrigOut_MoreThanUpperLimit = 14

KSG_TrigOut_BetweenLimits = 15

KSG_TrigOut_OutsideLimits = 16

```
class msl.equipment.resources.thorlabs.kinesis.enums.KSG_TriggerPortPolarity(value,
                                                                           names=None,
                                                                           *values,
                                                                           module=None,
                                                                           qual-
                                                                           name=None,
                                                                           type=None,
                                                                           start=1,
                                                                           bound-
                                                                           ary=None)
```

Bases: [IntEnum](#)

KSG_TrigPolarityHigh = 1

KSG_TrigPolarityLow = 2

```
class msl.equipment.resources.thorlabs.kinesis.enums.TIM_Channels(value,
                                                                   names=None,
                                                                   *values,
                                                                   module=None,
                                                                   qual-
                                                                   name=None,
                                                                   type=None,
                                                                   start=1, bound-
                                                                   ary=None)
```

Bases: [IntEnum](#)

Channel1 = 1

Channel2 = 2

Channel3 = 3

Channel14 = 4

```
class msl.equipment.resources.thorlabs.kinesis.enums.TIM_JogMode(value,
    names=None,
    *values,
    module=None,
    qualname=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

JogContinuous = 1

JogStep = 2

```
class msl.equipment.resources.thorlabs.kinesis.enums.TIM_ButtonsMode(value,
    names=None,
    *values,
    module=None,
    qualname=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

Jog = 1

Position = 2

```
class msl.equipment.resources.thorlabs.kinesis.enums.TIM_Direction(value,
    names=None,
    *values,
    module=None,
    qualname=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

Forward = 1

Reverse = 2

```
class msl.equipment.resources.thorlabs.kinesis.enums.LS_DisplayUnits(value,
    names=None,
    *values,
    module=None,
    qualname=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

LS_mAmps = 1

LS_mWatts = 2

LS_mBb = 3

```
class msl.equipment.resources.thorlabs.kinesis.enums.TST_Stages(value,
    names=None,
    *values,
    module=None,
    qualname=None,
    type=None,
    start=1,
    boundary=None)
```

Bases: [IntEnum](#)

ZST6 = 32

ZST13 = 33

ZST25 = 34

ZST206 = 48

ZST213 = 49

ZST225 = 50

ZFS206 = 64

ZFS213 = 65

ZFS225 = 66

TBD1 = 96

TBD2 = 97

TBD3 = 98

TBD4 = 99

NR360 = 112

```
MVS025 = 113
PLS_X25MM = 114
PLS_X25MM_HiRes = 115
FW103 = 117
NEWZFS06 = 10006
NEWZFS13 = 10013
NEWZFS25 = 10025
NEWZST06 = 11006
NEWZST13 = 11013
NEWZST25 = 12025
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.TC_SensorTypes(value,
    names=None,
    *values,
    module=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

```
TC_Transducer = 0
TC_TH20kOhm = 1
TC_TH200kOhm = 2
```

```
class msl.equipment.resources.thorlabs.kinesis.enums.TC_DisplayModes(value,
    names=None,
    *values,
    mod-
    ule=None,
    qual-
    name=None,
    type=None,
    start=1,
    bound-
    ary=None)
```

Bases: [IntEnum](#)

```
TC_ActualTemperature = 0
TC_TargetTemperature = 1
```

```
TC_TempDifference = 2
TC_Current = 3

class msl.equipment.resources.thorlabs.kinesis.enums.UnitType(value, names=None,
                                                               *values,
                                                               module=None,
                                                               qualname=None,
                                                               type=None, start=1,
                                                               boundary=None)

Bases: IntEnum
DISTANCE = 0
VELOCITY = 1
ACCELERATION = 2
```

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

Device and Low Level Error Codes defined in Thorlabs Kinesis v1.14.10

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

This module provides all the functionality required to control a Filter Flipper (MFF101, MFF102).

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

A wrapper around Thorlabs.MotionControl.FilterFlipper.dll.

The *properties* for a FilterFlipper connection supports the following key-value pairs in the *Connections Database*:

```
'device_name': str, the device name found in ThorlabsDefaultSettings.xml
↳ [default: None]
```

Do not instantiate this class directly. Use the `connect()` method to connect to the equipment.

Parameters

`record` (*EquipmentRecord*) – A 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

ThorlabsError – 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

ThorlabsError – 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

ThorlabsError – If not successful.

load_named_settings(*settings_name*)

Update device with named settings.

Parameters

`settings_name (str)` – The name of the device to load the settings for.
Examples for the value of `setting_name` can be found in `ThorlabsDefaultSettings.xml`, which gets created when the Kinesis software is installed.

Raises

ThorlabsError – If not successful.

`persist_settings()`

Persist the devices current settings.

Raises

ThorlabsError – 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

ThorlabsError – If not successful.

`move_to_position(position)`

Move the device to the specified position (index).

Parameters

`position (int)` – The required position. Must be 1 or 2.

Raises

ThorlabsError – If not successful.

`get_position()`

Get the current position.

Returns

`int` – The position, 1 or 2 (can be 0 during a move).

`get_io_settings()`

Gets the I/O settings from filter flipper.

Returns

`FF_IOSettings` – The Filter Flipper I/O settings.

Raises

ThorlabsError – If not successful.

`request_io_settings()`

Requests the I/O settings from the filter flipper.

Raises

ThorlabsError – If not successful.

**`set_io_settings(transit_time=500, oper1=FF_IOModes.FF_ToggleOnPositiveEdge,
sig1=FF_SignalModes.FF_InputButton, pw1=200,
oper2=FF_IOModes.FF_ToggleOnPositiveEdge,
sig2=FF_SignalModes.FF_OutputLevel, pw2=200)`**

Sets the settings on filter flipper.

Parameters

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

Raises

`ThorlabsError` – 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

`ThorlabsError` – 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

`ThorlabsError` – 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

`ThorlabsError` – 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 the 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`, optional) – 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 its settings to the DLL.

Raises

`ThorlabsError` – If not successful.

`clear_message_queue()`

Clears the device message queue.

`register_message_callback(callback)`

Registers a callback on the message queue.

Parameters

`callback` (`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

`ThorlabsError` – 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

`ThorlabsError` – 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, MLJ150)
- Cage Rotator (K10CR1)

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

Bases: `MotionControl`

A wrapper around `Thorlabs.MotionControl.IntegratedStepperMotors.dll`.

The `properties` for an `IntegratedStepperMotors` connection supports the following key-value pairs in the `Connections Database`:

```
'device_name': str, the device name found in ThorlabsDefaultSettings.xml
↳ [default: None]
```

Do not instantiate this class directly. Use the `connect()` method to connect to the equipment.

Parameters

record (*EquipmentRecord*) – A 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

ThorlabsError – 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

ThorlabsError – 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 DeviceUnit to a RealValue.

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 DeviceUnit to a RealValue.

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

`ThorlabsError` – If not successful.

get_button_params_block()

Get the button parameters.

Returns

`structs.MOT_ButtonParameters` – The button parameters.

Raises

`ThorlabsError` – If not successful.

get_calibration_file()

Get the calibration file for this motor.

Returns

`str` – The filename of the calibration file.

Raises

`ThorlabsError` – 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()`, `load_named_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** (`enums.UnitType`) – The unit of the real-world value.

Returns

`int` – The device value.

Raises

`ThorlabsError` – 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

`ThorlabsError` – If not successful.

get_hardware_info_block()

Gets the hardware information in a block.

Returns

`structs.TLI_HardwareInformation` – The hardware information.

Raises

`ThorlabsError` – If not successful.

get_homing_params_block()

Get the homing parameters.

Returns

`structs.MOT_HomingParameters` – The homing parameters.

Raises

`ThorlabsError` – If not successful.

get_homing_velocity()

Gets the homing velocity.

See `get_real_value_from_device_unit()` for converting from a `DeviceUnit` to a `RealValue`.

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

`ThorlabsError` – If not successful.

get_jog_params_block()

Get the jog parameters.

Returns

`structs.MOT_JogParameters` – The jog parameters.

Raises

`ThorlabsError` – 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 DeviceUnit to a RealValue.

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 DeviceUnit to a RealValue.

Returns

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

Raises

`ThorlabsError` – 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 DeviceUnit to a RealValue.

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

`ThorlabsError` – If not successful.

get_limit_switch_params_block()

Get the limit switch parameters.

Returns

`structs.MOT_LimitSwitchParameters` – The limit switch parameters.

Raises

`ThorlabsError` – If not successful.

get_motor_params()

Gets the motor stage parameters.

Deprecated: calls `get_motor_params_ext()`

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

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

Raises

`ThorlabsError` – 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

`ThorlabsError` – 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

`ThorlabsError` – If not successful.

get_move_absolute_position()

Gets the move absolute position.

See `get_real_value_from_device_unit()` for converting from a DeviceUnit to a RealValue.

Returns

`int` – The move 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 DeviceUnit to a RealValue.

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

`ThorlabsError` – If not successful.

get_number_positions()

Get the 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 DeviceUnit to a RealValue.

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 DeviceUnit to a RealValue.

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 DeviceUnit to a RealValue.

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

`ThorlabsError` – If not successful.

get_potentiometer_params_block()

Get the potentiometer parameters.

Returns

`structs.MOT_PotentiometerSteps` – The potentiometer parameters.

Raises

`ThorlabsError` – If not successful.

get_power_params()

Gets the power parameters for the stepper motor.

Returns

`structs.MOT_PowerParameters` – The power parameters.

Raises

`ThorlabsError` – 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()`, `load_named_settings()` or `set_motor_params_ext()` must be called before calling this function, otherwise the returned value will always be 0.

Parameters

- `device_value (int)` – The device value.

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

Returns

float – The real-world value.

Raises

ThorlabsError – 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 DeviceUnit to a RealValue.

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 DeviceUnit to a RealValue.

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 function, [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 DeviceUnit to a RealValue.

Returns

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

Raises

`ThorlabsError` – If not successful.

get_vel_params_block()

Get the move velocity parameters.

Returns

`structs.MOT_VelocityParameters` – The velocity parameters.

Raises

`ThorlabsError` – 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, `False` if monitoring is not enabled or if time of last message received is less than `lastMsgTimeout`.

home()

Home the device.

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

Raises

`ThorlabsError` – If not successful.

identify()

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

Raises

`ThorlabsError` – If not successful.

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

`ThorlabsError` – If not successful.

load_named_settings(*settings_name*)

Update device with named settings.

Parameters

settings_name (`str`) – The name of the device to load the settings for.
Examples for the value of *setting_name* can be found in *ThorlabsDefaultSettings.xml*, which gets created when the Kinesis software is installed.

Raises

ThorlabsError – 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

ThorlabsError – 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

ThorlabsError – 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

ThorlabsError – If not successful.

move_relative(*displacement*)

Move the motor by a relative amount.

See [get_device_unit_from_real_value\(\)](#) for converting from a `RealValue` to a `DeviceUnit`.

Parameters

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

Raises

ThorlabsError – If not successful.

move_relative_distance()

Moves the device by a relative distance defined by [set_move_relative_distance\(\)](#).

Raises

ThorlabsError – If not successful.

move_to_position(index)

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 `RealValue` to a `DeviceUnit`.

Parameters

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

Raises

ThorlabsError – 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

ThorlabsError – If not successful.

persist_settings()

Persist the devices current settings.

Raises

ThorlabsError – 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 (MotionControlCallback)` – A function to be called whenever messages are received.

request_backlash()

Requests the backlash.

Raises

ThorlabsError – If not successful.

request_bow_index()

Requests the stepper motor bow index.

Raises

ThorlabsError – If not successful.

request_button_params()

Requests the LTS button parameters.

Raises

ThorlabsError – If not successful.

request_homing_params()

Requests the homing parameters.

Raises

ThorlabsError – If not successful.

request_jog_params()

Requests the jog parameters.

Raises

ThorlabsError – If not successful.

request_limit_switch_params()

Requests the limit switch parameters.

Raises

ThorlabsError – If not successful.

request_move_absolute_position()

Requests the position of next absolute move.

Raises

ThorlabsError – If not successful.

request_move_relative_distance()

Requests the relative move distance.

Raises

ThorlabsError – If not successful.

request_position()

Requests the current position.

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

Raises

ThorlabsError – If not successful.

request_potentiometer_params()

Requests the potentiometer parameters.

Raises

ThorlabsError – If not successful.

request_power_params()

Requests the power parameters.

Raises

ThorlabsError – If not successful.

request_settings()

Requests that all settings are downloaded from the device.

This function requests that the device upload all its settings to the DLL.

Raises

ThorlabsError – 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

ThorlabsError – 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 its current status bits. Note, this is called automatically if Polling is enabled for the device using *start_polling()*.

Raises

ThorlabsError – If not successful.

request_trigger_switches()

Requests the trigger switch bits.

Raises

ThorlabsError – If not successful.

request_vel_params()

Requests the velocity parameters.

Raises

ThorlabsError – If not successful.

reset_rotation_modes()

Reset the rotation modes for a rotational device.

Raises

ThorlabsError – If not successful.

reset_stage_to_defaults()

Reset the stage settings to defaults.

Raises

ThorlabsError – 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 RealValue to a DeviceUnit.

Parameters

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

Raises

ThorlabsError – If not successful.

set_bow_index(*bow_index*)

Sets the stepper motor bow index.

Parameters

bow_index (`int`) – The bow index.

Raises

ThorlabsError – 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 `RealValue` to a `DeviceUnit`.

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

ThorlabsError – 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

ThorlabsError – 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

OSError – 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.

Raises

ThorlabsError – If not successful.

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

ThorlabsError – If not successful.

set_homing_velocity(*velocity*)

Sets the homing velocity.

See `get_device_unit_from_real_value()` for converting from a `RealValue` to a `DeviceUnit`.

Parameters

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

Raises

ThorlabsError – 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

ThorlabsError – If not successful.

set_jog_params_block(jog_params)

Set the jog parameters.

Parameters

jog_params (*structs.MOT_JogParameters*) – The jog parameters.

Raises

- **ThorlabsError** – 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 RealValue to a DeviceUnit.

Parameters

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

Raises

ThorlabsError – 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 RealValue to a DeviceUnit.

Parameters

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

Raises

ThorlabsError – 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

ThorlabsError – 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 RealValue to a DeviceUnit.

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

ThorlabsError – If not successful.

set_limit_switch_params_block(*params*)

Set the limit switch parameters.

Parameters

params (*structs.MOT_LimitSwitchParameters*) – The new limit switch parameters.

Raises

- *ThorlabsError* – If not successful.
- *TypeError* – If the data type of *joystick_params* is not *structs.MOT_JoystickParameters*

set_limits_software_approach_policy(*policy*)

Sets the software limits policy.

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 DeviceUnit to a RealValue.

Parameters

- **steps_per_rev** (*float*) – The steps per revolution.
- **gear_box_ratio** (*float*) – The gear box ratio.

- **pitch** (`float`) – The pitch.

Raises

`ThorlabsError` – 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 DeviceUnit to a RealValue.

Parameters

- **steps_per_rev** (`float`) – The steps per revolution.
- **gear_box_ratio** (`float`) – The gear box ratio.
- **pitch** (`float`) – The pitch.

Raises

`ThorlabsError` – 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 DeviceUnit to a RealValue.

Parameters

- **min_position** (`float`) – The minimum position in RealWorldUnits [millimeters or degrees].
- **max_position** (`float`) – The maximum position in RealWorldUnits [millimeters or degrees].

Raises

`ThorlabsError` – 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

`ThorlabsError` – 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 DeviceUnit to a RealValue.

Parameters

- **max_velocity** (`float`) – The maximum velocity in RealWorldUnits [millimeters or degrees].
- **max_acceleration** (`float`) – The maximum acceleration in RealWorldUnits [millimeters or degrees].

Raises

ThorlabsError – If not successful.

set_move_absolute_position(*position*)

Sets the move absolute position.

See `get_device_unit_from_real_value()` for converting from a RealValue to a DeviceUnit.

Parameters

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

Raises

ThorlabsError – If not successful.

set_move_relative_distance(*distance*)

Sets the move relative distance.

See `get_device_unit_from_real_value()` for converting from a RealValue to a DeviceUnit.

Parameters

distance (`int`) – The relative position in DeviceUnits (see manual).

Raises

ThorlabsError – 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 RealValue to a DeviceUnit.

Parameters

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

Raises

ThorlabsError – 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 RealValue to a DeviceUnit.

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

`ThorlabsError` – If not successful.

set_potentiometer_params_block(`params`)

Set the potentiometer parameters.

Parameters

- **params** (`structs.MOT_PotentiometerSteps`) – The potentiometer parameters.

Raises

`ThorlabsError` – 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

`ThorlabsError` – If not successful.

set_rotation_modes(`mode`, `direction`)

Set the rotation modes for a rotational device.

Parameters

- **mode** (`enums.MOT_MovementModes`) – The travel mode as a `enums.MOT_MovementModes` enum value or member name.
- **direction** (`enums.MOT_MovementDirections`) – The travel mode as a `enums.MOT_MovementDirections` enum value or member name.

Raises

`ThorlabsError` – 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 `RealValue` to a `DeviceUnit`.

Parameters

- **min_position** (`int`) – The minimum position in DeviceUnits (see manual).
- **max_position** (`int`) – The maximum position in DeviceUnits (see manual).

Raises

`ThorlabsError` – 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

`ThorlabsError` – 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 `RealValue` to a `DeviceUnit`.

Parameters

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

Raises

`ThorlabsError` – If not successful.

set_vel_params_block(*min_velocity, max_velocity, acceleration*)

Set the move velocity parameters.

See `get_device_unit_from_real_value()` for converting from a `RealValue` to a `DeviceUnit`.

Parameters

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

Raises

`ThorlabsError` – 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

`ThorlabsError` – If not successful.

stop_immediate()

Stop the current move immediately (with the risk of losing track of the position).

Raises

ThorlabsError – If not successful.

stop_polling()

Stops the internal polling loop.

stop_profiled()

Stop the current move using the current velocity profile.

Raises

ThorlabsError – If not successful.

time_since_last_msg_received()

Gets the time, in milliseconds, since the 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.
- **bool** – **True** if monitoring is enabled otherwise **False**.

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

ThorlabsError – If not successful.

msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo module

This module provides all the functionality required to control a KCube DC Servo (KDC101).

class `msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo.KCubeDCServo(record)`

Bases: [MotionControl](#)

A wrapper around `Thorlabs.MotionControl.KCube.DCServo.dll`.

The [properties](#) for a KCubeDCServo connection supports the following key-value pairs in the [Connections Database](#):

```
'device_name': str, the device name found in ThorlabsDefaultSettings.xml
↳ [default: None]
```

Do not instantiate this class directly. Use the `connect()` method to connect to the equipment.

Parameters

record (*EquipmentRecord*) – A record from an [Equipment-Register Database](#).

`can_device_lock_front_panel()`

Determine if the device front panel can be locked.

Returns

`bool` – `True` if the front panel of the device can be locked, `False` if not.

`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

`ThorlabsError` – 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

`ThorlabsError` – 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 DeviceUnit to a RealValue.

Returns

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

get_dcpid_params()

Get the DC PID parameters for DC motors used in an algorithm involving calculus.

Returns

`structs.MOT_DC_PIDParameters` – The DC PID parameters.

Raises

`ThorlabsError` – 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()`, `load_named_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** (`enums.UnitType`) – The unit of the real-world value.

Returns

`int` – The device value.

Raises

`ThorlabsError` – If not successful.

get_digital_outputs()

Gets the digital output bits.

Returns

`bytes` – Bit mask of states of the 4 digital output pins.

get_encoder_counter()

Get the encoder counter.

For devices that have an encoder, the current encoder position can be read.

Returns

`int` – The encoder count in encoder units.

get_front_panel_locked()

Query if the device front panel locked.

Returns

`bool` – `True` if the device front panel is locked, `False` if not.

get_hardware_info()

Gets the hardware information from the device.

Returns

`structs.TLI_HardwareInformation` – The hardware information.

Raises

`ThorlabsError` – If not successful.

get_hardware_info_block()

Gets the hardware information in a block.

Returns

`structs.TLI_HardwareInformation` – The hardware information.

Raises

`ThorlabsError` – If not successful.

get_homing_params_block()

Get the homing parameters.

Returns

`structs.MOT_HomingParameters` – The homing parameters.

Raises

`ThorlabsError` – If not successful.

get_homing_velocity()

Gets the homing velocity.

See `get_real_value_from_device_unit()` for converting from a `DeviceUnit` to a `RealValue`.

Returns

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

get_hub_bay()

Gets the hub bay number this device is fitted to.

Returns

`bytes` – The number, 0x00 if unknown or 0xff if not on a hub.

get_jog_mode()

Gets the jog mode.

Returns

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

Raises

`ThorlabsError` – If not successful.

get_jog_params_block()

Get the jog parameters.

Returns

`structs.MOT_JogParameters` – The jog parameters.

Raises

`ThorlabsError` – 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 DeviceUnit to a RealValue.

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 DeviceUnit to a RealValue.

Returns

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

Raises

`ThorlabsError` – If not successful.

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_limit_switch_params()

Gets the limit switch parameters.

See [get_real_value_from_device_unit\(\)](#) for converting from a DeviceUnit to a RealValue.

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

`ThorlabsError` – If not successful.

get_limit_switch_params_block()

Get the limit switch parameters.

Returns

`structs.MOT_LimitSwitchParameters` – The limit switch parameters.

Raises

ThorlabsError – If not successful.

get_mmi_params()

Get the MMI Parameters for the KCube Display Interface.

Deprecated calls by *get_mmi_params_ext()*

get_mmi_params_block()

Gets the MMI parameters for the device.

Returns

structs.KMOT_MMIParams – The MMI parameters for the device.

get_mmi_params_ext()

Get the MMI Parameters for the KCube Display Interface.

See *get_real_value_from_device_unit()* for converting from a DeviceUnit to a RealValue.

Returns

- *enums.KMOT_WheelMode* – The device joystick mode.
- *int* – The joystick maximum velocity in DeviceUnits.
- *int* – The joystick acceleration in DeviceUnits.
- *enums.KMOT_WheelDirectionSense* – The joystick direction sense.
- *int* – The first preset position in DeviceUnits.
- *int* – The second preset position in DeviceUnits.
- *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

ThorlabsError – 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

ThorlabsError – 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

ThorlabsError – 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

ThorlabsError – 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

ThorlabsError – If not successful.

get_move_absolute_position()

Gets the move absolute position.

See `get_real_value_from_device_unit()` for converting from a DeviceUnit to a RealValue.

Returns

`int` – The move 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 DeviceUnit to a RealValue.

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

`ThorlabsError` – If not successful.

get_number_positions()

Get the 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 DeviceUnit to a RealValue.

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 DeviceUnit to a RealValue.

Returns

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

get_real_value_from_device_unit(device_value, unit_type)

Converts a device value to a real-world value.

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

Parameters

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

Returns

`float` – The real-world value.

Raises

`ThorlabsError` – 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 Stepper Motor maximum stage position.

See `get_real_value_from_device_unit()` for converting from a `DeviceUnit` to a `RealValue`.

Returns

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

get_stage_axis_min_pos()

Gets the Stepper Motor minimum stage position.

See `get_real_value_from_device_unit()` for converting from a `DeviceUnit` to a `RealValue`.

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 the polling functions, `start_polling()`.

Returns

`int` – The status bits from the device.

get_trigger_config_params()

Get the Trigger Configuration Parameters.

Returns

- `enums.KMOT_TriggerPortMode` – The trigger 1 mode.
- `enums.KMOT_TriggerPortPolarity` – The trigger 1 polarity.

- `enums.KMOT_TriggerPortMode` – The trigger 2 mode.
- `enums.KMOT_TriggerPortPolarity` – The trigger 2 polarity.

Raises

`ThorlabsError` – If not successful.

get_trigger_config_params_block()

Gets the trigger configuration parameters block.

Returns

`structs.KMOT_TriggerConfig` – Options for controlling the trigger configuration.

Raises

`ThorlabsError` – If not successful.

get_trigger_params_params()

Get the Trigger Parameters parameters.

See `get_real_value_from_device_unit()` for converting from a `DeviceUnit` to a `RealValue`.

Returns

- `int` – The trigger start position, forward, in `DeviceUnits` (see manual).
- `int` – The trigger interval, forward, in `DeviceUnits` (see manual).
- `int` – Number of trigger pulses, forward.
- `int` – The trigger start position, reverse, in `DeviceUnits` (see manual).
- `int` – The trigger interval, reverse, in `DeviceUnits` (see manual).
- `int` – Number of trigger pulses, reverse.
- `int` – Width of the trigger pulse in milliseconds, range 10 (10us) to 650000 (650ms).
- `int` – Number of cycles to perform triggering.

Raises

`ThorlabsError` – If not successful.

get_trigger_params_params_block()

Gets the trigger parameters block.

Returns

`structs.KMOT_TriggerParams` – Options for controlling the trigger.

Raises

`ThorlabsError` – If not successful.

get_vel_params()

Gets the move velocity parameters.

See `get_real_value_from_device_unit()` for converting from a `DeviceUnit` to a `RealValue`.

Returns

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

Raises

`ThorlabsError` – If not successful.

get_vel_params_block()

Get the move velocity parameters.

Returns

`structs.MOT_VelocityParameters` – The velocity parameters.

Raises

`ThorlabsError` – 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()

Home the device.

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

Raises

`ThorlabsError` – If not successful.

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

`ThorlabsError` – If not successful.

load_named_settings(`settings_name`)

Update device with named settings.

Parameters

`settings_name` (`str`) – The name of the device to load the settings for.

Examples for the value of `setting_name` can be found in `ThorlabsDefaultSettings.xml`, which gets created when the Kinesis software is installed.

Raises

`ThorlabsError` – 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

`ThorlabsError` – 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

`ThorlabsError` – 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

`ThorlabsError` – If not successful.

move_relative(displacement)

Move the motor by a relative amount.

See `get_device_unit_from_real_value()`/`get_device_unit_from_real_value()` for converting from a `RealValue` to a `DeviceUnit`.

Parameters

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

Raises

`ThorlabsError` – If not successful.

move_relative_distance()

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

Raises

`ThorlabsError` – If not successful.

move_to_position(index)

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 `RealValue` to a `DeviceUnit`.

Parameters

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

Raises

`ThorlabsError` – 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

`ThorlabsError` – If not successful.

persist_settings()

Persist the devices current settings.

Raises

`ThorlabsError` – 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 (MotionControlCallback)` – A function to be called whenever messages are received.

request_backlash()

Requests the backlash.

Raises

`ThorlabsError` – If not successful.

request_dcpid_params()

Request the PID parameters for DC motors used in an algorithm involving calculus.

Raises

`ThorlabsError` – If not successful.

request_digital_outputs()

Requests the digital output bits.

Raises

`ThorlabsError` – If not successful.

request_encoder_counter()

Requests the encoder counter.

Raises

ThorlabsError – If not successful.

request_front_panel_locked()

Ask the device if its front panel is locked.

Raises

ThorlabsError – If not successful.

request_homing_params()

Requests the homing parameters.

Raises

ThorlabsError – If not successful.

request_jog_params()

Requests the jog parameters.

Raises

ThorlabsError – If not successful.

request_led_switches()

Requests the LED indicator bits on the cube.

Raises

ThorlabsError – If not successful.

request_limit_switch_params()

Requests the limit switch parameters.

Raises

ThorlabsError – If not successful.

request_mmi_params()

Requests the MMI Parameters for the KCube Display Interface.

Raises

ThorlabsError – If not successful.

request_move_absolute_position()

Requests the position of next absolute move.

Raises

ThorlabsError – If not successful.

request_move_relative_distance()

Requests the relative move distance.

Raises

ThorlabsError – If not successful.

request_pos_trigger_params()

Requests the position trigger parameters.

Raises

ThorlabsError – 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

ThorlabsError – 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

ThorlabsError – 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

ThorlabsError – If not successful.

request_trigger_config_params()

Requests the Trigger Configuration Parameters.

Raises

ThorlabsError – If not successful.

request_vel_params()

Requests the velocity parameters.

Raises

ThorlabsError – If not successful.

reset_rotation_modes()

Reset the rotation modes for a rotational device.

Raises

ThorlabsError – If not successful.

reset_stage_to_defaults()

Reset the stage settings to defaults.

Raises

ThorlabsError – If not successful.

resume_move_messages()

Resume suspended move messages.

Raises

ThorlabsError – 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 `RealValue` to a `DeviceUnit`.

Parameters

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

Raises

`ThorlabsError` – If not successful.

set_dcpid_params(*params*)

Set the PID parameters for DC motors used in an algorithm involving calculus.

Parameters

`params (structs.MOT_DC_PIDParameters)` – The DC PID parameters.

Raises

- `ThorlabsError` – If not successful.
- `TypeError` – If the data type of `params` is not `structs.MOT_DC_PIDParameters`

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

`ThorlabsError` – If not successful.

set_direction(*reverse*)

Sets the motor direction sense.

This function is used because some actuators 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.

Raises

`ThorlabsError` – If not successful.

set_encoder_counter(*count*)

Set the Encoder Counter values.

Setting the encoder counter to zero, effectively defines a home position on the encoder strip. Note, setting this value does not move the device.

Parameters

`count (int)` – The encoder count in encoder units.

Raises

`ThorlabsError` – If not successful.

set_front_panel_lock(*locked*)

Sets the device front panel lock state.

Parameters

locked (`bool`) – `True` to lock the device, `False` to unlock

Raises

`ThorlabsError` – If not successful.

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

`ThorlabsError` – If not successful.

set_homing_velocity(*velocity*)

Sets the homing velocity.

See `get_device_unit_from_real_value()` for converting from a `RealValue` to a `DeviceUnit`.

Parameters

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

Raises

`ThorlabsError` – 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

`ThorlabsError` – If not successful.

set_jog_params_block(*jog_params*)

Set the jog parameters.

Parameters

jog_params (`structs.MOT_JogParameters`) – The jog parameters.

Raises

- **ThorlabsError** – 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 RealValue to a DeviceUnit.

Parameters

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

Raises

ThorlabsError – 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 RealValue to a DeviceUnit.

Parameters

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

Raises

ThorlabsError – 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

ThorlabsError – 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 RealValue to a DeviceUnit.

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

`ThorlabsError` – If not successful.

set_limit_switch_params_block(*params*)

Set the limit switch parameters.

Parameters

params (`structs.MOT_LimitSwitchParameters`) – The limit switch parameters.

Raises

`ThorlabsError` – 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_mmi_params(*joystick_mode*, *joystick_max_velocity*, *joystick_acceleration*, *direction_sense*, *preset_position1*, *preset_position2*, *display_intensity*)

Set the MMI Parameters for the KCube Display Interface.

Deprecated calls `set_mmi_params_ext()` setting the `display_timeout` to 1 minute and the `display_dim_intensity` to 8.

Raises

`ThorlabsError` – If not successful.

set_mmi_params_block(*mmi_params*)

Sets the MMI parameters for the device.

Parameters

mmi_params (`structs.KMOT_MMIParams`) – Options for controlling the mmi.

Raises

`ThorlabsError` – If not successful.

set_mmi_params_ext(*joystick_mode*, *joystick_max_velocity*, *joystick_acceleration*, *direction_sense*, *preset_position1*, *preset_position2*, *display_intensity*, *display_timeout*, *display_dim_intensity*)

Set the MMI Parameters for the KCube Display Interface.

See `get_real_value_from_device_unit()` for converting from a `DeviceUnit` to a `RealValue`.

Parameters

- **joystick_mode** (`enums.KMOT_WheelMode`) – The device joystick mode as a `enums.KMOT_WheelMode` enum value or member name.
- **joystick_max_velocity** (`int`) – The joystick maximum velocity in DeviceUnits.
- **joystick_acceleration** (`int`) – The joystick acceleration in DeviceUnits.
- **direction_sense** (`enums.KMOT_WheelDirectionSense`) – The joystick direction sense as a `enums.KMOT_WheelDirectionSense` enum value or member name.
- **preset_position1** (`int`) – The first preset position in DeviceUnits.
- **preset_position2** (`int`) – The second preset position in DeviceUnits.
- **display_intensity** (`int`) – The display intensity, range 0 to 100%.
- **display_timeout** (`int`) – The display timeout, range 0 to 480 in minutes (0 is off, otherwise the inactivity period before dimming the display).
- **display_dim_intensity** (`int`) – The display dimmed intensity, range 0 to 10 (after the timeout period the device display will dim).

Raises

ThorlabsError – If not successful.

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 DeviceUnit to a RealValue.

Parameters

- **steps_per_rev** (`float`) – The steps per revolution.
- **gear_box_ratio** (`float`) – The gear box ratio.
- **pitch** (`float`) – The pitch.

Raises

ThorlabsError – 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 DeviceUnit to a RealValue.

Parameters

- `steps_per_rev` (`float`) – The steps per revolution.
- `gear_box_ratio` (`float`) – The gear box ratio.
- `pitch` (`float`) – The pitch.

Raises

`ThorlabsError` – 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 DeviceUnit to a RealValue.

Parameters

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

Raises

`ThorlabsError` – 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

`ThorlabsError` – 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 DeviceUnit to a RealValue.

Parameters

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

Raises

`ThorlabsError` – If not successful.

set_move_absolute_position(*position*)

Sets the move absolute position.

See [get_device_unit_from_real_value\(\)](#) for converting from a RealValue to a DeviceUnit.

Parameters

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

Raises

ThorlabsError – If not successful.

set_move_relative_distance(*distance*)

Sets the move relative distance.

See [get_device_unit_from_real_value\(\)](#) for converting from a RealValue to a DeviceUnit.

Parameters

distance (`int`) – The relative position in DeviceUnits (see manual).

Raises

ThorlabsError – 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 RealValue to a DeviceUnit.

Parameters

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

Raises

ThorlabsError – If not successful.

set_rotation_modes(*mode*, *direction*)

Set the rotation modes for a rotational device.

Parameters

- **mode** ([enums.MOT_MovementModes](#)) – The travel mode as a [enums.MOT_MovementModes](#) enum value or member name.
- **direction** ([enums.MOT_MovementDirections](#)) – The travel mode as a [enums.MOT_MovementDirections](#) enum value or member name.

Raises

ThorlabsError – 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 RealValue to a DeviceUnit.

Parameters

- **min_position** (`int`) – The minimum position in DeviceUnits (see manual).
- **max_position** (`int`) – The maximum position in DeviceUnits (see manual).

Raises

`ThorlabsError` – If not successful.

set_trigger_config_params(`mode1`, `polarity1`, `mode2`, `polarity2`)

Set the trigger configuration parameters.

Parameters

- **mode1** (`enums.KMOT_TriggerPortMode`) – The trigger 1 mode as a `KMOT_TriggerPortMode` enum value or member name.
- **polarity1** (`enums.KMOT_TriggerPortPolarity`) – The trigger 1 polarity as a `KMOT_TriggerPortPolarity` enum value or member name.
- **mode2** (`enums.KMOT_TriggerPortMode`) – The trigger 2 mode as a `KMOT_TriggerPortMode` enum value or member name.
- **polarity2** (`enums.KMOT_TriggerPortPolarity`) – The trigger 2 polarity as a `KMOT_TriggerPortPolarity` enum value or member name.

Raises

`ThorlabsError` – If not successful.

set_trigger_config_params_block(`trigger_config_params`)

Sets the trigger configuration parameters block.

Parameters

`trigger_config_params` (`structs.KMOT_TriggerConfig`) – Options for controlling the trigger configuration.

Raises

- `ThorlabsError` – If not successful.
- `TypeError` – If `trigger_config_params` is not a `structs.KMOT_TriggerConfig`.

set_trigger_params_params(`trigger_start_position_fwd`, `trigger_interval_fwd`,
`trigger_pulse_count_fwd`, `trigger_start_position_rev`,
`trigger_interval_rev`, `trigger_pulse_count_rev`,
`trigger_pulse_width`, `cycle_count`)

Set the Trigger Parameters parameters.

See `get_real_value_from_device_unit()` for converting from a DeviceUnit to a RealValue.

Parameters

- **trigger_start_position_fwd** (`int`) – The trigger start position, forward, in DeviceUnits (see manual).

- **trigger_interval_fwd** (`int`) – The trigger interval, forward, in DeviceUnits (see manual).
- **trigger_pulse_count_fwd** (`int`) – Number of trigger pulses, forward.
- **trigger_start_position_rev** (`int`) – The trigger start position, reverse, in DeviceUnits (see manual).
- **trigger_interval_rev** (`int`) – The trigger interval, reverse, in DeviceUnits (see manual).
- **trigger_pulse_count_rev** (`int`) – Number of trigger pulses., reverse.
- **trigger_pulse_width** (`int`) – Width of the trigger pulse in milliseconds, range 10 (10us) to 650000 (650ms).
- **cycle_count** (`int`) – Number of cycles to perform triggering.

Raises

`ThorlabsError` – If not successful.

set_trigger_params_params_block(trigger_params_params)

Set the Trigger Parameters parameters.

Parameters

`trigger_params_params` (`structs.KMOT_TriggerParams`) – Options for controlling the trigger.

Raises

- `ThorlabsError` – If not successful.
- `TypeError` – If `trigger_params_params` is not a `structs.KMOT_TriggerParams`.

set_vel_params(max_velocity, acceleration)

Sets the move velocity parameters.

See `get_device_unit_from_real_value()` for converting from a `RealValue` to a `DeviceUnit`.

Parameters

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

Raises

`ThorlabsError` – 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

`ThorlabsError` – 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

`ThorlabsError` – If not successful.

stop_immediate()

Stop the current move immediately (with the risk of losing track of the position).

Raises

`ThorlabsError` – If not successful.

stop_polling()

Stops the internal polling loop.

stop_profiled()

Stop the current move using the current velocity profile.

Raises

`ThorlabsError` – If not successful.

suspend_move_messages()

Suspend automatic messages at ends of moves.

Useful to speed up part of real-time system with lots of short moves.

Raises

`ThorlabsError` – If not successful.

time_since_last_msg_received()

Gets the time, in milliseconds, since the 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.
- `bool` – `True` if monitoring is enabled otherwise `False`.

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

`ThorlabsError` – 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: `MotionControl`

A wrapper around `Thorlabs.MotionControl.KCube.Solenoid.dll`.

The `properties` for a KCubeSolenoid connection supports the following key-value pairs in the *Connections Database*:

```
'device_name': str, the device name found in ThorlabsDefaultSettings.xml  
→[default: None]
```

Do not instantiate this class directly. Use the `connect()` method to connect to the equipment.

Parameters

`record (EquipmentRecord)` – A 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

`ThorlabsError` – If not successful.

get_cycle_params_block()

Get the cycle parameters.

Returns

`structs.SC_CycleParameters` – The cycle parameters.

Raises

`ThorlabsError` – 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

`ThorlabsError` – If not successful.

get_hardware_info_block()

Gets the hardware information in a block.

Returns

`structs.TLI_HardwareInformation` – The hardware information.

Raises

`ThorlabsError` – 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

`ThorlabsError` – If not successful.

`get_mmi_params_block()`

Gets the MMI parameters for the device.

Warning: This function is currently not in the DLL, as of v1.11.2, but it is in the header file.

Returns

`structs.KSC_MMIParams` – Options for controlling the MMI.

Raises

`ThorlabsError` – 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

`ThorlabsError` – 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

`ThorlabsError` – 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

`ThorlabsError` – 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

`ThorlabsError` – 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

`ThorlabsError` – If not successful.

load_named_settings(*settings_name*)

Update device with named settings.

Parameters

`settings_name (str)` – The name of the device to load the settings for.
Examples for the value of `setting_name` can be found in `ThorlabsDefaultSettings.xml`, which gets created when the Kinesis software is installed.

Raises

`ThorlabsError` – 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

`ThorlabsError` – If not successful.

persist_settings()

Persist the devices current settings.

Raises

`ThorlabsError` – 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 (MotionControlCallback)` – A function to be called whenever messages are received.

request_cycle_params()

Requests the cycle parameters.

Raises

ThorlabsError – If not successful.

request_digital_outputs()

Requests the digital output bits.

Raises

ThorlabsError – If not successful.

request_hub_bay()

Requests the hub bay number this device is fitted to.

Raises

ThorlabsError – If not successful.

request_led_switches()

Requests the LED indicator bits on the cube.

Raises

ThorlabsError – If not successful.

request_mmi_params()

Requests the MMI Parameters for the KCube Display Interface.

Raises

ThorlabsError – If not successful.

request_operating_mode()

Requests the operating mode.

Raises

ThorlabsError – If not successful.

request_operating_state()

Requests the operating state.

Raises

ThorlabsError – 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

ThorlabsError – 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

ThorlabsError – 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

`ThorlabsError` – If not successful.

request_trigger_config_params()

Requests the Trigger Configuration Parameters.

Raises

`ThorlabsError` – 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

`ThorlabsError` – If not successful.

set_cycle_params_block(*cycle_params*)

Sets the cycle parameters.

Parameters

cycle_params (`structs.SC_CycleParameters`) – The new cycle parameters.

Raises

`ThorlabsError` – 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

`ThorlabsError` – 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

`ThorlabsError` – 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

`ThorlabsError` – If not successful.

set_mmi_params_block(mmi_params)

Sets the MMI parameters for the device.

Warning: This function is currently not in the DLL, as of v1.11.2, but it is in the header file.

Parameters

`mmi_params` (`structs.KSC_MMIParams`) – Options for controlling the MMI.

Raises

`ThorlabsError` – 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

`ThorlabsError` – 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

`ThorlabsError` – 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

ThorlabsError – If not successful.

set_trigger_config_params(*mode1*, *polarity1*, *mode2*, *polarity2*)

Set the trigger configuration parameters.

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

ThorlabsError – 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

ThorlabsError – 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

ThorlabsError – If not successful.

stop_polling()

Stops the internal polling loop.

time_since_last_msg_received()

Gets the time, in milliseconds, since the 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

`ThorlabsError` – If not successful.

`msl.equipment.resources.thorlabs.kinesis.kcube stepper_motor` module

This module provides all the functionality required to control a KCube Stepper Motor (KST101).

`class msl.equipment.resources.thorlabs.kinesis.kcube stepper_motor.KCubeStepperMotor(record)`

Bases: `MotionControl`

A wrapper around `Thorlabs.MotionControl.KCube.StepperMotor.dll`.

The `properties` for a KCubeStepperMotor connection supports the following key-value pairs in the `Connections Database`:

```
'device_name': str, the device name found in ThorlabsDefaultSettings.xml
↳ [default: None]
```

Do not instantiate this class directly. Use the `connect()` method to connect to the equipment.

Parameters

`record (EquipmentRecord)` – A record from an `Equipment-Register Database`.

`can_device_lock_front_panel()`

Determine if the device front panel can be locked.

Returns

`bool` – `True` if the front panel of the device can be locked, `False` if not.

`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

ThorlabsError – 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

ThorlabsError – 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 `DeviceUnit` to a `RealValue`.

Returns

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

get_bow_index()

Gets the stepper motor bow index.

Returns

`int` – The bow index.

get_calibration_file()

Get calibration file for this motor.

Returns

`str` – The filename of the calibration file.

Raises

ThorlabsError – 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\(\)`](#), [`load_named_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** (`enums.UnitType`) – The unit of the real-world value.

Returns

`int` – The device value.

Raises

`ThorlabsError` – If not successful.

get_digital_outputs()

Gets the digital output bits.

Returns

`bytes` – Bit mask of states of the 4 digital output pins.

get_encoder_counter()

Get the encoder counter.

For devices that have an encoder, the current encoder position can be read.

Returns

`int` – The encoder count in encoder units.

get_front_panel_locked()

Query if the device front panel locked.

Returns

`bool` – `True` if the device front panel is locked, `False` if not.

get_hardware_info()

Gets the hardware information from the device.

Returns

`structs.TLI_HardwareInformation` – The hardware information.

Raises

`ThorlabsError` – If not successful.

get_hardware_info_block()

Gets the hardware information in a block.

Returns

`structs.TLI_HardwareInformation` – The hardware information.

Raises

`ThorlabsError` – If not successful.

get_homing_params_block()

Get the homing parameters.

Returns

`structs.MOT_HomingParameters` – The homing parameters.

Raises

`ThorlabsError` – If not successful.

get_homing_velocity()

Gets the homing velocity.

See [get_real_value_from_device_unit\(\)](#) for converting from a DeviceUnit to a RealValue.

Returns

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

get_hub_bay()

Gets the hub bay number this device is fitted to.

Returns

`bytes` – The number, 0x00 if unknown or 0xff if not on a hub.

get_jog_mode()

Gets the jog mode.

Returns

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

Raises

`ThorlabsError` – If not successful.

get_jog_params_block()

Get the jog parameters.

Returns

`structs.MOT_JogParameters` – The jog parameters.

Raises

`ThorlabsError` – 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 DeviceUnit to a RealValue.

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 DeviceUnit to a RealValue.

Returns

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

Raises

`ThorlabsError` – If not successful.

get_limit_switch_params()

Gets the limit switch parameters.

See [get_real_value_from_device_unit\(\)](#) for converting from a DeviceUnit to a RealValue.

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

`ThorlabsError` – If not successful.

get_limit_switch_params_block()

Get the limit switch parameters.

Returns

`structs.MOT_LimitSwitchParameters` – The limit switch parameters.

Raises

`ThorlabsError` – If not successful.

get_mmi_params()

Get the MMI Parameters for the KCube Display Interface.

Deprecated calls by [get_mmi_params_ext\(\)](#)

get_mmi_params_block()

Gets the MMI parameters for the device.

Returns

`structs.KMOT_MMIParams` – The MMI parameters for the device.

get_mmi_params_ext()

Get the MMI Parameters for the KCube Display Interface.

See [get_real_value_from_device_unit\(\)](#) for converting from a DeviceUnit to a RealValue.

Returns

- `enums.KMOT_WheelMode` – The device joystick mode.
- `int` – The joystick maximum velocity in DeviceUnits.
- `int` – The joystick acceleration in DeviceUnits.
- `enums.KMOT_WheeldirectionSense` – The joystick direction sense.
- `int` – The first preset position in DeviceUnits.

- `int` – The second preset position in DeviceUnits.
- `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).

Returns

`ThorlabsError` – 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

`ThorlabsError` – 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

`ThorlabsError` – 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

ThorlabsError – 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

ThorlabsError – If not successful.

get_move_absolute_position()

Gets the move absolute position.

See *get_real_value_from_device_unit()* for converting from a DeviceUnit to a RealValue.

Returns

int – The move 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 DeviceUnit to a RealValue.

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

ThorlabsError – If not successful.

get_number_positions()

Get the 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_pid_loop_encoder_coeff()

Gets the encoder PID loop coefficient.

This is the encoder coefficient. Use 0.0 to disable the encoder or if no encoder is present otherwise a positive encoder coefficient.

Returns

`float` – The encoder PID loop coefficient.

get_pid_loop_encoder_params()

Gets the Encoder PID loop parameters.

Returns

`structs.MOT_PIDLoopEncoderParams` – The Encoder PID loop parameters.

Raises

`ThorlabsError` – If not successful.

get_position()

Get the current position.

See `get_real_value_from_device_unit()` for converting from a `DeviceUnit` to a `RealValue`.

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 `DeviceUnit` to a `RealValue`.

Returns

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

get_power_params()

Gets the power parameters for the stepper motor.

Returns

`structs.MOT_PowerParameters` – The power parameters.

Raises

`ThorlabsError` – 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()`, `load_named_settings()` or `set_motor_params_ext()` must be called before calling this function, otherwise the returned value will always be 0.

Parameters

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

Returns

`float` – The real-world value.

Raises

`ThorlabsError` – 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 Stepper Motor maximum stage position.

See `get_real_value_from_device_unit()` for converting from a `DeviceUnit` to a `RealValue`.

Returns

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

get_stage_axis_min_pos()

Gets the Stepper Motor minimum stage position.

See `get_real_value_from_device_unit()` for converting from a `DeviceUnit` to a `RealValue`.

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 the polling functions, `start_polling()`.

Returns

`int` – The status bits from the device.

get_trigger_config_params()

Get the Trigger Configuration Parameters.

Returns

- `enums.KMOT_TriggerPortMode` – The trigger 1 mode.
- `enums.KMOT_TriggerPortPolarity` – The trigger 1 polarity.
- `enums.KMOT_TriggerPortMode` – The trigger 2 mode.

- `enums.KMOT_TriggerPortPolarity` – The trigger 2 polarity.

Raises

`ThorlabsError` – If not successful.

get_trigger_config_params_block()

Gets the trigger configuration parameters block.

Returns

`structs.KMOT_TriggerConfig` – Options for controlling the trigger configuration.

Raises

`ThorlabsError` – If not successful.

get_trigger_params_params()

Get the Trigger Parameters parameters.

See `get_real_value_from_device_unit()` for converting from a `DeviceUnit` to a `RealValue`.

Returns

- `int` – The trigger start position, forward, in `DeviceUnits` (see manual).
- `int` – The trigger interval, forward, in `DeviceUnits` (see manual).
- `int` – Number of trigger pulses, forward.
- `int` – The trigger start position, reverse, in `DeviceUnits` (see manual).
- `int` – The trigger interval, reverse, in `DeviceUnits` (see manual).
- `int` – Number of trigger pulses., reverse.
- `int` – Width of the trigger pulse in milliseconds, range 10 (10us) to 650000 (650ms).
- `int` – Number of cycles to perform triggering.

Raises

`ThorlabsError` – If not successful.

get_trigger_params_params_block()

Gets the trigger parameters block.

Returns

`structs.KMOT_TriggerParams` – Options for controlling the trigger.

Raises

`ThorlabsError` – If not successful.

get_vel_params()

Gets the move velocity parameters.

See `get_real_value_from_device_unit()` for converting from a `DeviceUnit` to a `RealValue`.

Returns

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

- **acceleration** (`int`) – The acceleration in DeviceUnits (see manual).

Raises

`ThorlabsError` – If not successful.

get_vel_params_block()

Get the move velocity parameters.

Returns

`structs.MOT_VelocityParameters` – The velocity parameters.

Raises

`ThorlabsError` – 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()

Home the device.

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

Raises

`ThorlabsError` – 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

`ThorlabsError` – If not successful.

load_named_settings(`settings_name`)

Update device with named settings.

Parameters

`settings_name` (`str`) – The name of the device to load the settings for. Examples for the value of `setting_name` can be found in `ThorlabsDefaultSettings.xml`, which gets created when the Kinesis software is installed.

Raises

`ThorlabsError` – 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

`ThorlabsError` – 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

`ThorlabsError` – 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

`ThorlabsError` – If not successful.

move_relative(displacement)

Move the motor by a relative amount.

See `get_device_unit_from_real_value()` for converting from a `RealValue` to a `DeviceUnit`.

Parameters

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

Raises

`ThorlabsError` – If not successful.

move_relative_distance()

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

Raises

`ThorlabsError` – If not successful.

move_to_position(index)

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 `RealValue` to a `DeviceUnit`.

Parameters

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

Raises

`ThorlabsError` – 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

`ThorlabsError` – If not successful.

persist_settings()

Persist the devices current settings.

Raises

`ThorlabsError` – 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` (`MotionControlCallback`) – A function to be called whenever messages are received.

request_backlash()

Requests the backlash.

Raises

`ThorlabsError` – If not successful.

request_bow_index()

Requests the stepper motor bow index.

Raises

`ThorlabsError` – If not successful.

request_digital_outputs()

Requests the digital output bits.

Raises

`ThorlabsError` – If not successful.

request_encoder_counter()

Requests the encoder counter.

Raises

ThorlabsError – If not successful.

request_front_panel_locked()

Ask the device if its front panel is locked.

Raises

ThorlabsError – If not successful.

request_homing_params()

Requests the homing parameters.

Raises

ThorlabsError – If not successful.

request_jog_params()

Requests the jog parameters.

Raises

ThorlabsError – If not successful.

request_limit_switch_params()

Requests the limit switch parameters.

Raises

ThorlabsError – If not successful.

request_mmi_params()

Requests the MMI Parameters for the KCube Display Interface.

Raises

ThorlabsError – If not successful.

request_move_absolute_position()

Requests the position of next absolute move.

Raises

ThorlabsError – If not successful.

request_move_relative_distance()

Requests the relative move distance.

Raises

ThorlabsError – If not successful.

request_pid_loop_encoder_params()

Requests the Encoder PID loop parameters.

Raises

ThorlabsError – If not successful.

request_pos_trigger_params()

Requests the position trigger parameters.

Raises

ThorlabsError – 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

ThorlabsError – If not successful.

request_power_params()

Requests the power parameters.

Raises

ThorlabsError – 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

ThorlabsError – 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

ThorlabsError – If not successful.

request_trigger_config_params()

Requests the Trigger Configuration Parameters.

Raises

ThorlabsError – If not successful.

request_vel_params()

Requests the velocity parameters.

Raises

ThorlabsError – If not successful.

reset_rotation_modes()

Reset the rotation modes for a rotational device.

Raises

ThorlabsError – If not successful.

resume_move_messages()

Resume suspended move messages.

Raises

ThorlabsError – 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 `RealValue` to a `DeviceUnit`.

Parameters

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

Raises

`ThorlabsError` – If not successful.

set_bow_index(*bow_index*)

Sets the stepper motor bow index.

Parameters

`bow_index` (`int`) – The bow index.

Raises

`ThorlabsError` – 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

`OSError` – If the `path` does not exist.

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

`ThorlabsError` – If not successful.

set_direction(*reverse*)

Sets the motor direction sense.

This function is used because some actuators 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.

Raises

`ThorlabsError` – If not successful.

set_encoder_counter(*count*)

Set the Encoder Counter values.

Setting the encoder counter to zero, effectively defines a home position on the encoder strip. Note, setting this value does not move the device.

Parameters

count (`int`) – The encoder count in encoder units.

Raises

`ThorlabsError` – If not successful.

set_front_panel_lock(*locked*)

Sets the device front panel lock state.

Parameters

locked (`bool`) – `True` to lock the device, `False` to unlock

Raises

`ThorlabsError` – If not successful.

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

`ThorlabsError` – If not successful.

set_homing_velocity(*velocity*)

Sets the homing velocity.

See `get_device_unit_from_real_value()` for converting from a `RealValue` to a `DeviceUnit`.

Parameters

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

Raises

`ThorlabsError` – 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

`ThorlabsError` – If not successful.

set_jog_params_block(jog_params)

Set the jog parameters.

Parameters

jog_params (*structs.MOT_JogParameters*) – The jog parameters.

Raises

- **ThorlabsError** – 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 RealValue to a DeviceUnit.

Parameters

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

Raises

ThorlabsError – 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 RealValue to a DeviceUnit.

Parameters

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

Raises

ThorlabsError – 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 RealValue to a DeviceUnit.

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

`ThorlabsError` – If not successful.

set_limit_switch_params_block(*params*)

Set the limit switch parameters.

Parameters

params (`structs.MOT_LimitSwitchParameters`) – The limit switch parameters.

Raises

`ThorlabsError` – 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_mmi_params(*joystick_mode*, *joystick_max_velocity*, *joystick_acceleration*, *direction_sense*, *preset_position1*, *preset_position2*, *display_intensity*)

Set the MMI Parameters for the KCube Display Interface.

Deprecated calls `set_mmi_params_ext()` setting the `display_timeout` to 1 minute and the `display_dim_intensity` to 8.

Raises

`ThorlabsError` – If not successful.

set_mmi_params_block(*mmi_params*)

Sets the MMI parameters for the device.

Parameters

mmi_params (`structs.KMOT_MMIPParams`) – Options for controlling the mmi.

Raises

`ThorlabsError` – If not successful.

set_mmi_params_ext(*joystick_mode*, *joystick_max_velocity*, *joystick_acceleration*, *direction_sense*, *preset_position1*, *preset_position2*, *display_intensity*, *display_timeout*, *display_dim_intensity*)

Set the MMI Parameters for the KCube Display Interface.

See `get_real_value_from_device_unit()` for converting from a `DeviceUnit` to a `RealValue`.

Parameters

- **joystick_mode** (`enums.KMOT_WheelMode`) – The device joystick mode as a `enums.KMOT_WheelMode` enum value or member name.
- **joystick_max_velocity** (`int`) – The joystick maximum velocity in DeviceUnits.
- **joystick_acceleration** (`int`) – The joystick acceleration in DeviceUnits.
- **direction_sense** (`enums.KMOT_WheelDirectionSense`) – The joystick direction sense as a `enums.KMOT_WheelDirectionSense` enum value or member name.
- **preset_position1** (`int`) – The first preset position in DeviceUnits.
- **preset_position2** (`int`) – The second preset position in DeviceUnits.
- **display_intensity** (`int`) – The display intensity, range 0 to 100%.
- **display_timeout** (`int`) – The display timeout, range 0 to 480 in minutes (0 is off, otherwise the inactivity period before dimming the display).
- **display_dim_intensity** (`int`) – The display dimmed intensity, range 0 to 10 (after the timeout period the device display will dim).

Raises

`ThorlabsError` – If not successful.

`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 DeviceUnit to a RealValue.

Parameters

- **steps_per_rev** (`float`) – The steps per revolution.
- **gear_box_ratio** (`float`) – The gear box ratio.
- **pitch** (`float`) – The pitch.

Raises

`ThorlabsError` – 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 DeviceUnit to a RealValue.

Parameters

- `steps_per_rev` (`float`) – The steps per revolution.
- `gear_box_ratio` (`float`) – The gear box ratio.
- `pitch` (`float`) – The pitch.

Raises

`ThorlabsError` – 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 DeviceUnit to a RealValue.

Parameters

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

Raises

`ThorlabsError` – 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

`ThorlabsError` – 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 DeviceUnit to a RealValue.

Parameters

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

Raises

`ThorlabsError` – If not successful.

set_move_absolute_position(*position*)

Sets the move absolute position.

See [get_device_unit_from_real_value\(\)](#) for converting from a RealValue to a DeviceUnit.

Parameters

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

Raises

ThorlabsError – If not successful.

set_move_relative_distance(*distance*)

Sets the move relative distance.

See [get_device_unit_from_real_value\(\)](#) for converting from a RealValue to a DeviceUnit.

Parameters

distance (`int`) – The relative position in DeviceUnits (see manual).

Raises

ThorlabsError – If not successful.

set_pid_loop_encoder_coeff(*coeff*)

Sets the encoder PID loop coefficient.

This is the encoder coefficient. Use 0.0 to disable the encoder or if no encoder is present otherwise a positive encoder coefficient.

Parameters

coeff (`float`) – The encoder PID loop coefficient.

Raises

ThorlabsError – If not successful.

set_pid_loop_encoder_params(*params*)

Sets the encoder PID loop parameters.

Parameters

params (`structs.MOT_PIDLoopEncoderParams`) – The encoder PID loop parameters.

Raises

- **ThorlabsError** – If not successful.
- **TypeError** – If *params* is not a `structs.MOT_PIDLoopEncoderParams`.

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 RealValue to a DeviceUnit.

Parameters

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

Raises

`ThorlabsError` – 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

`ThorlabsError` – If not successful.

set_rotation_modes(*mode, direction*)

Set the rotation modes for a rotational device.

Parameters

- **mode** (`enums.MOT_MovementModes`) – The travel mode as a `enums.MOT_MovementModes` enum value or member name.
- **direction** (`enums.MOT_MovementDirections`) – The travel mode as a `enums.MOT_MovementDirections` enum value or member name.

Raises

`ThorlabsError` – 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 `RealValue` to a `DeviceUnit`.

Parameters

- **min_position** (`int`) – The minimum position in DeviceUnits (see manual).
- **max_position** (`int`) – The maximum position in DeviceUnits (see manual).

Raises

`ThorlabsError` – If not successful.

set_trigger_config_params(*mode1, polarity1, mode2, polarity2*)

Set the trigger configuration parameters.

Parameters

- **mode1** (`enums.KMOT_TriggerPortMode`) – The trigger 1 mode as a `KMOT_TriggerPortMode` enum value or member name.
- **polarity1** (`enums.KMOT_TriggerPortPolarity`) – The trigger 1 polarity as a `KMOT_TriggerPortPolarity` enum value or member name.

- **mode2** (*enums.KMOT_TriggerPortMode*) – The trigger 2 mode as a *KMOT_TriggerPortMode* enum value or member name.
- **polarity2** (*enums.KMOT_TriggerPortPolarity*) – The trigger 2 polarity as a *KMOT_TriggerPortPolarity* enum value or member name.

Raises

ThorlabsError – If not successful.

set_trigger_config_params_block(*trigger_config_params*)

Sets the trigger configuration parameters block.

Parameters

trigger_config_params (*structs.KMOT_TriggerConfig*) – Options for controlling the trigger configuration.

Raises

- *ThorlabsError* – If not successful.
- **TypeError** – If *trigger_config_params* is not a *structs.KMOT_TriggerConfig*.

**set_trigger_params_params(*trigger_start_position_fwd*, *trigger_interval_fwd*,
trigger_pulse_count_fwd, *trigger_start_position_rev*,
trigger_interval_rev, *trigger_pulse_count_rev*,
trigger_pulse_width, *cycle_count*)**

Set the Trigger Parameters parameters.

See *get_real_value_from_device_unit()* for converting from a DeviceUnit to a RealValue.

Parameters

- **trigger_start_position_fwd** (*int*) – The trigger start position, forward, in DeviceUnits (see manual).
- **trigger_interval_fwd** (*int*) – The trigger interval, forward, in DeviceUnits (see manual).
- **trigger_pulse_count_fwd** (*int*) – Number of trigger pulses, forward.
- **trigger_start_position_rev** (*int*) – The trigger start position, reverse, in DeviceUnits (see manual).
- **trigger_interval_rev** (*int*) – The trigger interval, reverse, in DeviceUnits (see manual).
- **trigger_pulse_count_rev** (*int*) – Number of trigger pulses, reverse.
- **trigger_pulse_width** (*int*) – Width of the trigger pulse in milliseconds, range 10 (10us) to 650000 (650ms).
- **cycle_count** (*int*) – Number of cycles to perform triggering.

Raises

ThorlabsError – If not successful.

set_trigger_params_params_block(trigger_params_params)

Set the Trigger Parameters parameters.

Parameters

trigger_params_params (*structs.KMOT_TriggerParams*) – Options for controlling the trigger.

Raises

- **ThorlabsError** – If not successful.
- **TypeError** – If *trigger_params_params* is not a *structs.KMOT_TriggerParams*.

set_vel_params(max_velocity, acceleration)

Sets the move velocity parameters.

See [get_device_unit_from_real_value\(\)](#) for converting from a RealValue to a DeviceUnit.

Parameters

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

Raises

ThorlabsError – 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

ThorlabsError – 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

ThorlabsError – If not successful.

stop_immediate()

Stop the current move immediately (with the risk of losing track of the position).

Raises

ThorlabsError – If not successful.

stop_polling()

Stops the internal polling loop.

stop_profiled()

Stop the current move using the current velocity profile.

Raises

ThorlabsError – If not successful.

suspend_move_messages()

Suspend automatic messages at ends of moves.

Useful to speed up part of real-time system with lots of short moves.

Raises

ThorlabsError – If not successful.

time_since_last_msg_received()

Gets the time, in milliseconds, since the 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.
- `bool` – `True` if monitoring is enabled otherwise `False`.

uses_pid_loop_encoding()

Determines if we can uses PID loop encoding.

This is true if the stage supports PID Loop Encoding. Requires `get_pid_loop_encoder_coeff()` to have a positive non zero coefficient.

Returns

`bool` – Whether PID loop encoding is supported.

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

ThorlabsError – If not successful.

msl.equipment.resources.thorlabs.kinesis.messages module

Device Message Queue defined in Thorlabs Kinesis v1.14.18

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.

The message consists of 3 components, a messageType, a messageID and messageData:

```
WORD messageType
WORD messageID
WORD messageData
```

```
msl.equipment.resources.thorlabs.kinesis.messages.MessageTypes = {0:
'GenericDevice', 1: 'GenericPiezo', 2: 'GenericMotor', 3: 'GenericDCMotor',
4: 'GenericSimpleMotor', 5: 'RackDevice', 6: 'Laser', 7: 'TECCtrlr', 8:
'Quad', 9: 'NanoTrak', 10: 'Specialized', 11: 'Solenoid'}
```

MessageTypes

```
msl.equipment.resources.thorlabs.kinesis.messages.GenericDevice = {0:
'settingsInitialized', 1: 'settingsUpdated', 2: 'settingsExtern', 3:
'error', 4: 'close', 5: 'settingsReset'}
```

GenericDevice

```
msl.equipment.resources.thorlabs.kinesis.messages.GenericMotor = {0: 'Homed',
1: 'Moved', 2: 'Stopped', 3: 'LimitUpdated'}
```

GenericMotor

```
msl.equipment.resources.thorlabs.kinesis.messages.GenericDCMotor = {0:
'error', 1: 'status'}
```

GenericDCMotor

```
msl.equipment.resources.thorlabs.kinesis.messages.GenericPiezo = {0:
'maxVoltageChanged', 1: 'controlModeChanged', 2: 'statusChanged', 3:
'maxTravelChanged', 4: 'TSG_Status', 5: 'TSG_DisplayModeChanged'}
```

GenericPiezo

```
msl.equipment.resources.thorlabs.kinesis.messages.RackDevice = {0:
'RackCountEstablished', 1: 'RackBayState'}
```

RackDevice

```
msl.equipment.resources.thorlabs.kinesis.messages.Quad = {0: 'statusChanged'}
```

Quad

```
msl.equipment.resources.thorlabs.kinesis.messages.TECCtrlr = {0:
'statusChanged', 2: 'displaySettingsChanged', 3: 'feedbackParamsChanged'}
```

TECCtrlr

```
msl.equipment.resources.thorlabs.kinesis.messages.Laser = {0:
'statusChanged', 1: 'controlSourceChanged', 2: 'displayModeChanged'}
```

Laser

```
msl.equipment.resources.thorlabs.kinesis.messages.Solenoid = {0:  
    'statusChanged'}
```

Solenoid

```
msl.equipment.resources.thorlabs.kinesis.messages.NanoTrak = {0:  
    'statusChanged'}
```

NanoTrak

```
msl.equipment.resources.thorlabs.kinesis.messages.Specialized = {}
```

Specialized

```
msl.equipment.resources.thorlabs.kinesis.messages.GenericSimpleMotor = {}
```

GenericSimpleMotor

```
msl.equipment.resources.thorlabs.kinesis.messages.MessageID =  
{'GenericDCMotor': {0: 'error', 1: 'status'}, 'GenericDevice': {0:  
    'settingsInitialized', 1: 'settingsUpdated', 2: 'settingsExtern', 3:  
    'error', 4: 'close', 5: 'settingsReset'}, 'GenericMotor': {0: 'Homed', 1:  
    'Moved', 2: 'Stopped', 3: 'LimitUpdated'}, 'GenericPiezo': {0:  
    'maxVoltageChanged', 1: 'controlModeChanged', 2: 'statusChanged', 3:  
    'maxTravelChanged', 4: 'TSG_Status', 5: 'TSG_DisplayModeChanged'},  
    'GenericSimpleMotor': {}, 'Laser': {0: 'statusChanged', 1:  
    'controlSourceChanged', 2: 'displayModeChanged'}, 'NanoTrak': {0:  
    'statusChanged'}, 'Quad': {0: 'statusChanged'}, 'RackDevice': {0:  
    'RackCountEstablished', 1: 'RackBayState'}, 'Solenoid': {0:  
    'statusChanged'}, 'Specialized': {}, 'TECCtrlr': {0: 'statusChanged', 2:  
    'displaySettingsChanged', 3: 'feedbackParamsChanged'}}
```

MessageID

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_function,  
                           build_device_list=)
```

Bases: `ConnectionSDK`

Base **Thorlabs.MotionControl** class.

Do not instantiate this class directly. Use the `connect()` method to connect to the equipment.

Parameters

- **record** (*EquipmentRecord*) – A record from an *Equipment-Register Database*.
- **api_function** (*api_functions*) – An API function list from *api_functions* that the subclass is a wrapper around.
- **build_device_list** (*bool*, optional) – Whether to call *build_device_list()* before opening the connection to the device.

Raises

ThorlabsError – 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_Inertial_Motor = 97

KCube Inertial Motor device ID

KCube_LaserSource = 56

KCube Laser Source device ID

KCube_NanoTrak = 57

KCube NanoTrak device ID

KCube_Piezo = 29

KCube Piezo device ID

KCube_Solenoid = 68

KCube Solenoid device ID

KCube_Stripper_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_Qquad = 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

Vertical_Stage = 24

Vertical Stage 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.

property settings

The device settings specified in ThorlabsDefaultSettings.xml

If this is an empty dict then you can specify the device_name in the properties field in the *Connections Database* or you can run the Kinesis software and allow Kinesis to configure the actuator that is connected to the motor controller.

The possible values for device_name can be found in the ThorlabsDefaultSettings.xml file (located in the Kinesis installation folder, e.g., C:\Program Files\Thorlabs\Kinesis). as the Name value in one of the <DeviceSettingsType> tags.

Type
dict

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

ThorlabsError – 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

ThorlabsError – 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

`ThorlabsError` – 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 `dict`.

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

`dict` – The message represented as

`{ 'type': MessageTypes, 'id': MessageID, 'data': int }`

msl.equipment.resources.thorlabs.kinesis.structs module

Structs defined in Thorlabs Kinesis v1.14.10

class msl.equipment.resources.thorlabs.kinesis.structs.TLI_DeviceInfo

Bases: 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: 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: Structure

acceleration
    Structure/Union member

maxVelocity
    Structure/Union member

minVelocity
    Structure/Union member

class msl.equipment.resources.thorlabs.kinesis structs.MOT_JogParameters
    Bases: 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: 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_VelocityProfileParameters
    Bases: 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: 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: 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_BrushlessPositionLoopParameters

Bases: 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_BrushlessTrackSettleParameters**

Bases: 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_BrushlessCurrentLoopParameters**

Bases: 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_BrushlessElectricOutputParameters**

Bases: 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: 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: Structure

    movePercentage
        Structure/Union member

    restPercentage
        Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.MOT_DC_PIDParameters
    Bases: 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: 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: Structure

horizontalComponent
    Structure/Union member

verticalComponent
    Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.NT_CircleParameters
Bases: 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: Structure

LUTdiameter
    Structure/Union member

class msl.equipment.resources.thorlabs.kinesis structs.NT_TIARangeParameters
Bases: 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: 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: 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: 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: Structure

    controlMode
        Structure/Union member

    gain
        Structure/Union member

class
msl.equipment.resources.thorlabs.kinesis.structs.PZ_FeedbackLoopConstants
    Bases: Structure

    integralTerm
        Structure/Union member

    proportionalTerm
        Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.PZ_LUTWaveParameters
    Bases: 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: 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: Structure

filter1Fc
    Structure/Union member

filter1Q
    Structure/Union member

filter2Fc
    Structure/Union member

filter2Q
    Structure/Union member

filterNo
    Structure/Union member

notchFilter10n
    Structure/Union member

notchFilter20n
    Structure/Union member
```

```
class msl.equipment.resources.thorlabs.kinesis.structs.PPC_IOSettings
Bases: 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: 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: 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: 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: Structure

thresholdDeflection
    Structure/Union member

velocity
    Structure/Union member

class msl.equipment.resources.thorlabs.kinesis structs.MOT_PotentiometerSteps
Bases: Structure

potentiometerStepParameters
    Structure/Union member

class msl.equipment.resources.thorlabs.kinesis structs.KMOT_MMIParams
Bases: Structure
```

DisplayDimIntensity

Structure/Union member

DisplayIntensity

Structure/Union member

DisplayTimeout

Structure/Union member

PresetPos1

Structure/Union member

PresetPos2

Structure/Union member

WheelAcceleration

Structure/Union member

WheelDirectionSense

Structure/Union member

WheelMaxVelocity

Structure/Union member

WheelMode

Structure/Union member

reserved

Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.KMOT_TriggerConfig

Bases: 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: 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.KIM_DriveOPParameters

Bases: Structure

class msl.equipment.resources.thorlabs.kinesis structs.KIM_JogParameters

Bases: Structure

class

msl.equipment.resources.thorlabs.kinesis structs.KIM_LimitSwitchParameters

Bases: Structure

class msl.equipment.resources.thorlabs.kinesis structs.KIM_HomeParameters

Bases: Structure

class msl.equipment.resources.thorlabs.kinesis structs.KIM_MMIParameters

Bases: Structure

class

msl.equipment.resources.thorlabs.kinesis structs.KIM_MMIChannelParameters

Bases: Structure

class msl.equipment.resources.thorlabs.kinesis structs.KIM_TrigIOConfig

Bases: Structure

class

msl.equipment.resources.thorlabs.kinesis structs.KIM_TrigParamsParameters

Bases: Structure

class msl.equipment.resources.thorlabs.kinesis structs.KIM_FeedbackSigParams

Bases: Structure

class msl.equipment.resources.thorlabs.kinesis structs.KIM_Status

Bases: Structure

class msl.equipment.resources.thorlabs.kinesis structs.KLD_MMIParams

Bases: Structure

displayIntensity

Structure/Union member

reserved

Structure/Union member

class msl.equipment.resources.thorlabs.kinesis structs.**KLD_TrigIOParams**

Bases: Structure

mode1

Structure/Union member

mode2

Structure/Union member

polarity1

Structure/Union member

polarity2

Structure/Union member

reserved1

Structure/Union member

reserved2

Structure/Union member

class msl.equipment.resources.thorlabs.kinesis structs.**KLS_MMIParams**

Bases: Structure

displayIntensity

Structure/Union member

reserved

Structure/Union member

class msl.equipment.resources.thorlabs.kinesis structs.**KLS_TrigIOParams**

Bases: Structure

mode1

Structure/Union member

mode2

Structure/Union member

polarity1

Structure/Union member

polarity2

Structure/Union member

reserved1

Structure/Union member

reserved2

Structure/Union member

class msl.equipment.resources.thorlabs.kinesis structs.**KNA_TIARangeParameters**

Bases: 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.KNA_TIAResult
Bases: Structure

    absoluteReading
        Structure/Union member

    relativeReading
        Structure/Union member

    selectedRange
        Structure/Union member

    underOrOverRead
        Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.KNA_IOSettings
Bases: Structure

    highVoltageOutRange
        Structure/Union member

    highVoltageOutputRoute
        Structure/Union member

    lowVoltageOutRange
        Structure/Union member

    lowVoltageOutputRoute
        Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.KNA_MMIParams
Bases: Structure

    DisplayIntensity
        Structure/Union member

    WheelAdjustRate
        Structure/Union member
```

reserved
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis structs.KNA_TriggerConfig
Bases: Structure

Trigger1Mode
Structure/Union member

Trigger1Polarity
Structure/Union member

Trigger2Mode
Structure/Union member

Trigger2Polarity
Structure/Union member

reserved
Structure/Union member

unused1
Structure/Union member

unused2
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis structs.KNA_FeedbackLoopConstants
Bases: Structure

integralTerm
Structure/Union member

proportionalTerm
Structure/Union member

class msl.equipment.resources.thorlabs.kinesis structs.TPZ_IOSettings
Bases: Structure

class msl.equipment.resources.thorlabs.kinesis structs.KPZ_MMIParams
Bases: 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: 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: 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: Structure

    differentialGain
        Structure/Union member

    integralGain
        Structure/Union member

    proportionalGain
        Structure/Union member

class
msl.equipment.resources.thorlabs.kinesis.structs.QD_LowPassFilterParameters
    Bases: Structure

    lowPassFilterCutOffFreq
        Structure/Union member

    lowPassFilterEnabled
        Structure/Union member

class
msl.equipment.resources.thorlabs.kinesis.structs.QD_NotchFilterParameters
    Bases: Structure

    notchFilterCenterFrequency
        Structure/Union member

    notchFilterEnabled
        Structure/Union member

    notchFilterQ
        Structure/Union member

class
msl.equipment.resources.thorlabs.kinesis.structs.QD_PositionDemandParameters
    Bases: 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: Structure

    x
        Structure/Union member

    y
        Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.QD_Readings
Bases: Structure

    demandedPos
        Structure/Union member

    posDifference
        Structure/Union member

    sum
        Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.QD_KPA_TrigIOConfig
Bases: 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: Structure

    wDigOPs
        Structure/Union member

    wReserved
        Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.SC_CycleParameters
    Bases: Structure

    closedTime
        Structure/Union member

    numCycles
        Structure/Union member

    openTime
        Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.KSC_MMIParams
    Bases: 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: 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.TSG_IOSettings

Bases: 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.KSG_MMIParams

Bases: Structure

DisplayDimIntensity

Structure/Union member

DisplayIntensity

Structure/Union member

DisplayTimeout

Structure/Union member

reserved

Structure/Union member

class msl.equipment.resources.thorlabs.kinesis structs.KSG_TriggerConfig

Bases: Structure

LowerLimit

Structure/Union member

SmoothingSamples

Structure/Union member

Trigger1Mode

Structure/Union member

Trigger1Polarity

Structure/Union member

Trigger2Mode

Structure/Union member

Trigger2Polarity

Structure/Union member

UpperLimit

Structure/Union member

reserved

Structure/Union member

class msl.equipment.resources.thorlabs.kinesis.structs.TIM_DriveOPParameters

Bases: Structure

class msl.equipment.resources.thorlabs.kinesis.structs.TIM_JogParameters

Bases: Structure

class msl.equipment.resources.thorlabs.kinesis.structs.TIM_ButtonParameters

Bases: Structure

class msl.equipment.resources.thorlabs.kinesis.structs.TIM_Status

Bases: Structure

class msl.equipment.resources.thorlabs.kinesis.structs.TC_LoopParameters

Bases: Structure

differentialGain

Structure/Union member

integralGain

Structure/Union member

proportionalGain

Structure/Union member

Submodules**msl.equipment.resources.thorlabs.fwxx2c module**

Wrapper around Thorlabs FilterWheel102.dll, v4.0.0.

Thorlabs FW102C Series and FW212C Series Motorized Filter Wheels.

class msl.equipment.resources.thorlabs.fwxx2c.FilterCount(*value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None*)Bases: [IntEnum](#)

The number of filter positions that the filter wheel has.

SIX = 6

TWELVE = 12

```
class msl.equipment.resources.thorlabs.fwxx2c.SensorMode(value, names=None,
                                                       *values, module=None,
                                                       qualname=None,
                                                       type=None, start=1,
                                                       boundary=None)
```

Bases: [IntEnum](#)

Sensor modes of the filter wheel.

ON = 0

OFF = 1

```
class msl.equipment.resources.thorlabs.fwxx2c.SpeedMode(value, names=None, *values,
                                                       module=None,
                                                       qualname=None, type=None,
                                                       start=1, boundary=None)
```

Bases: [IntEnum](#)

Speed modes of the filter wheel.

SLOW = 0

FAST = 1

```
class msl.equipment.resources.thorlabs.fwxx2c.TriggerMode(value, names=None,
                                                       *values, module=None,
                                                       qualname=None,
                                                       type=None, start=1,
                                                       boundary=None)
```

Bases: [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.fwxx2c.FilterWheelXX2C(record)
```

Bases: [ConnectionSDK](#)

Wrapper around Thorlabs FilterWheel102.dll, v4.0.0.

Connects to the Thorlabs FW102C Series and FW212C Series Motorized Filter Wheels.

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 [properties](#) for a FilterWheelXX2C connection supports the following key-value pairs in the [Connections Database](#):

```
'port': str, the serial port number, e.g., 'COM3'
'baud_rate': int, the baud rate for the serial connection [default:  
→115200]
'timeout': int, the timeout in seconds [default: 10]
```

Do not instantiate this class directly. Use the `connect()` method to connect to the equipment.

Parameters

`record` (*EquipmentRecord*) – A record from an *Equipment-Register Database*.

Raises

`ThorlabsError` – 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` – 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 a 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.

msl.equipment.utils module

Common functions.

`msl.equipment.utils.convert_to_enum(obj, enum, prefix=None, to_upper=False, strict=True)`

Convert *obj* to an `Enum` member.

Parameters

- **obj** (`object`) – Any object to be converted to the specified *enum*. Can be a value of member of the specified *enum*.
- **enum** (`Type[Enum]`) – The `Enum` object that *obj* should be converted to.
- **prefix** (`str`, optional) – If *obj* is a `str`, then ensures that *prefix* is included at the beginning of *obj* before converting *obj* to the *enum*.
- **to_upper** (`bool`, optional) – If *obj* is a `str`, then whether to change *obj* to be upper case before converting *obj* to the *enum*.
- **strict** (`bool`, optional) – Whether errors should be raised. If `False` and *obj* cannot be converted to *enum* then *obj* is returned and the error is logged.

Returns

`Enum` – The *enum* member.

Raises

`ValueError` – If *obj* is not in *enum* and *strict* is `True`.

`msl.equipment.utils.convert_to_primitive(text)`

Convert text into a primitive value.

Parameters

`text` (`str` or `bytes`) – The text to convert.

Returns

- The *text* as a `None`, `bool`, `int`,
- `float` or `complex` object. Returns the
- original *text* if it cannot be converted to any of these types.
- The *text* 0 and 1 get converted to an integer not a boolean.

`msl.equipment.utils.convert_to_date(obj, fmt='%Y-%m-%d', strict=True)`

Convert an object to a `datetime.date` object.

Parameters

- **obj** (`datetime.date`, `datetime.datetime` or `str`) – Any object that can be converted to a `datetime.date` object.
- **fmt** (`str`) – If *obj* is a `str` then the format to use to convert *obj* to a `datetime.date`.

- **strict** (`bool`, optional) – Whether errors should be raised. If `False` and `obj` cannot be converted to `datetime.date` then `datetime.date(datetime.MINYEAR, 1, 1)` is returned and the error is logged.

Returns

`datetime.date` – A `datetime.date` object.

```
msl.equipment.utils.convert_to_xml_string(element, indent=' ', encoding='utf-8',
                                         fix_newlines=True)
```

Convert an XML `Element` in to a string with proper indentation.

Parameters

- **element** (`Element`) – The element to convert.
- **indent** (`str`, optional) – The value to use for the indentation.
- **encoding** (`str`, optional) – The encoding to use.
- **fix_newlines** (`bool`, optional) – Whether to remove newlines inside text nodes.

Returns

`str` – The `element` as a pretty string. The returned value can be directly written to a file (i.e., it includes the XML declaration).

Examples

If the `Element` contains unicode characters then you should use the `codecs` module to create the file if you are using Python 2.7:

```
import codecs
with codecs.open('my_file.xml', mode='w', encoding='utf-8') as fp:
    fp.write(convert_to_xml_string(element))
```

otherwise you can use the builtin `open()` function:

```
with open('my_file.xml', mode='w', encoding='utf-8') as fp:
    fp.write(convert_to_xml_string(element))
```

```
msl.equipment.utils.xml_element(tag, text=None, tail=None, **attributes)
```

Create a new XML element.

Parameters

- **tag** (`str`) – The element's name.
- **text** (`str`, optional) – The text before the first sub-element. Can either be a string or `None`.
- **tail** (`str`, optional) – The text after this element's end tag, but before the next sibling element's start tag.
- **attributes** – All additional key-value pairs are included as XML attributes for the element. The value must be of type `str`.

Returns

`Element` – The new XML element.

`msl.equipment.utils.xml_comment(text)`

Create a new XML comment element.

Parameters

`text (str)` – The comment.

Returns

`Comment()` – A special element that is an XML comment.

`msl.equipment.utils.to_bytes(iterable, fmt='ieee', dtype='<f'`)

Convert an iterable of numbers into bytes.

Parameters

- **iterable** – An object to convert to bytes. Must be a 1-dimensional sequence of elements (not a multidimensional array).
- **fmt (str or None, optional)** – The format to use to convert *iterable*. Possible values are:
 - '' (empty string or `None`) – convert *iterable* to bytes without a header.

None: <byte><byte><byte>...

– 'ascii' – comma-separated ASCII characters, see the *<PROGRAM DATA SEPARATOR>* standard that is defined in Section 7.4.2.2, [IEEE 488.2-1992](#).

ascii: <string>,<string>,<string>,...

– 'ieee' – arbitrary block data for *SCPI* messages, see the *<DEFINITE LENGTH ARBITRARY BLOCK RESPONSE DATA>* standard that is defined in Section 8.7.9, [IEEE 488.2-1992](#).

ieee: #<length of num bytes value><num bytes><byte><byte><byte>...

– 'hp' – the HP-IB data transfer standard, i.e., the *FORM#* command option. See the programming guide for an [HP 8530A](#) for more details.

hp: #A<num bytes as uint16><byte><byte><byte>...

- **dtype** – The data type to use to convert each element in *iterable* to. If *fmt* is 'ascii' then *dtype* must be of type `str` and it is used as the *format_spec* argument in `format()` to first convert each element in *iterable* to a string, and then it is encoded (e.g., '.2e' converts each element to scientific notation with two digits after the decimal point). If *dtype* includes a byte-order character, it is ignored. For all other values of *fmt*, the *dtype* can be any object that `numpy.dtype` supports (e.g., 'H', 'uint16' and `numpy.ushort` are equivalent values to convert each element to an *unsigned short*). If a byte-order character is specified then it is used, otherwise the native byte order of the CPU architecture is used. See [Format Strings](#) for more details.

Returns

`bytes` – The *iterable* converted to bytes.

```
msl.equipment.utils.from_bytes(buffer, fmt='ieee', dtype='<f')
```

Convert bytes into an array.

Parameters

- **buffer** (`bytes`, `bytearray` or `str`) – A byte buffer. Can be an already-decoded buffer of type `str`, but only if `fmt` equals 'ascii'.
- **fmt** (`str` or `None`, optional) – The format that `buffer` is in. See `to_bytes()` for more details.
- **dtype** – The data type of each element in `buffer`. Can be any object that `numpy.dtype` supports. See `to_bytes()` for more details.

Returns

`numpy.ndarray` – The array.

```
msl.equipment.utils.ipv4_addresses() → set[str]
```

Get all IPv4 addresses on all network interfaces.

```
msl.equipment.utils.parse_lxi_webserver(host, port=80, timeout=1)
```

Get the information about an LXI device from the device's webserver.

Parameters

- **host** (`str`) – The IP address or hostname of the LXI device.
- **port** (`int`, optional) – The port number of the device's webservice.
- **timeout** (`float`, optional) – The maximum number of seconds to wait for a reply.

Returns

`dict` – The information about the LXI device.

msl.equipment.vxi11 module

Implementation of the `VXI-11` protocol.

`VXI-11` is a client-service model to send messages through a network. The messages are formatted using the Remote Procedure Call protocol [RFC-1057] and are encoded/decoded using the eXternal Data Representation standard [RFC-1014].

References

- `VXI-11 – TCP/IP Instrument Protocol Specification (Revision 1.0)`, **VXIbus Consortium**, July 1995.
- `RFC-1057 – RPC: Remote Procedure Call Protocol Specification (Version 2)`, **Sun Microsystems**, June 1988.
- `RFC-1014 – XDR: External Data Representation Standard`, **Sun Microsystems**, June 1987.

```
class msl.equipment.vxi11.OperationFlag(value, names=None, *values, module=None,
                                         qualname=None, type=None, start=1,
                                         boundary=None)
```

Bases: [IntEnum](#)

VXI-11: Additional information concerning how a request is carried out.

NULL = 0

WAITLOCK = 1

END = 8

TERMCHRSET = 128

```
class msl.equipment.vxi11.MessageType(value, names=None, *values, module=None,
                                         qualname=None, type=None, start=1,
                                         boundary=None)
```

Bases: [IntEnum](#)

RPC: The message type.

CALL = 0

REPLY = 1

```
class msl.equipment.vxi11.ReplyStatus(value, names=None, *values, module=None,
                                         qualname=None, type=None, start=1,
                                         boundary=None)
```

Bases: [IntEnum](#)

RPC: Message reply status.

MSG_ACCEPTED = 0

MSG_DENIED = 1

```
class msl.equipment.vxi11.AcceptStatus(value, names=None, *values, module=None,
                                         qualname=None, type=None, start=1,
                                         boundary=None)
```

Bases: [IntEnum](#)

RPC: Message accepted status.

SUCCESS = 0

PROG_UNAVAIL = 1

PROG_MISMATCH = 2

PROC_UNAVAIL = 3

GARBAGE_ARGS = 4

```
class msl.equipment.vxi11.RejectStatus(value, names=None, *values, module=None,
                                         qualname=None, type=None, start=1,
                                         boundary=None)
```

Bases: [IntEnum](#)

RPC: Message rejected status.

RPC_MISMATCH = 0

AUTH_ERROR = 1

```
class msl.equipment.vxi11.AuthStatus(value, names=None, *values, module=None,
                                       qualname=None, type=None, start=1,
                                       boundary=None)
```

Bases: `IntEnum`

RPC: Authorization status.

AUTH_BADCRED = 1

AUTH_REJECTEDCRED = 2

AUTH_BADVERF = 3

AUTH_REJECTEDVERF = 4

AUTH_TOOWEAK = 5

```
class msl.equipment.vxi11.RPCClient(host)
```

Bases: `object`

Remote Procedure Call implementation for a client.

Parameters

host (`str`) – The hostname or IP address of the remote device.

append(*data*)

Append data to the body of the current RPC message.

Parameters

data (`bytes` or `memoryview`) – The data to append.

append_opaque(*text*)

Append a variable-length string to the body of the current RPC message.

Parameters

text (`memoryview`, `bytes` or `str`) – The data to append.

property chunk_size

The maximum number of bytes to receive at a time from the socket.

Type

`int`

close()

Close the RPC socket, if one is open.

connect(*port*, *timeout*=10)

Connect to a specific port on the device.

Parameters

- **port** (`int`) – The port number to connect to.
- **timeout** (`float` or `None`, optional) – The maximum number of seconds to wait for the connection to be established.

get_buffer()

Get the data in the buffer.

Returns

`bytearray` – The data in the current RPC message.

get_port(*prog, vers, prot, timeout=10*)

Call the Port Mapper procedure to determine which port to use for a program.

This method will automatically open and close the socket connection.

Parameters

- ***prog* (`int`)** – The program number to get the port number of.
- ***vers* (`int`)** – The version number of *prog*.
- ***prot* (`int`)** – The socket protocol family type to use when sending requests to *prog* (IPPROTO_TCP or IPPROTO_UDP).
- ***timeout* (`float` or `None`, optional)** – The maximum number of seconds to wait to get the port value.

Returns

`int` – The port number that corresponds to *prog*.

init(*prog, vers, proc*)

Construct a new RPC message.

Parameters

- ***prog* (`int`)** – The program number.
- ***vers* (`int`)** – The version number of program.
- ***proc* (`int`)** – The procedure number within the program to be called.

interrupt_handler()

Override this method to be notified of a service interrupt.

This method gets called if an interrupt is received during a `read()`. It does not continuously poll the device.

read()

Read an RPC message, check for errors, and return the procedure-specific data.

Returns

`memoryview` – The procedure-specific data.

set_timeout(*timeout*)

Set the socket timeout value.

Parameters

- ***timeout* (`float`)** – The timeout, in seconds, to use for the socket.

property socket

The reference to the socket.

Type

`socket`

static unpack_opaque(data)

Unpack and return a variable-length string.

Parameters

data (`bytes`, `bytearray` or `memoryview`) – The data to unpack.

Returns

`bytes`, `bytearray` or `memoryview` – The unpacked data.

write()

Write the RPC message that is in the buffer.

check_reply(message)

Checks the message for errors and returns the procedure-specific data.

Parameters

message (`memoryview`) – The reply from an RPC message.

Returns

`memoryview` or `None` – The reply or `None` if the transaction id does not match the value that was used in the corresponding `write()` call.

class msl.equipment.vxi11.VXIClient(host)

Bases: `RPCClient`

Base class for a VXI-11 program.

Parameters

host (`str`) – The hostname or IP address of the remote device.

read_reply()

Check the RPC message for an error and return the remaining data.

Returns

`memoryview` – The reply data.

class msl.equipment.vxi11.CoreClient(host)

Bases: `VXIClient`

Communicate with the *Device Core* program on the remote device.

Parameters

host (`str`) – The hostname or IP address of the remote device.

create_link(device, lock_device, lock_timeout)

Create a link.

Parameters

- **device** (`bytes` or `str`) – Name of the device to link with.
- **lock_device** (`bool`) – Whether to attempt to lock the device.
- **lock_timeout** (`int`) – Time, in milliseconds, to wait on a lock.

Returns

- `int` – The link ID.
- `int` – The port number of the *Device Async* program (see `AsyncClient`).

- **int** – The maximum data size the device will accept on a [device_write\(\)](#).

device_write(*lid, io_timeout, lock_timeout, flags, data*)

Write data to the specified device.

Parameters

- **lid (int)** – Link id from [create_link\(\)](#).
- **io_timeout (int)** – Time, in milliseconds, to wait for I/O to complete.
- **lock_timeout (int)** – Time, in milliseconds, to wait on a lock.
- **flags (int or OperationFlag)** – Operation flags to use.
- **data (memoryview, bytes or str)** – The data to write.

Returns

int – The number of bytes written.

device_read(*lid, request_size, io_timeout, lock_timeout, flags, term_char*)

Read data from the device.

Parameters

- **lid (int)** – Link id from [create_link\(\)](#).
- **request_size (int)** – The number of bytes requested.
- **io_timeout (int)** – Time, in milliseconds, to wait for I/O to complete.
- **lock_timeout (int)** – Time, in milliseconds, to wait on a lock.
- **flags (int or OperationFlag)** – Operation flags to use.
- **term_char (int)** – The termination character. Valid only if *flags* is [TERMCHRSET](#).

Returns

- **int** – The reason(s) the read completed.
- **memoryview** – A view of the data (the RPC header is removed).

device_readstb(*lid, flags, lock_timeout, io_timeout*)

Read the status byte from the device.

Parameters

- **lid (int)** – Link id from [create_link\(\)](#).
- **flags (int or OperationFlag)** – Operation flags to use.
- **lock_timeout (int)** – Time, in milliseconds, to wait on a lock.
- **io_timeout (int)** – Time, in milliseconds, to wait for I/O to complete.

Returns

int – The status byte.

device_trigger(*lid, flags, lock_timeout, io_timeout*)

Send a trigger to the device.

Parameters

- **lid** (`int`) – Link id from `create_link()`.
- **flags** (`int` or `OperationFlag`) – Operation flags to use.
- **lock_timeout** (`int`) – Time, in milliseconds, to wait on a lock.
- **io_timeout** (`int`) – Time, in milliseconds, to wait for I/O to complete.

device_clear(*lid, flags, lock_timeout, io_timeout*)

Send the *clear* command to the device.

Parameters

- **lid** (`int`) – Link id from `create_link()`.
- **flags** (`int` or `OperationFlag`) – Operation flags to use.
- **lock_timeout** (`int`) – Time, in milliseconds, to wait on a lock.
- **io_timeout** (`int`) – Time, in milliseconds, to wait for I/O to complete.

device_remote(*lid, flags, lock_timeout, io_timeout*)

Place the device in a remote state wherein all programmable local controls are disabled.

Parameters

- **lid** (`int`) – Link id from `create_link()`.
- **flags** (`int` or `OperationFlag`) – Operation flags to use.
- **lock_timeout** (`int`) – Time, in milliseconds, to wait on a lock.
- **io_timeout** (`int`) – Time, in milliseconds, to wait for I/O to complete.

device_local(*lid, flags, lock_timeout, io_timeout*)

Place the device in a local state wherein all programmable local controls are enabled.

Parameters

- **lid** (`int`) – Link id from `create_link()`.
- **flags** (`int` or `OperationFlag`) – Operation flags to use.
- **lock_timeout** (`int`) – Time, in milliseconds, to wait on a lock.
- **io_timeout** (`int`) – Time, in milliseconds, to wait for I/O to complete.

device_lock(*lid, flags, lock_timeout*)

Acquire a device's lock.

Parameters

- **lid** (`int`) – Link id from `create_link()`.
- **flags** (`int` or `OperationFlag`) – Operation flags to use.
- **lock_timeout** (`int`) – Time, in milliseconds, to wait on a lock.

device_unlock(lid)

Release a lock acquired by [device_lock\(\)](#).

Parameters

• **lid** (`int`) – Link id from [create_link\(\)](#).

device_enable_srq(lid, enable, handle)

Enable or disable the sending of *device_intr_srq* RPCs by the network instrument server.

Parameters

- **lid** (`int`) – Link id from [create_link\(\)](#).

- **enable** (`bool`) – Whether to enable or disable interrupts.

- **handle** (`bytes`) – Host specific data (maximum length is 40 characters).

device_docmd(lid, flags, io_timeout, lock_timeout, cmd, network_order, datasize, data_in)

Allows for a variety of operations to be executed.

Parameters

- **lid** (`int`) – Link id from [create_link\(\)](#).

- **flags** (`int` or *OperationFlag*) – Operation flags to use.

- **io_timeout** (`int`) – Time, in milliseconds, to wait for I/O to complete.

- **lock_timeout** (`int`) – Time, in milliseconds, to wait on a lock.

- **cmd** (`int`) – Which command to execute.

- **network_order** (`bool`) – Client's byte order.

- **datasize** (`int`) – Size of individual data elements.

- **data_in** (`bytes` or `str`) – Data input parameters.

Returns

`bytes` – The results defined by *cmd*.

destroy_link(lid)

Destroy the link.

Parameters

• **lid** (`int`) – Link id from [create_link\(\)](#).

create_intr_chan(host_addr, host_port, prog_num, prog_vers, prog_family)

Inform the network instrument server to establish an interrupt channel.

Parameters

- **host_addr** (`int`) – Host servicing the interrupt.

- **host_port** (`int`) – Valid port number on the client.

- **prog_num** (`int`) – Program number.

- **prog_vers** (`int`) – Program version number.

- **prog_family** (`int`) – The underlying socket protocol family type (IPPROTO_TCP or IPPROTO_UDP).

destroy_intr_chan()

Inform the network instrument server to close its interrupt channel.

class msl.equipment.vxi11.AsyncClient(host)

Bases: [VXIClient](#)

Communicate with the *Device Async* program on the remote device.

Parameters

host ([str](#)) – The hostname or IP address of the remote device.

device_abort(lid)

Stops an in-progress call.

Parameters

lid ([int](#)) – Link id from [create_link\(\)](#).

msl.equipment.vxi11.find_vxi11(*, ip: list[str] | None = None, timeout: float = 1) → dict[str, dict[str, str | list[str]]]

Find all VXI-11 devices that are on the network.

The RPC port-mapper protocol ([RFC-1057](#), Appendix A) broadcasts a message via UDP to port 111 for VXI-11 device discovery.

Parameters

- **ip** – The IP address(es) on the local computer to use to broadcast the discovery message. If not specified, broadcast on all network interfaces.
- **timeout** – The maximum number of seconds to wait for a reply.

Returns

The information about the VXI-11 devices that were found.

1.7 License

MIT License

Copyright (c) 2017 - 2023, Measurement Standards Laboratory of New Zealand

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

(continues on next page)

(continued from previous page)

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.

1.8 Developers

- Joseph Borbely <joseph.borbely@measurement.govt.nz>

1.9 Release Notes

1.9.1 Version 0.2.0 (in development)

- Added
 - support for Python 3.12
 - `find-equipment` console script
 - `ConnectionGPIB` class
 - `GMH3000` resource
- Fixed
 - issue #9 - Missing functions from Avantes AvaSpec DLL
 - issue #8 - Invalid URL for LXI XML identification document
- Removed
 - support for Python 2.7, 3.5, 3.6 and 3.7

1.9.2 Version 0.1.0 (2023-06-18)

Initial release.

It is also the last release to support Python 2.7, 3.5, 3.6 and 3.7

**CHAPTER
TWO**

INDEX

- modindex

PYTHON MODULE INDEX

M

msl.equipment, 17
msl.equipment.config, 17
msl.equipment.connection, 19
msl.equipment.connection_demo, 21
msl.equipment.connection_gpib, 22
msl.equipment.connection_message_based,
 29
msl.equipment.connection_nidaq, 31
msl.equipment.connection_prologix, 33
msl.equipment.connection_pyvisa, 37
msl.equipment.connection_sdk, 38
msl.equipment.connection_serial, 39
msl.equipment.connection_socket, 41
msl.equipment.connection_tcpip_hislip,
 42
msl.equipment.connection_tcpip_vxi11,
 45
msl.equipment.connection_zeromq, 48
msl.equipment.constants, 50
msl.equipment.database, 51
msl.equipment.dns_service_discovery, 54
msl.equipment.exceptions, 54
msl.equipment.factory, 56
msl.equipment.hislip, 57
msl.equipment.resources, 87
msl.equipment.resources.aim_tti, 90
msl.equipment.resources.aim_tti.mx_series,
 90
msl.equipment.resources.avantes, 97
msl.equipment.resources.avantes.avaspec,
 97
msl.equipment.resources.bentham, 127
msl.equipment.resources.bentham.benhw32,
 128
msl.equipment.resources.bentham.benhw64,
 130
msl.equipment.resources.bentham.errors,
 131
msl.equipment.resources.bentham.tokens,
 131
msl.equipment.resources.cmi, 131
msl.equipment.resources.cmi.sia3, 131
msl.equipment.resources.dataray, 133
msl.equipment.resources.dataray.datarayocx_32,
 133
msl.equipment.resources.dataray.datarayocx_64,
 133
msl.equipment.resources.electron_dynamics,
 136
msl.equipment.resources.electron_dynamics.tc_series
 136
msl.equipment.resources.energetiq, 141
msl.equipment.resources.energetiq.eq99,
 141
msl.equipment.resources.greisinger, 147
msl.equipment.resources.greisinger.gmh3000,
 147
msl.equipment.resources.mks_instruments,
 151
msl.equipment.resources.mks_instruments.pr4000b,
 151
msl.equipment.resources.nkt, 162
msl.equipment.resources.nkt.nktpdll,
 162
msl.equipment.resources.omega, 195
msl.equipment.resources.omega.ithx, 195
msl.equipment.resources.optosigma, 199
msl.equipment.resources.optosigma.shot702,
 200
msl.equipment.resources.optronic_laboratories,
 206
msl.equipment.resources.optronic_laboratories.ol756
 206
msl.equipment.resources.optronic_laboratories.ol756
 224
msl.equipment.resources.optronic_laboratories.ol_cu
 225
msl.equipment.resources.picotech, 228
msl.equipment.resources.picotech.errors,
 228
msl.equipment.resources.picotech.picoscope,

```
    229                               msl.equipment.resources.thorlabs.kinesis.api_funci
msl.equipment.resources.picotech.picoscope.callbacks,
    229                               msl.equipment.resources.thorlabs.kinesis.benchtop_s
msl.equipment.resources.picotech.picoscope.channel,
    230                               msl.equipment.resources.thorlabs.kinesis.callbacks,
msl.equipment.resources.picotech.picoscope.enums,
    231                               msl.equipment.resources.thorlabs.kinesis.enums,
msl.equipment.resources.picotech.picoscope.functions,
    232                               msl.equipment.resources.thorlabs.kinesis.errors,
msl.equipment.resources.picotech.picoscope.help,
    233                               msl.equipment.resources.thorlabs.kinesis.filter_fli
msl.equipment.resources.picotech.picoscope.picope,
    235                               msl.equipment.resources.thorlabs.kinesis.integrated
msl.equipment.resources.picotech.picope_2k3k,
    239                               msl.equipment.resources.thorlabs.kinesis.kcube_dc_s
msl.equipment.resources.picotech.picope_api,
    231                               msl.equipment.resources.thorlabs.kinesis.kcube_sole
msl.equipment.resources.picotech.picope_ps2000,
    232                               msl.equipment.resources.thorlabs.kinesis.kcube_step
msl.equipment.resources.picotech.picope_ps2000a,
    234                               msl.equipment.resources.thorlabs.kinesis.messages,
msl.equipment.resources.picotech.picope_ps3000,
    235                               msl.equipment.resources.thorlabs.kinesis.motion_con
msl.equipment.resources.picotech.picope_ps3000a,
    237                               msl.equipment.resources.thorlabs.kinesis structs,
msl.equipment.resources.picotech.picope_ps4000,
    239                               msl.equipment.resources.utils, 88
msl.equipment.resources.picotech.picope_ps4000a,
    231                               msl.equipment.vxi11, 598
msl.equipment.resources.picotech.picope_ps5000,
    233                               msl.equipment.vxi11, 601
msl.equipment.resources.picotech.picope_ps5000a,
    234                               msl.equipment.vxi11, 601
msl.equipment.resources.picotech.picope_ps6000,
    236                               msl.equipment.vxi11, 601
msl.equipment.resources.picotech.picope_structs,
    238                               msl.equipment.vxi11, 601
msl.equipment.resources.picotech.pt104,
    230                               msl.equipment.vxi11, 601
msl.equipment.resources.princeton_instruments,
    234                               msl.equipment.vxi11, 601
msl.equipment.resources.princeton_instruments.arc_instrument,
    234                               msl.equipment.vxi11, 601
msl.equipment.resources.raicol, 401
msl.equipment.resources.raicol.raicol_tec,
    401                               msl.equipment.vxi11, 601
msl.equipment.resources.thorlabs, 402
msl.equipment.resources.thorlabs.fwxx2c,
    593                               msl.equipment.vxi11, 601
msl.equipment.resources.thorlabs.kinesis,
    402                               msl.equipment.vxi11, 601
```

INDEX

A

A (msl.equipment.resources.picotech.picoscope.enums.PS1000AChannelAttribute), 240	A_MIN (msl.equipment.resources.picotech.picoscope.enums.PS5000AChannelAttribute), 306
A (msl.equipment.resources.picotech.picoscope.enums.PS2000AChannelAttribute), 297	A_MIN (msl.equipment.resources.picotech.picoscope.enums.PS6000AChannelAttribute), 297
A (msl.equipment.resources.picotech.picoscope.enums.PS3000AChannelAttribute), 315	abort () (msl.equipment.connection_tcpip_vxi11.ConnectionTCPILMethod), 47
A (msl.equipment.resources.picotech.picoscope.enums.PS3000AChannelAttribute), 251	ABOVE (msl.equipment.resources.picotech.picoscope.enums.PS2000AChannelAttribute), 47
A (msl.equipment.resources.picotech.picoscope.enums.PS4000AChannelAttribute), 248	ABOVE (msl.equipment.resources.picotech.picoscope.enums.PS2000AChannelAttribute), 248
A (msl.equipment.resources.picotech.picoscope.enums.PS4000AChannelAttribute), 281	ABOVE (msl.equipment.resources.picotech.picoscope.enums.PS2000AChannelAttribute), 281
A (msl.equipment.resources.picotech.picoscope.enums.PS4000AChannelAttribute), 238	ABOVE (msl.equipment.resources.picotech.picoscope.enums.PS3000AChannelAttribute), 238
A (msl.equipment.resources.picotech.picoscope.enums.PS5000AChannelAttribute), 266	ABOVE (msl.equipment.resources.picotech.picoscope.enums.PS3000AChannelAttribute), 266
A (msl.equipment.resources.picotech.picoscope.enums.PS5000AChannelAttribute), 296	ABOVE (msl.equipment.resources.picotech.picoscope.enums.PS4000AChannelAttribute), 296
A (msl.equipment.resources.picotech.picoscope.enums.PS6000AChannelAttribute), 289	ABOVE (msl.equipment.resources.picotech.picoscope.enums.PS4000AChannelAttribute), 289
A_MAX (msl.equipment.resources.picotech.picoscope.enums.PS2000AChannelBufferIndexAttribute), 277	ABOVE (msl.equipment.resources.picotech.picoscope.enums.PS5000AChannelBufferIndexAttribute), 277
A_MAX (msl.equipment.resources.picotech.picoscope.enums.PS3000AChannelBufferIndexAttribute), 311	ABOVE (msl.equipment.resources.picotech.picoscope.enums.PS5000AChannelBufferIndexAttribute), 311
A_MAX (msl.equipment.resources.picotech.picoscope.enums.PS4000AChannelBufferIndexAttribute), 301	ABOVE (msl.equipment.resources.picotech.picoscope.enums.PS6000AChannelBufferIndexAttribute), 301
A_MAX (msl.equipment.resources.picotech.picoscope.enums.PS4000AChannelBufferIndexAttribute), 321	ABOVE_LOWER (msl.equipment.resources.picotech.picoscope.enums.PS6000AChannelBufferIndexAttribute), 321
A_MAX (msl.equipment.resources.picotech.picoscope.enums.PS5000AChannelBufferIndexAttribute), 249	ABOVE_LOWER (msl.equipment.resources.picotech.picoscope.enums.PS7000AChannelBufferIndexAttribute), 249
A_MAX (msl.equipment.resources.picotech.picoscope.enums.PS5000AChannelBufferIndexAttribute), 306	ABOVE_LOWER (msl.equipment.resources.picotech.picoscope.enums.PS7000AChannelBufferIndexAttribute), 306
A_MAX (msl.equipment.resources.picotech.picoscope.enums.PS5000AChannelBufferIndexAttribute), 297	ABOVE_LOWER (msl.equipment.resources.picotech.picoscope.enums.PS7000AChannelBufferIndexAttribute), 297
A_MAX (msl.equipment.resources.picotech.picoscope.enums.PS6000AChannelBufferIndexAttribute), 289	ABOVE_LOWER (msl.equipment.resources.picotech.picoscope.enums.PS8000AChannelBufferIndexAttribute), 289
A_MAX (msl.equipment.resources.picotech.picoscope.enums.PS7000AChannelBufferIndexAttribute), 315	ABOVE_LOWER (msl.equipment.resources.picotech.picoscope.enums.PS8000AChannelBufferIndexAttribute), 315
A_MIN (msl.equipment.resources.picotech.picoscope.enums.PS2000AChannelBufferIndexAttribute), 277	ABOVE_LOWER (msl.equipment.resources.picotech.picoscope.enums.PS9000AChannelBufferIndexAttribute), 277
A_MIN (msl.equipment.resources.picotech.picoscope.enums.PS3000AChannelBufferIndexAttribute), 311	ABOVE_LOWER (msl.equipment.resources.picotech.picoscope.enums.PS9000AChannelBufferIndexAttribute), 311
A_MIN (msl.equipment.resources.picotech.picoscope.enums.PS4000AChannelBufferIndexAttribute), 321	absoluteReading (msl.equipment.resources.picotech.picoscope.enums.PS4000AChannelBufferIndexAttribute), 321
A_MIN (msl.equipment.resources.picotech.picoscope.enums.PS4000AChannelBufferIndexAttribute), 269	attribute (msl.equipment.resources.picotech.picoscope.enums.PS4000AChannelBufferIndexAttribute), 269
A_MIN (msl.equipment.resources.picotech.picoscope.enums.PS4000AChannelBufferIndexAttribute), 269	thorlabs.kinesis.structs.KNA_TIAmountmentBufferIndexAttribute), 586

AllowAllMoves (<i>msl.equipment.resources.thorlabs.kinesis.enums.MOTerminateITILimitsSoftwareApproachPolicy</i> attribute), 436	<i>async_lock_info()</i>
AllowPartialMoves (<i>msl.equipment.resources.thorlabs.kinesis.enums.MOTerminateITISoftwareApproachPolicy</i> attribute), 436	<i>(msl.equipment.hislip.AsyncClient async_lock_release())</i>
amplifierCurrentLimit (<i>msl.equipment.resources.thorlabs.kinesis structs.BNTheHoldSettings</i> attribute), 577	<i>(msl.equipment.hislip.AsyncClient async_lock_request())</i>
amplifierLowPassFilter (<i>msl.equipment.resources.thorlabs.kinesis structs.BNTheHoldSettings</i> attribute), 577	<i>(msl.equipment.hislip.AsyncClient async_maximum_message_size())</i>
ANALOGUE_HARDWARE_VERSION (<i>msl.equipment.resources.picotech.picoscope.enums.PicotechSdPInfoApi</i> attribute), 233	<i>(msl.equipment.hislip.AsyncClient async_remote_local_control())</i>
AnalogueCh1 (<i>msl.equipment.resources.thorlabs.kinesis.enums.msl.HardwareMode</i> attribute), 468	<i>(msl.equipment.hislip.AsyncClient method), 77</i>
AnalogueCh2 (<i>msl.equipment.resources.thorlabs.kin</i> attribute), 468	<i>asynchronousHardwareModes (msl.equipment.hislip.AsyncClient</i>
AND (<i>msl.equipment.resources.picotech.picoscope.enums.PS2000ATridge</i> attribute), 241	<i>Operand async_status_query())</i>
anticlockwiseHardwareLimit (<i>msl.equipment.resources.thorlabs.kinesis.structs.MOTerminateITISwitchParameters</i> attribute), 576	<i>(msl.equipment.hislip.AsyncClient AsyncClient (class in msl.equipment.hislip), 76</i>
anticlockwisePosition (<i>msl.equipment.resources.thorlabs.kinesis.structs.msl.DigitalSwitchParameters</i> attribute), 576	<i>AsyncClient (class in msl.equipment.vxi11), 609 in msl.equipment.hislip), 65</i>
append() (<i>msl.equipment.vxi11.RPCClient</i>	<i>AsyncDeviceClear (msl.equipment.hislip.MessageType</i>
append_opaque() (<i>msl.equipment.vxi11.RPCClient</i> method), 603	<i>attribute), 57</i>
apply_resistance_scaling() (<i>msl.equipment.resources.picotech.picoscope.ps4000anBlockScope4000A</i> method), 352	<i>AsyncDeviceClearAcknowledge (class in msl.equipment.hislip), 65</i>
ARM (<i>msl.equipment.resources.picotech.picoscope.enums.msl.PS5000AStrigWithInputPiggment</i> attribute), 312	<i>(msl.equipment.hislip.MessageType attribute), 58</i>
ARRAY_OF_CHANNELS (<i>msl.equipment.resources.picotech.picoscope.enums.msl.PS5000AStrigWithInputPiggment</i> attribute), 293	<i>AsyncEndTLS (msl.equipment.hislip.MessageType attribute), 58</i>
ask() (<i>msl.equipment.connection_gpib.Connection</i> method), 23	<i>AsyncEndTLSResponse (msl.equipment.hislip.MessageType</i>
assembly (<i>msl.equipment.connection_sdk.ConnectionSDK</i> property), 38	<i>attribute), 58</i>
async_device_clear() (<i>msl.equipment.hislip.AsyncClient</i> method), 78	<i>asynchronous (msl.equipment.connection_tcpip_hislip.Connection property), 43</i>
async_end_tls() (<i>msl.equipment.hislip.AsyncClient</i> method), 78	<i>AsyncInitialize (class in msl.equipment.hislip), 67</i>
async_initialize() (<i>msl.equipment.hislip.AsyncClient</i>	<i>AsyncInitialize (msl.equipment.hislip.MessageType attribute), 57</i>
	<i>AsyncInitializeResponse (class in msl.equipment.hislip), 68</i>

AsyncInitializeResponse (<i>msl.equipment.hislip.MessageType</i> <i>attribute</i>), 57		<i>msl.equipment.hislip)</i> , 69
AsyncInterrupted (<i>msl.equipment.hislip.MessageType</i> <i>attribute</i>), 57		AsyncStartTLSResponse (<i>msl.equipment.hislip.MessageType</i> <i>attribute</i>), 58
AsyncLock (<i>class</i> in <i>msl.equipment.hislip</i>), 62		AsyncStatusQuery (<i>class</i> in <i>msl.equipment.hislip</i>), 68
AsyncLock (<i>msl.equipment.hislip.MessageType</i> <i>attribute</i>), 57		AsyncStatusQuery (<i>msl.equipment.hislip.MessageType</i> <i>attribute</i>), 57
AsyncLockInfo (<i>class</i> in <i>msl.equipment.hislip</i>), 63		AsyncStatusResponse (<i>class</i> in <i>msl.equipment.hislip</i>), 68
AsyncLockInfo (<i>msl.equipment.hislip.MessageType</i> <i>attribute</i>), 58		AsyncStatusResponse (<i>msl.equipment.hislip.MessageType</i> <i>attribute</i>), 57
AsyncLockInfoResponse (<i>class</i> in <i>msl.equipment.hislip</i>), 64		attrib() (<i>msl.equipment.config.Config</i> method), 18
AsyncLockInfoResponse (<i>msl.equipment.hislip.MessageType</i> <i>attribute</i>), 58		AUTH_BADCRED (<i>msl.equipment.vxi11.AuthStatus</i> <i>attribute</i>), 603
AsyncLockResponse (<i>class</i> in <i>msl.equipment.hislip</i>), 63		AUTH_BADVERF (<i>msl.equipment.vxi11.AuthStatus</i> <i>attribute</i>), 603
AsyncLockResponse (<i>msl.equipment.hislip.MessageType</i> <i>attribute</i>), 57		AUTH_ERROR (<i>msl.equipment.vxi11.RejectStatus</i> <i>attribute</i>), 603
AsyncMaximumMessageSize (<i>class</i> in <i>msl.equipment.hislip</i>), 66		AUTH_REJECTEDCRED (<i>msl.equipment.vxi11.AuthStatus</i> at- tribute), 603
AsyncMaximumMessageSize (<i>msl.equipment.hislip.MessageType</i> <i>attribute</i>), 57		AUTH_REJECTEDVERF (<i>msl.equipment.vxi11.AuthStatus</i> at- tribute), 603
AsyncMaximumMessageSizeResponse (<i>class</i> in <i>msl.equipment.hislip</i>), 66		AUTH_TOOWEAK (<i>msl.equipment.vxi11.AuthStatus</i> <i>attribute</i>), 603
AsyncMaximumMessageSizeResponse (<i>msl.equipment.hislip.MessageType</i> <i>attribute</i>), 57		AUTHENTICATION_FAILED (<i>msl.equipment.hislip.ErrorType</i> at- tribute), 58
AsyncRemoteLocalControl (<i>class</i> in <i>msl.equipment.hislip</i>), 64		authentication_result() (<i>msl.equipment.hislip.SyncClient</i> <i>method</i>), 76
AsyncRemoteLocalControl (<i>msl.equipment.hislip.MessageType</i> <i>attribute</i>), 57		authentication_start() (<i>msl.equipment.hislip.SyncClient</i> <i>method</i>), 76
AsyncRemoteLocalResponse (<i>class</i> in <i>msl.equipment.hislip</i>), 64		AuthenticationExchange (<i>class</i> in <i>msl.equipment.hislip</i>), 72
AsyncRemoteLocalResponse (<i>msl.equipment.hislip.MessageType</i> <i>attribute</i>), 57		AuthenticationExchange (<i>msl.equipment.hislip.MessageType</i> <i>attribute</i>), 58
AsyncServiceRequest (<i>msl.equipment.hislip.MessageType</i> <i>attribute</i>), 57		AuthenticationResult (<i>class</i> in <i>msl.equipment.hislip</i>), 73
AsyncStartTLS (<i>class</i> in <i>msl.equipment.hislip</i>), 69		AuthenticationResult (<i>msl.equipment.hislip.MessageType</i> <i>attribute</i>), 58
AsyncStartTLS (<i>msl.equipment.hislip.MessageType</i> <i>attribute</i>), 58		AuthenticationStart (<i>class</i> in <i>msl.equipment.hislip</i>), 72
AsyncStartTLSResponse (<i>class</i> in		

AuthenticationStart (msl.equipment.hislip.MessageType attribute), 58
AuthStatus (class in msl.equipment.vxi11), 603
auto_measure() (msl.equipment.resources.bentham.benhw32BANKIN(msl.equipment.resources.picotech.picoscope.enums.PS2000PwqConditions attribute), 370
attribute), 128
auto_measure() (msl.equipment.resources.bentham.benhw64.BenthamAttribute attribute), 265
method), 130
auto_range() (msl.equipment.resources.bentham.benhw32.BenthamAttribute attribute), 288
method), 128
auto_zero() (msl.equipment.resources.mks_instruments.pr4000BDR4000B
method), 151
AUX (msl.equipment.resources.picotech.picoscope.enums.PS2000AChannel attribute), 241
AUX (msl.equipment.resources.picotech.picoscope.enums.PS3000AChannel attribute), 259
AUX (msl.equipment.resources.picotech.picoscope.enums.PS4000AChannel attribute), 281
AUX (msl.equipment.resources.picotech.picoscope.enums.PS4000CDeviceType attribute), 270
AUX (msl.equipment.resources.picotech.picoscope.enums.PS5000ACIdentityType attribute), 306
AUX (msl.equipment.resources.picotech.picoscope.enums.PS5000Channel attribute), 296
AUX (msl.equipment.resources.picotech.picoscope.enums.PS6000CDeviceType attribute), 315
aux (msl.equipment.resources.picotech.picoscope structs.PS2000APtrigSettingsType attribute), 359
aux (msl.equipment.resources.picotech.picoscope structs.PS2000ATriggerConditions attribute), 359
aux (msl.equipment.resources.picotech.picoscope structs.PS3000ATriggerConditions attribute), 362
aux (msl.equipment.resources.picotech.picoscope structs.PS3000ATriggerConditionsV2 attribute), 363
aux (msl.equipment.resources.picotech.picoscope structs.PS3000ATriggerConditions attribute), 361
aux (msl.equipment.resources.picotech.picoscope structs.PS3000ATriggerConditions attribute), 362
aux (msl.equipment.resources.picotech.picoscope structs.PS3000ATriggerConditions attribute), 364
aux (msl.equipment.resources.picotech.picoscope structs.PS4000TriggerConditions attribute), 364
aux (msl.equipment.resources.picotech.picoscope structs.PS5000ATriggerConditions attribute), 368
aux (msl.equipment.resources.picotech.picoscope structs.PS5000ATriggerConditions attribute), 368
aux (msl.equipment.resources.picotech.picoscope structs.PS5000PwqConditions attribute), 367
attribute), 366 aux (msl.equipment.resources.picotech.picoscope structs.PS6000PwqConditions attribute), 370
attribute), 370 aux (msl.equipment.resources.picotech.picoscope structs.PS6000TrigConditions attribute), 369
AUX_IN (msl.equipment.resources.picotech.picoscope.enums.PS3000AChannel attribute), 247
AUX_IN (msl.equipment.resources.picotech.picoscope.enums.PS4000AChannel attribute), 288
AUX_IN (msl.equipment.resources.picotech.picoscope.enums.PS4000CDeviceType attribute), 270
AUX_IN (msl.equipment.resources.picotech.picoscope.enums.PS5000ACIdentityType attribute), 306
AUX_IN (msl.equipment.resources.picotech.picoscope.enums.PS5000Channel attribute), 296
AUX_IN (msl.equipment.resources.picotech.picoscope.enums.PS6000CDeviceType attribute), 315
Avantes (class in msl.equipment.resources.avantes.avaspec), 106
Avantes.BroadcastAnswerType (class in msl.equipment.resources.avantes.avaspec), 110
Avantes.DarkCorrectionType (class in msl.equipment.resources.avantes.avaspec), 110
Avantes.DeviceConfigType (class in msl.equipment.resources.avantes.avaspec), 110
Avantes.DeviceConfigTypeV2 (class in msl.equipment.resources.avantes.avaspec), 110
Avantes.DynamicStorageType (class in msl.equipment.resources.avantes.avaspec), 113
Avantes.DynamicStorageType (class in msl.equipment.resources.avantes.avaspec), 113
Avantes.HeartbeatRespType (class in msl.equipment.resources.avantes.avaspec), 113
Avantes.HeartbeatRespType (class in msl.equipment.resources.avantes.avaspec), 113

115
Avantes.InterfaceType (class in AVERAGE (msl.equipment.resources.picotech.picoscope.enums.PS3000AChar msl.equipment.resources.avantes.avaspec), attribute), 268
108 AVERAGE (msl.equipment.resources.picotech.picoscope.enums.PS4000AChar attribute), 278
Avantes.IrradianceType (class in attribute), 292
msl.equipment.resources.avantes.avaspec), AVERAGE (msl.equipment.resources.picotech.picoscope.enums.PS4000AChar attribute), 278
112
Avantes.MeasConfigType (class in AVERAGE (msl.equipment.resources.picotech.picoscope.enums.PS5000AChar msl.equipment.resources.avantes.avaspec), attribute), 313
112 AVERAGE (msl.equipment.resources.picotech.picoscope.enums.PS5000AChar attribute), 313
Avantes.OemDataType (class in attribute), 303
msl.equipment.resources.avantes.avaspec), AVERAGE (msl.equipment.resources.picotech.picoscope.enums.PS6000AChar attribute), 322
115
Avantes.ProcessControlType (class in AVS_SERIAL_LEN
msl.equipment.resources.avantes.avaspec), (msl.equipment.resources.avantes.avaspec.Avantes attribute), 106
114
Avantes.SensType (class in AvsIdentityType (class in
msl.equipment.resources.avantes.avaspec), msl.equipment.resources.avantes.avaspec), 99
108
Avantes.SmoothingType (class in AWG_DAC_FREQUENCY
msl.equipment.resources.avantes.avaspec), (msl.equipment.resources.picotech.picoscope.ps4000a.Pic attribute), 352
111
Avantes.SpectrumCalibrationType (class in AWG_DAC_FREQUENCY
msl.equipment.resources.avantes.avaspec), (msl.equipment.resources.picotech.picoscope.ps5000a.Pic attribute), 355
111
Avantes.SpectrumCorrectionType (class in AWG_PHASE_ACCUMULATOR
msl.equipment.resources.avantes.avaspec), (msl.equipment.resources.picotech.picoscope.ps4000a.Pic attribute), 352
112
Avantes.StandAloneType (class in AWG_PHASE_ACCUMULATOR
msl.equipment.resources.avantes.avaspec), (msl.equipment.resources.picotech.picoscope.ps5000a.Pic attribute), 355
113
Avantes.TecControlType (class in axisID (msl.equipment.resources.thorlabs.kinesis.structs.MOT_Sta msl.equipment.resources.avantes.avaspec), attribute), 573
114
Avantes.TempSensorType (class in B
msl.equipment.resources.avantes.avaspec), B (msl.equipment.resources.picotech.picoscope.enums.PS2000AChar attribute), 240
114
Avantes.TimeStampType (class in B (msl.equipment.resources.picotech.picoscope.enums.PS2000Char msl.equipment.resources.avantes.avaspec), attribute), 233
113 B (msl.equipment.resources.picotech.picoscope.enums.PS3000AChar attribute), 233
Avantes.TriggerType (class in attribute), 258
msl.equipment.resources.avantes.avaspec), B (msl.equipment.resources.picotech.picoscope.enums.PS3000Char attribute), 252
112
AvantesError, 54 B (msl.equipment.resources.picotech.picoscope.enums.PS4000AChar attribute), 281
AVASPEC_ERROR_MSG_LEN (msl.equipment.resources.avantes.avaspec.AVASPEC_ERROR_MSG_LEN (msl.equipment.resources.picotech.picoscope.enums.PS4000Char attribute), 270
attribute), 106
AVASPEC_MIN_MSG_LEN B (msl.equipment.resources.picotech.picoscope.enums.PS5000AChar (msl.equipment.resources.avantes.avaspec.AVASPEC_MIN_MSG_LEN (msl.equipment.resources.picotech.picoscope.enums.PS5000Char attribute), 305
attribute), 106 B (msl.equipment.resources.picotech.picoscope.enums.PS5000Char attribute), 305
AVERAGE (msl.equipment.resources.picotech.picoscope.enums.PS2000AChar attribute), 206

B (msl.equipment.resources.picotech.picoscope.enums.BATCH_AND_SERIAL attribute), 315 (msl.equipment.resources.picotech.picoscope.enums.PS3000AChannelBufferIndex baud_rate (msl.equipment.connection_serial.ConnectionSerial attribute), 240 BELOW (msl.equipment.resources.picotech.picoscope.enums.PS2000AChannelBufferIndex attribute), 254 baud_rate (msl.equipment.connection_serial.ConnectionSerial attribute), 258 BELOW (msl.equipment.resources.picotech.picoscope.enums.PS3000AChannelBufferIndex attribute), 254 baud_rate (msl.equipment.connection_serial.ConnectionSerial attribute), 258 BELOW (msl.equipment.resources.picotech.picoscope.enums.PS2000AChannelBufferIndex attribute), 282 baud_rate (msl.equipment.connection_serial.ConnectionSerial attribute), 282 BELOW (msl.equipment.resources.picotech.picoscope.enums.PS3000AChannelBufferIndex attribute), 282 baud_rate (msl.equipment.connection_serial.ConnectionSerial attribute), 282 BELOW (msl.equipment.resources.picotech.picoscope.enums.PS3000AChannelBufferIndex attribute), 282 baud_rate (msl.equipment.connection_serial.ConnectionSerial attribute), 282 BELOW (msl.equipment.resources.picotech.picoscope.enums.PS3000AChannelBufferIndex attribute), 306 baud_rate (msl.equipment.connection_serial.ConnectionSerial attribute), 306 BELOW (msl.equipment.resources.picotech.picoscope.enums.PS5000AChannelBufferIndex attribute), 297 baud_rate (msl.equipment.connection_serial.ConnectionSerial attribute), 297 BELOW (msl.equipment.resources.picotech.picoscope.enums.PS4000AChannelBufferIndex attribute), 315 baud_rate (msl.equipment.connection_serial.ConnectionSerial attribute), 315 BELOW (msl.equipment.resources.picotech.picoscope.enums.PS5000AChannelBufferIndex attribute), 240 baud_rate (msl.equipment.connection_serial.ConnectionSerial attribute), 240 BELOW (msl.equipment.resources.picotech.picoscope.enums.PS5000AChannelBufferIndex attribute), 258 baud_rate (msl.equipment.connection_serial.ConnectionSerial attribute), 258 BELOW (msl.equipment.resources.picotech.picoscope.enums.PS6000AChannelBufferIndex attribute), 282 baud_rate (msl.equipment.connection_serial.ConnectionSerial attribute), 282 BELOW (msl.equipment.resources.picotech.picoscope.enums.PS4000AChannelBufferIndex attribute), 269 baud_rate (msl.equipment.connection_serial.ConnectionSerial attribute), 269 BELOW LOWER (msl.equipment.resources.picotech.picoscope.enums.PS5000AChannelBufferIndex attribute), 277 baud_rate (msl.equipment.connection_serial.ConnectionSerial attribute), 277 BELOW LOWER (msl.equipment.resources.picotech.picoscope.enums.PS5000AChannelBufferIndex attribute), 306 baud_rate (msl.equipment.connection_serial.ConnectionSerial attribute), 306 BELOW LOWER (msl.equipment.resources.picotech.picoscope.enums.PS5000AChannelBufferIndex attribute), 297 baud_rate (msl.equipment.connection_serial.ConnectionSerial attribute), 297 BELOW LOWER (msl.equipment.resources.picotech.picoscope.enums.PS6000AChannelBufferIndex attribute), 315 baud_rate (msl.equipment.connection_serial.ConnectionSerial attribute), 315 BELOW LOWER (msl.equipment.resources.picotech.picoscope.enums.PS6000AChannelBufferIndex attribute), 277 Backend (class in msl.equipment.constants), 50 attribute), 277 backend (msl.equipment.record_types.ConnectionRecord attribute), 85 BELOW LOWER (msl.equipment.resources.picotech.picoscope.enums.PS5000AChannelBufferIndex attribute), 311 BAD_CONTROL_CODE (msl.equipment.hislip.ErrorType attribute), 58 at- BELOW LOWER (msl.equipment.resources.picotech.picoscope.enums.PS5000AChannelBufferIndex attribute), 321 Benchtop_Brushless_Motor (msl.equipment.hislip.ErrorType attribute), 58 at- Benchtop_NanoTrak (msl.equipment.hislip.ErrorType attribute), 58 at- (msl.equipment.resources.thorlabs.kinesis.motion_control.S567ScopeChannel attribute), 567 BAD_HEADER (msl.equipment.hislip.ErrorType attribute), 58 at- Benchtop_Piezo_1_Channel (msl.equipment.resources.thorlabs.kinesis.motion_control.S567ScopeChannel attribute), 567 BAD_MESSAGE_TYPE (msl.equipment.hislip.ErrorType attribute), 58 at- Benchtop_Piezo_3_Channel (msl.equipment.resources.thorlabs.kinesis.motion_control.S567ScopeChannel attribute), 567 BAD_VENDOR (msl.equipment.hislip.ErrorType attribute), 58 at- Benchtop_Piezo_1_Channel (msl.equipment.resources.thorlabs.kinesis.motion_control.S567ScopeChannel attribute), 567 bandwidth (msl.equipment.resources.picotech.picoscope.channel_property), 231 Benchtop_Piezo_3_Channel (msl.equipment.resources.thorlabs.kinesis.motion_control.S567ScopeChannel attribute), 567 BATCH_AND_SERIAL (msl.equipment.resources.picotech.picoscope.enums.BatchAndSerialInfoApi attribute), 233 Benchtop_Stripper_Motor_1_Channel (msl.equipment.resources.thorlabs.kinesis.motion_control.S567ScopeChannel attribute), 567 BATCH_AND_SERIAL (msl.equipment.resources.picotech.picoscope.enums.BatchAndSerialInfoApi attribute), 235 Benchtop_Stripper_Motor_3_Channel (msl.equipment.resources.thorlabs.kinesis.motion_control.S567ScopeChannel attribute), 567

(*msl.equipment.resources.thorlabs.kinesis.motion_co*~~ntralMotionControl~~*ntrolMotionControl**resources.bentham.benhw64.Bentham*
attribute), 567
method), 130

BenchtopStepperMotor (class in *busy* (*msl.equipment.hislip.AsyncEndTLSResponse*
msl.equipment.resources.thorlabs.kinesis.benchtop_st~~epper~~*otor*),
402
busy (*msl.equipment.hislip.AsyncStartTLSResponse*

Bentham (class *in* *property*), 70
msl.equipment.resources.bentham.benhw64buttonMode (*msl.equipment.resources.thorlabs.kinesis.structs.MO*
130
attribute), 582

Bentham32 (class *in* *BW_20KHZ* (*msl.equipment.resources.picotech.picoscope.enums.PS4*
msl.equipment.resources.bentham.benhw32),
128
attribute), 280
BW_20MHZ (*msl.equipment.resources.picotech.picoscope.enums.PS3*
attribute), 257

BenthamError, 54

BlockReady (in *module* *BW_20MHZ* (*msl.equipment.resources.picotech.picoscope.enums.PSS*
msl.equipment.resources.picotech.picoscope.callback),
229
attribute), 305
BW_20MHZ (*msl.equipment.resources.picotech.picoscope.enums.PS6*
attribute), 314

BNCtriggerOrLowVoltageOut (*msl.equipment.resources.thorlabs.kinesis.bn*~~C~~*NC*~~T~~*BN*~~T~~*LowVoltageOut**resources.picotech.picoscope.enums.PS6*
attribute), 577
attribute), 314

BNT_BNCTriggerModes (class *in* *BW_FULL* (*msl.equipment.resources.picotech.picoscope.enums.PS30*
msl.equipment.resources.thorlabs.kinesis.enums),
445
attribute), 257
BW_FULL (*msl.equipment.resources.picotech.picoscope.enums.PS40*
attribute), 314

BNT_CurrentLimit (class *in* *attribute*), 280
msl.equipment.resources.thorlabs.kinesis.en~~BN~~*CurrentLimit* (*msl.equipment.resources.picotech.picoscope.enums.PS50*
444
attribute), 305

BNT_FeedbackSignalSelection (class *in* *BW_FULL* (*msl.equipment.resources.picotech.picoscope.enums.PS60*
msl.equipment.resources.thorlabs.kinesis.enums),
445
attribute), 314
byte_buffer (*msl.equipment.connection_socket.ConnectionSocket*

BNT_IO_Settings (class *in* *property*), 41
msl.equipment.resources.thorlabs.kinesis.st~~BYT~~*byte_buffer* (*msl.equipment.connection_tcpip_vxi11.Connection*
577
property), 46

BNT_OutputLowPassFilter (class *in* *C*
msl.equipment.resources.thorlabs.kinesis.enums),
444
C (*msl.equipment.resources.picotech.picoscope.enums.PS2000ACh*
attribute), 281

board (*msl.equipment.connection_gpib.ConnectionGPIB* *attribute*), 240
property), 23
C (*msl.equipment.resources.picotech.picoscope.enums.PS2000Char*
attribute), 223

Bright (*msl.equipment.resources.thorlabs.kinesis.enums.PPCatDipoleIntensity*
attribute), 450
C (*msl.equipment.resources.picotech.picoscope.enums.PS3000ACh*
attribute), 323

BroadcastAnswerType (class *in* *attribute*), 258
msl.equipment.resources.avantes.avaspec, *C* (*msl.equipment.resources.picotech.picoscope.enums.PS3000Char*
99
attribute), 252

buffer (*msl.equipment.resources.picotech.picoscope.C*~~hannel~~*buffer* (*msl.equipment.resources.picotech.picoscope.enums.PS4000ACh*
property), 232
attribute), 281

build_device_list() (*msl.equipment.resources.picotech.picoscope.C*~~hannel~~*DeviceList* (*msl.equipment.resources.picotech.picoscope.enums.PS4000Char*
static method), 569
C (*msl.equipment.resources.picotech.picoscope.enums.PS5000ACh*
attribute), 297

build_group() (*msl.equipment.resources.bentham.benhw32*), 3205
method), 128
C (*msl.equipment.resources.picotech.picoscope.enums.PS5000Char*
attribute), 296

build_system_model() (*msl.equipment.resources.bentham.benhw32*), 3205
method), 128
C (*msl.equipment.resources.picotech.picoscope.enums.PS6000Char*
attribute), 315

build_system_model() (*msl.equipment.resources.picotech.picoscope.C_MAX* (*msl.equipment.resources.picotech.picoscope.enums.PS2000Char*
attribute), 296

attribute), 240

C_MAX(*msl.equipment.resources.picotech.picoscope.cameras.PS4000ACameraChannelBufferIndex.CHeader attribute*), 258

C_MAX(*msl.equipment.resources.picotech.picoscope.cameras.PS4000ACameraChannelBufferIndex(index in module attribute)*), 282

C_MAX(*msl.equipment.resources.picotech.picoscope.cameras.PS4000ACameraChannelBufferIndex(attribute)*), 269

C_MAX(*msl.equipment.resources.picotech.picoscope.enums.PS4000ACameraMeasurement attribute*), 306

C_MAX(*msl.equipment.resources.picotech.picoscope.enums.PS4000ACameraMeasurement(attribute)*), 297

C_MAX(*msl.equipment.resources.picotech.picoscope.cameras.PS4000ACameraMeasurementIndex(attribute)*), 315

C_MIN(*msl.equipment.resources.picotech.picoscope.enums.PS2000AChannelBufferIndex attribute*), 240

C_MIN(*msl.equipment.resources.picotech.picoscope.enums.PS3000AChannelBufferIndex(attribute)*), 258

C_MIN(*msl.equipment.resources.picotech.picoscope.cameras.PS4000ACameraMeasurementIndex(attribute)*), 282

C_MIN(*msl.equipment.resources.picotech.picoscope.enums.PS4000AChannelBufferIndex(attribute)*), 269

C_MIN(*msl.equipment.resources.picotech.picoscope.enums.PS4000ACameraMeasurementIndex(attribute)*), 306

C_MIN(*msl.equipment.resources.picotech.picoscope.enums.PS4000ACameraMeasurementIndex(attribute)*), 297

C_MIN(*msl.equipment.resources.picotech.picoscope.enums.PS4000ACameraMeasurementIndex(attribute)*), 315

Cage_Rotator(*msl.equipment.resources.thorlabs.kinesis.motion.MotionControl attribute*), 568

CAL_DATE(*msl.equipment.resources.picotech.picoscope.enums.PS2000ACameraMeasurement(attribute)*), 233

CAL_DATE(*msl.equipment.resources.picotech.picoscope.enums.PS2000ACameraMeasurement(attribute)*), 235

CAL_DATE(*msl.equipment.resources.picotech.picoscope.enums.PS3000ACameraMeasurement(attribute)*), 254

calc_mono_slit_bandpass()

calibration(*msl.equipment.record_types.MeasurandRecord(attribute)*), 84

calibration_cycle

calibration_date

CalibrationRecord

calibrations

CALL

tribute), 602

method), 89

module msl.equipment.resources.utils), 88

(msl.equipment.resources.bentham.benhw32.Bentham32 camera_measurement()

(msl.equipment.resources.bentham.benhw32.Bentham32 method), 129

(msl.equipment.resources.bentham.benhw32.Bentham32 can_device_lock_front_panel()

(method), 505

(msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo can_debounce(attribute)

(msl.equipment.resources.thorlabs.kinesis.benchtop_s can_home()

(msl.equipment.resources.thorlabs.kinesis.integrated_ can_home()

(msl.equipment.resources.thorlabs.kinesis.kcube_dc_s can_home()

(msl.equipment.resources.thorlabs.kinesis.kcube_step can_home()

(msl.equipment.resources.thorlabs.kinesis.kcube_dc_s can_home()

(msl.equipment.resources.thorlabs.kinesis.kcube_dc_s can_home()

(msl.equipment.resources.thorlabs.kinesis.kcube_dc_s can_home()

(msl.equipment.resources.thorlabs.kinesis.kcube_dc_s can_move_without_homing_first()

(msl.equipment.resources.thorlabs.kinesis.integrated_step can_move_without_homing_first()

(msl.equipment.resources.thorlabs.kinesis.kcube_dc_s can_move_without_homing_first()

(msl.equipment.resources.princeton_instruments.arc_princeton_instruments can_move_without_homing_first()

(method), 539

(Capacitive(msl.equipment.resources.thorlabs.kinesis.enums.PPC attribute), 449

(capture()) (msl.equipment.resources.dataray.datarayocx_32.Data method), 133

(capture()) (msl.equipment.resources.dataray.datarayocx_64.Data method), 134

(category(msl.equipment.record_types.EquipmentRecord attribute), 80

(charge(power_source()) (msl.equipment.resources.picotech.picoscope.picoscope_a at- method), 331

changeToOddOrEven
(*msl.equipment.resources.thorlabs.kinesis.structs.NTAttribute*), 585
changeToOddOrEven
(*msl.equipment.resources.thorlabs.kinesis.structs.NTAttribute*), 578
channel (*msl.equipment.resources.picotech.picoscope.channelProperties*)
property), 231
CHANNEL (*msl.equipment.resources.picotech.picoscope.enums.PS4000AP16StringValue*
attribute), 293
channel (*msl.equipment.resources.picotech.picoscope.picoscopeProperties*)
property), 325
channel (*msl.equipment.resources.picotech.picoscope.structs.PS2000ADigitalChannelDirections*
attribute), 360
channel (*msl.equipment.resources.picotech.picoscope.structs.PS2000ATriggerChannelProperties*
attribute), 360
channel (*msl.equipment.resources.picotech.picoscope.structs.PS2000ATriggerChannelProperties*
attribute), 358
channel (*msl.equipment.resources.picotech.picoscope.structs.PS3000ADigitalChannelDirections*
attribute), 363
channel (*msl.equipment.resources.picotech.picoscope.structs.PS3000ATriggerChannelProperties*
attribute), 363
channel (*msl.equipment.resources.picotech.picoscope.structs.PS3000ATriggerChannelProperties*
attribute), 360
channel (*msl.equipment.resources.picotech.picoscope.structs.PS3000ATriggerChannelProperties*
attribute), 360
channel (*msl.equipment.resources.picotech.picoscope.structs.PS4000AC10ChannelLedSetting*
attribute), 365
channel (*msl.equipment.resources.picotech.picoscope.structs.PS4000AC20ConnectDetect*
attribute), 366
channel (*msl.equipment.resources.picotech.picoscope.structs.PS4000AC42Direction*
attribute), 365
channel (*msl.equipment.resources.picotech.picoscope.structs.PS4000ATriggerChannelProperties*
attribute), 366
channel (*msl.equipment.resources.picotech.picoscope.structs.PS4000ATriggerChannelProperties*
attribute), 365
channel (*msl.equipment.resources.picotech.picoscope.structs.PS5000ATriggerChannelProperties*
attribute), 369
channel (*msl.equipment.resources.picotech.picoscope.structs.PS5000ATriggerChannelProperties*
attribute), 367
channel (*msl.equipment.resources.picotech.picoscope.structs.PS5000ATriggerChannelProperties*
attribute), 370
channel (*msl.equipment.resources.thorlabs.kinesis.structs.BNT110Settings*
attribute), 577
Channel1 (*msl.equipment.resources.thorlabs.kinesis.enums.KIM16Bit*)
attribute), 453
Channel11 (*msl.equipment.resources.thorlabs.kinesis.enums.TM16Bit*)
attribute), 474
Channel10only (*msl.equipment.resources.thorlabs.kinesis.enums.ChannelEnableModes*
attribute), 456
Channel12 (*msl.equipment.resources.thorlabs.kinesis.enums.KIM16Bit*)
attribute), 453
Channel2 (*msl.equipment.resources.thorlabs.kinesis.enums.TM16Bit*)
attribute), 470
attribute), 474
CHANNEL_0 (*msl.equipment.resources.thorlabs.kinesis.enums.C*
attribute), 456
Channel13 (*msl.equipment.resources.thorlabs.kinesis.enums.KIM_C*
attribute), 456
Channel13only (*msl.equipment.resources.thorlabs.kinesis.enums.TIM_C*
attribute), 293
Channel14 (*msl.equipment.resources.thorlabs.kinesis.enums.KIM_C*
attribute), 293
Channel14only (*msl.equipment.resources.thorlabs.kinesis.enums.TIM_C*
attribute), 293
CHANNEL_0 (*msl.equipment.resources.picotech.picoscope.enums.PS*
attribute), 360
CHANNEL_1 (*msl.equipment.resources.picotech.picoscope.enums.PS*
attribute), 360
CHANNEL_10 (*msl.equipment.resources.picotech.picoscope.enums.PS*
attribute), 360
CHANNEL_11 (*msl.equipment.resources.picotech.picoscope.enums.PS*
attribute), 360
CHANNEL_12 (*msl.equipment.resources.picotech.picoscope.enums.PS*
attribute), 360
CHANNEL_13 (*msl.equipment.resources.picotech.picoscope.enums.PS*
attribute), 360
CHANNEL_14 (*msl.equipment.resources.picotech.picoscope.enums.PS*
attribute), 360
CHANNEL_15 (*msl.equipment.resources.picotech.picoscope.enums.P*
attribute), 453
CHANNEL_15only (*msl.equipment.resources.picotech.picoscope.enums.ChannelEnableModes*
attribute), 456
CHANNEL_16 (*msl.equipment.resources.picotech.picoscope.enums.P*
attribute), 453
CHANNEL_16only (*msl.equipment.resources.picotech.picoscope.enums.ChannelEnableModes*
attribute), 456

CHANNEL_17 (*msl.equipment.resources.picotech.picoscope.enums.PS2000ADigitalChannel*
attribute), 242 CHANNEL_29 (*msl.equipment.resources.picotech.picoscope.enums.PS3000ADigitalChannel*
attribute), 260 CHANNEL_29 (*msl.equipment.resources.picotech.picoscope.enums.PS3000ADigitalChannel*
attribute), 242 CHANNEL_3 (*msl.equipment.resources.picotech.picoscope.enums.PS3000ADigitalChannel*
attribute), 260 CHANNEL_3 (*msl.equipment.resources.picotech.picoscope.enums.PS3000ADigitalChannel*
attribute), 242 CHANNEL_30 (*msl.equipment.resources.picotech.picoscope.enums.PS3000ADigitalChannel*
attribute), 260 CHANNEL_30 (*msl.equipment.resources.picotech.picoscope.enums.PS3000ADigitalChannel*
attribute), 242 CHANNEL_31 (*msl.equipment.resources.picotech.picoscope.enums.PS3000ADigitalChannel*
attribute), 259 CHANNEL_31 (*msl.equipment.resources.picotech.picoscope.enums.PS3000ADigitalChannel*
attribute), 242 CHANNEL_4 (*msl.equipment.resources.picotech.picoscope.enums.PS3000ADigitalChannel*
attribute), 260 CHANNEL_4 (*msl.equipment.resources.picotech.picoscope.enums.PS3000ADigitalChannel*
attribute), 242 CHANNEL_5 (*msl.equipment.resources.picotech.picoscope.enums.PS3000ADigitalChannel*
attribute), 260 CHANNEL_5 (*msl.equipment.resources.picotech.picoscope.enums.PS3000ADigitalChannel*
attribute), 242 CHANNEL_6 (*msl.equipment.resources.picotech.picoscope.enums.PS3000ADigitalChannel*
attribute), 242 CHANNEL_6 (*msl.equipment.resources.picotech.picoscope.enums.PS3000ADigitalChannel*
attribute), 260 CHANNEL_7 (*msl.equipment.resources.picotech.picoscope.enums.PS3000ADigitalChannel*
attribute), 243 CHANNEL_7 (*msl.equipment.resources.picotech.picoscope.enums.PS3000ADigitalChannel*
attribute), 260 CHANNEL_8 (*msl.equipment.resources.picotech.picoscope.enums.PS3000ADigitalChannel*
attribute), 243 CHANNEL_8 (*msl.equipment.resources.picotech.picoscope.enums.PS3000ADigitalChannel*
attribute), 260 CHANNEL_9 (*msl.equipment.resources.picotech.picoscope.enums.PS3000ADigitalChannel*
attribute), 243 CHANNEL_9 (*msl.equipment.resources.picotech.picoscope.enums.PS3000ADigitalChannel*
attribute), 260 CHANNEL_ANALOGUE_OFFSET CHANNEL_BANDWIDTH (*msl.equipment.resources.picotech.picoscope.enums.PS4000ADigitalChannel*
attribute), 293 CHANNEL_BANDWIDTH (*msl.equipment.resources.picotech.picoscope.enums.PS4000ADigitalChannel*
attribute), 243 CHANNEL_CHANNEL_COUNT () CHANNEL_DIGITAL_CHANGE (*msl.equipment.resources.picotech.picoscope.enums.PS4000ADigitalChannel*
attribute), 148 CHANNEL_EQUIVLOG (*msl.equipment.resources.picotech.picoscope.enums.PS4000ADigitalChannel*
attribute), 243 CHANNEL_EQUIVLOG (*msl.equipment.resources.picotech.picoscope.enums.PS4000ADigitalChannel*
attribute), 260

attribute), 361
channel1C(msl.equipment.resources.picotech.picoscope.structs.PS4000TriggerConditions
attribute), 361
channel1C(msl.equipment.resources.picotech.picoscope.structs.PS4000P369Conditions
attribute), 365
channel1C(msl.equipment.resources.picotech.picoscope.structs.PS4000TriggerConditions
attribute), 364
channel1C(msl.equipment.resources.picotech.picoscope.structs.PS4000TriggerConditions
attribute), 369
channel1C(msl.equipment.resources.picotech.picoscope.structs.PS4000TriggerConditions
attribute), 368
channel1C(msl.equipment.resources.picotech.picoscope.structs.PS4000TriggerConditions
attribute), 367
channel1C(msl.equipment.resources.picotech.picoscope.structs.PS4000TriggerConditions
attribute), 366
channel1C(msl.equipment.resources.picotech.picoscope.structs.PS6000PwqConditions
attribute), 370
channel1C(msl.equipment.resources.picotech.picoscope.structs.PS6000TriggerConditions
attribute), 369
channel1D(msl.equipment.resources.picotech.picoscope.structs.PS2000PwqConditions
attribute), 359
channel1D(msl.equipment.resources.picotech.picoscope.structs.PS2000ATriggerConditions
attribute), 359
channel1D(msl.equipment.resources.picotech.picoscope.structs.PS2000ATriggerConditions
attribute), 359
channel1D(msl.equipment.resources.picotech.picoscope.structs.PS2000ATriggerConditions
attribute), 358
channel1D(msl.equipment.resources.picotech.picoscope.structs.PS2000ATriggerConditions
attribute), 358
channel1D(msl.equipment.resources.picotech.picoscope.structs.PS2000ATriggerConditions
attribute), 362
channel1D(msl.equipment.resources.picotech.picoscope.structs.PS2000ATriggerConditions
attribute), 363
channel1D(msl.equipment.resources.picotech.picoscope.structs.PS2000ATriggerConditions
attribute), 361
channel1D(msl.equipment.resources.picotech.picoscope.structs.PS2000ATriggerConditionsV2
attribute), 362
channel1D(msl.equipment.resources.picotech.picoscope.structs.PS2000ATriggerConditionspicotech.picoscope.helper),
attribute), 361
channel1D(msl.equipment.resources.picotech.picoscope.structs.PS4000TriggerConditions1.RPCClient
attribute), 361
channel1D(msl.equipment.resources.picotech.picoscope.structs.PS4000PwqConditionsxii11.RPCClient
attribute), 365
channel1D(msl.equipment.resources.picotech.picoscope.structs.PS4000TriggerConditionspicotech.picoscope.enums.PS4000
attribute), 364
channel1D(msl.equipment.resources.picotech.picoscope.structs.PS5000PwqCondition_gpib.ConnectionGPIB
attribute), 369
channel1D(msl.equipment.resources.picotech.picoscope.structs.PS5000TriggerCondition_tcpip_hislip.ConnectionTCPI
attribute), 368
channel1D(msl.equipment.resources.picotech.picoscope.structs.PS5000PwqCondition_tcpip_vxi11.ConnectionTCP
attribute), 367
channel1D(msl.equipment.resources.picotech.picoscope.structs.PS5000TriggerConditionusim_tti.mx_series.MXSeries
attribute), 367

clear_max_value() static method), 374
(msl.equipment.resources.greisinger.gmh3000.GMH3000() (msl.equipment.resources.nkt.nktpdll.NKT
method), 148 static method), 172

clear_message_queue() close_shutter()
(msl.equipment.resources.thorlabs.kinesis.benchtop_kinematics.BenchtopStepperMotor.benhw32.Bentham32
method), 403 method), 128

clear_message_queue() close_unit() (msl.equipment.resources.picotech.picoscope.picos
(msl.equipment.resources.thorlabs.kinesis.filter_flipper.Flipper
method), 482 closedTime (msl.equipment.resources.thorlabs.kinesis structs.SC_
attribute), 591

clear_message_queue() close_unit() (msl.equipment.resources.thorlabs.kinesis.integrated_stepper_motors.IntegratedStepperMotors
method), 484 COMMA_SEPERATED

clear_message_queue() command() (msl.equipment.connection_gpib.ConnectionGPIB
(msl.equipment.resources.thorlabs.kinesis.kcube_dc_attributes.DCServo
method), 506 command(), 23

clear_message_queue() command() (msl.equipment.resources.picotech.picoscope.enums.PS4000
(msl.equipment.resources.thorlabs.kinesis.kcube_dc_attributes.MaintenanceRecord
method), 530 attribute), 84

clear_message_queue() COMPLETE (msl.equipment.resources.picotech.picoscope.enums.PS2000
(msl.equipment.resources.thorlabs.kinesis.kcube_dc_attributes.KCubeStepperMotor
method), 539 COMPLETE (msl.equipment.resources.picotech.picoscope.enums.PS3000
attribute), 255

clear_min_value() select_w1()
(msl.equipment.resources.greisinger.gmh3000.GMH3000()
method), 148 (msl.equipment.resources.bentham.benhw32.Bentham32
method), 128

clockwiseHardwareLimit condition_register()
(msl.equipment.resources.thorlabs.kinesis structs.MQTSwitchPositioner.energetiq.eq99.EQ99
attribute), 576 method), 142

clockwisePosition conditions (msl.equipment.record_types.MeasurandRecord
attribute), 84

close() config() (msl.equipment.hislip.HiSLIPClient
method), 73 config() (msl.equipment.connection_gpib.ConnectionGPIB
method), 128

close() config() (msl.equipment.resources.bentham.benhw32.Bentham32
method), 128 config() (msl.equipment.connection_gpib.ConnectionGPIB
method), 17

close() CONFIG_FAIL (msl.equipment.resources.picotech.errors.PS2000Er
(msl.equipment.resources.thorlabs.fwxx2c.FilterWheelX2C(), 23
method), 595 CONFIG_FAIL (msl.equipment.resources.picotech.errors.PS3000Er
method), 403 CONFIG_FAIL (msl.equipment.resources.picotech.errors.PS3000Er
method), 403

close() connect() (in module msl.equipment.factory), 56
(msl.equipment.resources.thorlabs.kinesis.filter_flipper.Flipper
method), 479 connect(), 56

close() connect() (msl.equipment.resources.thorlabs.kinesis integrated_stepper_motors.BenchtopStepperMotor
method), 484 connect(), 73

close() connect() (msl.equipment.resources.picotech.picoscope.picos
(msl.equipment.resources.thorlabs.kinesis.kcube_dc_attributes.EquipmentRecord
method), 506 method), 81

close() connect() (msl.equipment.resources.thorlabs.kinesis.kcube_dc_attributes.KCubeStepperMotor
method), 530 method), 603

close() connect() (msl.equipment.resources.thorlabs.kinesis.kcube_dc_attributes.KCubeStepperMotor
method), 539 (msl.equipment.resources.picotech.picoscope.ps4000a.Pic
method), 352

close() CONNECT_STATE_FLOATING
(msl.equipment.vxi11.RPCClient
method), 603

close_enum() (msl.equipment.resources.princeton_instrument_ps4000a.Pic
method), 352

attribute), 290

connect_to_ol756() *(msl.equipment.resources.optronic_laboratories.ol756_attributes, 166)* *continuousCurrentLimit attribute), 575*

Connection (*class in msl.equipment.connection*), **control_atn()** (*msl.equipment.connection_gpib.ConnectionGPIB* method), 24

connection (*msl.equipment.record_types.EquipmentControl*), **control_atn()** (*msl.equipment.connection_gpib.ConnectionGPIB* attribute), 24

ConnectionDemo (*class in msl.equipment.connection_demo*), **in controller** (*msl.equipment.connection_prologix.ConnectionPrologix* property), 34

ConnectionGPIB (*class in msl.equipment.connection_gpib*), **controllers** (*msl.equipment.connection_prologix.ConnectionPrologix* attribute), 34

ConnectionMessageBased (*class in msl.equipment.connection_message_based*) **controlMode** (*msl.equipment.resources.thorlabs.kinesis.structs.NT* attribute), 579

ConnectionNIDAQ (*class in msl.equipment.connection_nidaq*), **ControlSettingsType** (*class in msl.equipment.resources.avantes.avaspec*), 100

ConnectionPrologix (*class in msl.equipment.connection_prologix*), **convert_datetime()** (*in module msl.equipment.resources.omega.ithx*), 199

ConnectionPyVISA (*class in msl.equipment.connection_pyvisa*), **convert_message()** (*msl.equipment.resources.thorlabs.kinesis.motion_controller* static method), 570

ConnectionRecord (*class in msl.equipment.record_types*), 85

connections() (*msl.equipment.database.Database* method), 52

ConnectionSDK (*class in msl.equipment.connection_sdk*), 38

ConnectionSerial (*class in msl.equipment.connection_serial*), 39

ConnectionSocket (*class in msl.equipment.connection_socket*), 41

ConnectionTCPIPHiSLIP (*class in msl.equipment.connection_tcpip_hislip*), 42

ConnectionTCPiPVXI11 (*class in msl.equipment.connection_tcpip_vxi11*), 45

ConnectionZeroMQ (*class in msl.equipment.connection_zeromq*), 48

constants (*msl.equipment.connection_nidaq.ConnectionNIDAQ* attribute), 29

constants() (*msl.equipment.resources.utils.CHeader* method), 88

context (*msl.equipment.connection_zeromq.ConnectionZeroMQ*), **create_zero_mq_chan()**

CoreClient (*class in msl.equipment.vxi11*), 605

count() (*msl.equipment.connection_gpib.ConnectionGPIB* method), 25

countsPerUnit (*msl.equipment.resources.thorlabs.kinesis.structs* attribute), 573

coupling (*msl.equipment.resources.picotech.picoscope.channel.PicoScopeChannel* property), 231

CR (*msl.equipment.connection_message_based.ConnectionMessageBased*), 231

create_callbacks_file() (*in module msl.equipment.resources.picotech.picoscope.helper*), 324

<code>(msl.equipment.connection_tcpip_vxi11.ConnectionTCPPIPWithSources.picotech.picoscope.enums.PS3000Char method), 48</code>	<code>D (msl.equipment.resources.picotech.picoscope.enums.PS4000AChar attribute), 252</code>
<code>create_intr_chan()</code>	<code>D (msl.equipment.resources.picotech.picoscope.enums.PS4000AChar attribute), 281</code>
<code>(msl.equipment.vxi11.CoreClient method), 608</code>	<code>D (msl.equipment.resources.picotech.picoscope.enums.PS4000Char attribute), 270</code>
<code>create_link() (msl.equipment.vxi11.CoreClient method), 605</code>	<code>D (msl.equipment.resources.picotech.picoscope.enums.PS5000AChar attribute), 305</code>
<code>create_picoscopeEnums_file() (in module msl.equipment.resources.picotech.picoscope)</code>	<code>D (msl.equipment.resources.picotech.picoscope.enums.PS5000Char attribute), 296</code>
<code>324</code>	
<code>create_picoscopeFunctions_file()</code>	<code>D (msl.equipment.resources.picotech.picoscope.enums.PS6000Char attribute), 315</code>
<code>(in module msl.equipment.resources.picotech.picoscope)</code>	<code>D (msl.equipment.resources.picotech.picoscope.enums.PS2000Char attribute), 240</code>
<code>324</code>	
<code>create_picoscopeStructs_file()</code>	<code>D_MAX (msl.equipment.resources.picotech.picoscope.enums.PS3000Char attribute), 258</code>
<code>(in module msl.equipment.resources.picotech.picoscope)</code>	<code>D (msl.equipment.resources.picotech.picoscope.enums.PS4000Char attribute), 282</code>
<code>324</code>	
<code>CtrFreq (msl.equipment.connection_nidaq.ConnectionNIDAQ property), 32</code>	<code>D (msl.equipment.resources.picotech.picoscope.enums.PS4000Char attribute), 269</code>
<code>CtrTick (msl.equipment.connection_nidaq.ConnectionNIDAQ property), 32</code>	<code>D (msl.equipment.resources.picotech.picoscope.enums.PS5000Char attribute), 306</code>
<code>CtrTime (msl.equipment.connection_nidaq.ConnectionNIDAQ property), 32</code>	<code>D (msl.equipment.resources.picotech.picoscope.enums.PS5000Char attribute), 297</code>
<code>ctypes_map()</code>	<code>D_MAX (msl.equipment.resources.picotech.picoscope.enums.PS6000Char attribute), 315</code>
<code>(in module msl.equipment.resources.picotech.picoscope.helper), 324</code>	<code>D_MIN (msl.equipment.resources.picotech.picoscope.enums.PS2000Char attribute), 240</code>
<code>CURRENT_CLAMP_1000A</code>	<code>D_MIN (msl.equipment.resources.picotech.picoscope.enums.PS3000Char attribute), 258</code>
<code>(msl.equipment.resources.picotech.picoscope)</code>	
<code>attribute), 272</code>	
<code>CURRENT_CLAMP_10A</code>	<code>D_MIN (msl.equipment.resources.picotech.picoscope.enums.PS4000Char attribute), 282</code>
<code>(msl.equipment.resources.picotech.picoscope.enums.PS4000Char attribute), 272</code>	<code>D_MIN (msl.equipment.resources.picotech.picoscope.enums.PS4000Char attribute), 282</code>
<code>CURRENT_MEASURING_DEVICE</code>	<code>D_MIN (msl.equipment.resources.picotech.picoscope.enums.PS4000Char attribute), 269</code>
<code>(msl.equipment.resources.picotech.picoscope)</code>	
<code>attribute), 272</code>	<code>D_MIN (msl.equipment.resources.picotech.picoscope.enums.PS5000Char attribute), 306</code>
<code>currentPowerSource()</code>	<code>D_MIN (msl.equipment.resources.picotech.picoscope.enums.PS5000Char attribute), 287</code>
<code>(msl.equipment.resources.picotech.picoscope.picoscopeApi method), 332</code>	<code>D_MIN (msl.equipment.resources.picotech.picoscope.enums.PS6000Char attribute), 327</code>
<code>CycleCount (msl.equipment.resources.thorlabs.kinesis.structs.KN016TriggerParams attribute), 583</code>	<code>DaqError (msl.equipment.connection_nidaq.ConnectionNIDAQ property), 32</code>
<code>cycleLength (msl.equipment.resources.thorlabs.kinesis.structs.KN016TriggerParams attribute), 579</code>	<code>DaqResourceWarning (msl.equipment.connection_nidaq.ConnectionNIDAQ property), 32</code>
<code>D</code>	
<code>D (msl.equipment.resources.picotech.picoscope.enums.PS2000Char attribute), 240</code>	<code>D (msl.equipment.resources.picotech.picoscope.enums.PS2000Char attribute), 32</code>
<code>D (msl.equipment.resources.picotech.picoscope.enums.PS2000Char attribute), 233</code>	<code>D (msl.equipment.resources.picotech.picoscope.enums.PS2000Char attribute), 233</code>
<code>D (msl.equipment.resources.picotech.picoscope.enums.PS3000Char attribute), 258</code>	<code>DAQ2000CharType (class in msl.equipment.resources.avantes.avaspec), Data (class in msl.equipment.hislip), 62</code>

data (*msl.equipment.hislip.AuthenticationResult* DC_VOLTAGE (*msl.equipment.resources.picotech.picoscope.enums.POWER_PROPERTY*), 73
attribute), 285
data (*msl.equipment.hislip.GetSaslMechanismListResponse* DC_VOLTAGE (*msl.equipment.resources.picotech.picoscope.enums.POWER_PROPERTY*), 72
attribute), 275
Data (*msl.equipment.hislip.MessageType* at- DC_VOLTAGE (*msl.equipment.resources.picotech.picoscope.enums.POWER_ATTRIBUTE*), 57
attribute), 309
data() (*msl.equipment.resources.omega.ithx.iTHX* DC_VOLTAGE (*msl.equipment.resources.picotech.picoscope.enums.HIGH_ATTRIBUTE*), 199
static method), 300
data_bits (*msl.equipment.connection_serial.Connection* DC_VOLTAGE (*msl.equipment.resources.picotech.picoscope.enums.HIGH_ATTRIBUTE*), 40
property), 318
Database (*class in msl.equipment.database*), 51
database() (*msl.equipment.config.Config* method), 18
DataBits (*class in msl.equipment.constants*), 51
DataEnd (*class in msl.equipment.hislip*), 62
DataEnd (*msl.equipment.hislip.MessageType* attribute), 57
DataRayError, 55
DataRayOCX32 (*class in msl.equipment.resources.dataray.datarayocx_32*), 133
DataRayOCX64 (*class in msl.equipment.resources.dataray.datarayocx_64*), 133
DataType (*msl.equipment.resources.picotech.pt104*.DECIMATE attribute), 372
date (*msl.equipment.record_types.MaintenanceRecord* attribute), 84
DateTimeType (*class in msl.equipment.resources.nkt.nktpdll*), 162
Day (*msl.equipment.resources.nkt.nktpdll.DateTimeType* attribute), 162
DC (*msl.equipment.resources.picotech.picoscope.enums.PS2000A* attribute), 244
DC (*msl.equipment.resources.picotech.picoscope.enums.PS3000A* attribute), 261
DC (*msl.equipment.resources.picotech.picoscope.enums.PS4000A* attribute), 281
DC (*msl.equipment.resources.picotech.picoscope.enums.PS5000A* attribute), 305
DC_1M (*msl.equipment.resources.picotech.picoscope.enums.PS6000A* attribute), 317
DC_50R (*msl.equipment.resources.picotech.picoscope.enums.PS6000A* attribute), 317
DC_VOLTAGE (*msl.equipment.resources.picotech.picoscope.enums.PS2000AWaveType* attribute), 246
DC_VOLTAGE (*msl.equipment.resources.picotech.picoscope.enums.PS2000AWaveType* attribute), 237
DC_VOLTAGE (*msl.equipment.resources.picotech.picoscope.enums.PS3000AWaveType* attribute), 264
attribute), 285
attribute), 275
attribute), 309
attribute), 300
attribute), 318
deactivate() (*msl.equipment.resources.avantes.avaspec.Avantes* method), 116
deadErrorBand (*msl.equipment.resources.thorlabs.kinesis structs.QD* attribute), 575
DECIMATE (*msl.equipment.resources.picotech.picoscope.enums.PS2000A* attribute), 250
DECIMATE (*msl.equipment.resources.picotech.picoscope.enums.PS3000A* attribute), 268
DECIMATE (*msl.equipment.resources.picotech.picoscope.enums.PS4000A* attribute), 292
DECIMATE (*msl.equipment.resources.picotech.picoscope.enums.PS5000A* attribute), 313
DECIMATE (*msl.equipment.resources.picotech.picoscope.enums.PS6000A* attribute), 303
DECIMATE (*msl.equipment.resources.picotech.picoscope.enums.PS7000A* attribute), 322
decrement_current() (*msl.equipment.resources.aim_tti.mx_series.MXSeries* method), 90
decrement_voltage() (*msl.equipment.resources.aim_tti.mx_series.MXSeries* method), 90
default() (*msl.equipment.resources.mks_instruments.pr4000b.PP* method), 142
delay() (*msl.equipment.resources.energetiq.eq99.EQ99* method), 142
delete_group() (*msl.equipment.resources.bentham.benhw32.Bentham32* method), 129
derivative (*msl.equipment.resources.thorlabs.kinesis structs.QD* attribute), 590
DEMO_MODE (*msl.equipment.config.Config* attribute), 18
Demonstrator (*msl.equipment.resources.nkt.nktpdll.ParameterSetT* attribute), 163
DerivFilterOff (*msl.equipment.resources.thorlabs.kinesis structs.MOT_B* attribute), 447

DerivFilterOn(*msl.equipment.resources.thorlabs.kinesis.enums.DP_C600*.DerivFilterState attribute), 447
device_exists()
description(*msl.equipment.record_types.EquipmentRecord*(*msl.equipment.resources.nkt.nktpdll.NKT* attribute), 80
method), 173
description(*msl.equipment.resources.thorlabs.kindestorage.LAL*.DeviceType),
attribute), 570
(msl.equipment.resources.nkt.nktpdll.NKT static method), 173
destroy_intr_chan()
(msl.equipment.connection_tcpip_vxi11.CoreClient method), 48
device_get_boot_loader_version()
(msl.equipment.resources.nkt.nktpdll.NKT method), 173
destroy_intr_chan()
(msl.equipment.vxi11.CoreClient method), 608
device_get_boot_loader_version_str()
(msl.equipment.resources.nkt.nktpdll.NKT method), 174
destroy_link()
(msl.equipment.connection_tcpip_vxi11.CoreClient method), 48
device_get_firmware_code()
(msl.equipment.resources.nkt.nktpdll.NKT method), 174
destroy_link()
(msl.equipment.vxi11.CoreClient method), 608
device_get_firmware_version()
(msl.equipment.resources.nkt.nktpdll.NKT method), 174
det_nonblock_read_done()
(msl.equipment.resources.princeton_instruments.device_get_firmware_version() method), 375
(msl.equipment.resources.nkt.nktpdll.NKT method), 174
det_read() (*msl.equipment.resources.princeton_instruments*.PrincetonInstruments method), 374
device_get_live()
det_readall() (*msl.equipment.resources.princeton_instruments*.PrincetonInstruments method), 374
device_get_mode()
det_start_nonblock_read()
(msl.equipment.resources.princeton_instruments.arc.(*instaqipm*.PrincetonInstruments) method), 375
(msl.equipment.resources.nkt.nktpdll.NKT method), 175
DetectorType (class in *msl.equipment.resources.avantes.avaspec*), 100
device_abort()
(msl.equipment.vxi11.AsyncClient method), 609
device_get_module_serial_number_str()
(msl.equipment.resources.nkt.nktpdll.NKT method), 175
DEVICE_CAPABILITY
(msl.equipment.resources.picotech.picoscope.enums.PS4000AItemTypesources.nkt.nktpdll.NKT attribute), 286
device_get_part_number_str()
(msl.equipment.resources.nkt.nktpdll.NKT method), 175
device_get_pcb_serial_number_str()
(msl.equipment.resources.nkt.nktpdll.NKT method), 176
device_get_pcb_version()
(msl.equipment.resources.nkt.nktpdll.NKT method), 176
device_get_status_bits()
(msl.equipment.resources.nkt.nktpdll.NKT method), 176
device_get_type()
(msl.equipment.resources.nkt.nktpdll.NKT method), 176
device_local()
(msl.equipment.vxi11.CoreClient method), 607
device_lock() (*msl.equipment.vxi11.CoreClient* method), 607
device_enable_srq()
(msl.equipment.vxi11.CoreClient

device_manager()	(in module <code>msl.equipment.resources.thorlabs.kinesis.motion_controller</code>)	<code>DeviceConfigType</code>	(class <code>msl.equipment.resources.avantes.avaspec</code>), 566	<code>in</code>
device_meta_data()	(<code>msl.equipment.resources.picotech.picoscope.ps4000</code>)	<code>deviceDependantData</code>	(<code>msl.equipment.resources.picotech.picoscope.ps4000</code>)	<code>attribute</code> , 571
device_read()	(<code>msl.equipment.vxi11.CoreClient</code>)	<code>DeviceErrorCodeChanged</code>	(<code>msl.equipment.resources.nkt.nktpdll.DeviceStatusTypes</code>)	
device_readstb()	(<code>msl.equipment.vxi11.CoreClient</code>)	<code>DeviceErrorCodeChanged</code>	(<code>msl.equipment.resources.nkt.nktpdll.NKT.DeviceStatusTy</code>)	
device_remote()	(<code>msl.equipment.vxi11.CoreClient</code>)	<code>DeviceFwVerChanged</code>	(<code>msl.equipment.resources.nkt.nktpdll.DeviceStatusTypes</code>)	
device_remove()	(<code>msl.equipment.resources.nkt.nktpdll.NKT</code>)	<code>DeviceFwVerChanged</code>	(<code>msl.equipment.resources.nkt.nktpdll.NKT.DeviceStatusTy</code>)	
device_remove_all()	(<code>msl.equipment.resources.nkt.nktpdll.NKT</code>)	<code>DeviceLiveChanged</code>	(<code>msl.equipment.resources.nkt.nktpdll.DeviceStatusTypes</code>)	
device_set_live()	(<code>msl.equipment.resources.nkt.nktpdll.NKT</code>)	<code>DeviceLiveChanged</code>	(<code>msl.equipment.resources.nkt.nktpdll.NKT.DeviceStatusTy</code>)	
DEVICE_SETTINGS	(<code>msl.equipment.resources.picotech.picoscope.ps4000</code>)	<code>DeviceModuleAcknowledgeType</code>	(<code>msl.equipment.resources.nkt.nktpdll.DeviceStatusTypes</code>)	
device_trigger()	(<code>msl.equipment.vxi11.CoreClient</code>)	<code>DeviceModeChanged</code>	(<code>msl.equipment.resources.nkt.nktpdll.DeviceStatusTypes</code>)	
device_unlock()	(<code>msl.equipment.vxi11.CoreClient</code>)	<code>DeviceModeTypes</code>	(class <code>msl.equipment.resources.nkt.nktpdll</code>), 163	<code>in</code>
device_write()	(<code>msl.equipment.vxi11.CoreClient</code>)	<code>DeviceModuleSerialChanged</code>	(<code>msl.equipment.resources.nkt.nktpdll.DeviceStatusTypes</code>)	
DeviceBlVerChanged	(<code>msl.equipment.resources.nkt.nktpdll.DeviceStatusTy</code>)	<code>DeviceModuleSerialChanged</code>	(<code>msl.equipment.resources.nkt.nktpdll.NKT.DeviceStatusTy</code>)	
DeviceBlVerChanged	(<code>msl.equipment.resources.nkt.nktpdll.NKT.DeviceStatusTy</code>)	<code>DevicePartNumberChanged</code>	(<code>msl.equipment.resources.nkt.nktpdll.DeviceStatusTypes</code>)	
DeviceClearAcknowledge	(class <code>msl.equipment.hislip</code>), 66	<code>DevicePartNumberChanged</code>	(<code>msl.equipment.resources.nkt.nktpdll.DeviceStatusTypes</code>)	
DeviceClearAcknowledge	(<code>msl.equipment.hislip.MessageType</code>)	<code>DevicePCBSerialChanged</code>	(<code>msl.equipment.resources.nkt.nktpdll.NKT.DeviceStatusTy</code>)	
DeviceClearComplete	(class <code>msl.equipment.hislip</code>), 65	<code>DevicePCBSerialChanged</code>	(<code>msl.equipment.resources.nkt.nktpdll.DeviceStatusTypes</code>)	
DeviceClearComplete	(<code>msl.equipment.hislip.MessageType</code>)	<code>DevicePCBSerialChanged</code>	(<code>msl.equipment.resources.nkt.nktpdll.NKT.DeviceStatusTy</code>)	

DevicePCBVersionChanged	DevModeDisabled
(msl.equipment.resources.nkt.nktpdll.DeviceStatusTypesAttribute), 164	(msl.equipment.resources.nkt.nktpdll.NKT.DeviceModeTypeAttribute), 168
DevicePCBVersionChanged	DevModeError (msl.equipment.resources.nkt.nktpdll.DeviceModeTypeAttribute), 164
(msl.equipment.resources.nkt.nktpdll.NKT.DeviceStatusTypesAttribute), 169	(msl.equipment.resources.nkt.nktpdll.NKT.DeviceModeTypeAttribute), 169
DeviceStatus (class in msl.equipment.resources.avantes.avaspec), 97	DevModeLogDownload (msl.equipment.resources.nkt.nktpdll.DeviceModeTypesAttribute), 164
DeviceStatusBitsChanged	DevModeLogDownload (msl.equipment.resources.nkt.nktpdll.NKT.DeviceModeTypeAttribute), 164
(msl.equipment.resources.nkt.nktpdll.DeviceStatusTypesAttribute), 164	(msl.equipment.resources.nkt.nktpdll.NKT.DeviceModeTypeAttribute), 169
DeviceStatusBitsChanged	DevModeNormal (msl.equipment.resources.nkt.nktpdll.DeviceModeTypeAttribute), 164
(msl.equipment.resources.nkt.nktpdll.NKT.DeviceStatusTypesAttribute), 169	(msl.equipment.resources.nkt.nktpdll.DeviceModeTypeAttribute), 164
DeviceStatusCallback (in module msl.equipment.resources.nkt.nktpdll), 162	DevModeNormal (msl.equipment.resources.nkt.nktpdll.NKT.DeviceModeTypeAttribute), 168
DeviceStatusCallback	DevModeTimeout
(msl.equipment.resources.nkt.nktpdll.NKT.DeviceStatusTypesAttribute), 168	(msl.equipment.resources.nkt.nktpdll.DeviceModeTypeAttribute), 164
DeviceStatusTypes (class in msl.equipment.resources.nkt.nktpdll), 164	DevModeTimeout
DeviceSysTypeChanged	DevModeUpload (msl.equipment.resources.nkt.nktpdll.DeviceModeTypeAttribute), 164
(msl.equipment.resources.nkt.nktpdll.DeviceStatusTypesAttribute), 164	(msl.equipment.resources.nkt.nktpdll.NKT.DeviceModeTypeAttribute), 169
DeviceSysTypeChanged	dewpoint() (msl.equipment.resources.omega.ithx.iTHX_DeviceStatusTypes), 196
(msl.equipment.resources.nkt.nktpdll.NKT.DeviceStatusTypesAttribute), 169	diameter (msl.equipment.resources.thorlabs.kinesis.structs.NT_CAttribute), 577
DeviceTypeChanged	DIFFERENTIAL_TO_115MV
(msl.equipment.resources.nkt.nktpdll.DeviceStatusTypesAttribute), 164	(msl.equipment.resources.picotech.pt104.Pt104DataTypeAttribute), 371
DeviceTypeChanged	DIFFERENTIAL_TO_2500MV
(msl.equipment.resources.nkt.nktpdll.NKT.DeviceStatusTypesAttribute), 169	(msl.equipment.resources.picotech.pt104.Pt104DataTypeAttribute), 371
DevModeAnalyze	MOT_B_PotentialGain
(msl.equipment.resources.nkt.nktpdll.DeviceStatusTypesAttribute), 164	(msl.equipment.resources.thorlabs.kinesis.structs.MOT_BAttribute), 574
DevModeAnalyze	MOT_D_PotentialGain
(msl.equipment.resources.nkt.nktpdll.NKT.DeviceStatusTypesAttribute), 168	(msl.equipment.resources.thorlabs.kinesis.structs.MOT_DAttribute), 576
DevModeAnalyzeInit	MOT_P_PotentialGain
(msl.equipment.resources.nkt.nktpdll.DeviceStatusTypesAttribute), 163	(msl.equipment.resources.thorlabs.kinesis.structs.MOT_PAttribute), 581
DevModeAnalyzeInit	QD_L_PotentialGain
(msl.equipment.resources.nkt.nktpdll.NKT.DeviceStatusTypesAttribute), 168	(msl.equipment.resources.thorlabs.kinesis.structs.QD_LAttribute), 588
DevModeDisabled	QD_P_PotentialGain
(msl.equipment.resources.nkt.nktpdll.DeviceStatusTypesAttribute), 163	(msl.equipment.resources.thorlabs.kinesis.structs.QD_PPotentialGainAttribute), 588

attribute), 589
differentialGain
(msl.equipment.resources.thorlabs.kinesis.structs.TCaItoIpPerAmeters
attribute), 593
digIO1OperMode
(msl.equipment.resources.thorlabs.kinesis.structs.FFmIOSetting
attribute), 581
digIO1PulseWidth
(msl.equipment.resources.thorlabs.kinesis.structs.FFmIOSetting
attribute), 581
digIO1SignalMode
(msl.equipment.resources.thorlabs.kinesis.structs.FFmIOSetting
attribute), 581
digIO2OperMode
(msl.equipment.resources.thorlabs.kinesis.structs.FFmIOSetting
attribute), 582
digIO2PulseWidth
(msl.equipment.resources.thorlabs.kinesis.structs.FFmIOSetting
attribute), 582
digIO2SignalMode
(msl.equipment.resources.thorlabs.kinesis.structs.FFmIOSetting
attribute), 582
digital (*msl.equipment.resources.picotech.picoscope.enums.PS5000ADigital*)
attribute), 359
digital (*msl.equipment.resources.picotech.picoscope.enums.PS2000ADigital*)
attribute), 359
digital (*msl.equipment.resources.picotech.picoscope.enums.PS3000ADigital*)
attribute), 363
digital (*msl.equipment.resources.picotech.picoscope.enums.PS3000ADigital*)
attribute), 362
DIGITAL_HARDWARE_VERSION
(msl.equipment.resources.picotech.picoscope.enums.PicScopeInfoApi
attribute), 233
Dim (*msl.equipment.resources.thorlabs.kinesis.enums.PPC_Dispatchability*)
attribute), 450
Dir_Disabled (*msl.equipment.resources.thorlabs.kinesis.enums.KIM_DirectionSense*)
attribute), 454
Dir_Forward (*msl.equipment.resources.thorlabs.kinesis.enums.KIM_DirectionSense*)
attribute), 455
Dir_Reverse (*msl.equipment.resources.thorlabs.kinesis.enums.KIM_DirectionSense*)
attribute), 455
direction (*msl.equipment.resources.picotech.picoscope.structs.PSD9000ADigitalChannelDirections*)
attribute), 360
direction (*msl.equipment.resources.picotech.picoscope.structs.PSD9000ADigitalChannelDirections*)
attribute), 363
direction (*msl.equipment.resources.picotech.picoscope.structs.PS4000ADirection*)
attribute), 365
direction (*msl.equipment.resources.thorlabs.kinesis.structs.MOT_HomingParameters*)
attribute), 572
directionSense
(msl.equipment.resources.thorlabs.kinesis.structs.MOT_HomingParameters)
(msl.equipment.resources.picotech.picoscope.picoscopy.enums.PT104_PT104)
method), 326

method), 372
disconnect() (*msl.equipment.resources.princeton_instrumentattribute*), 590
method), 374
displays_enable()
disconnect() (*msl.equipment.resources.thorlabs.fwxx2c.FilterWheelX2c*), 152
method), 595
display_4() (*msl.equipment.resources.mks_instruments.pr4000b.PR4000b*)
method), 152
DisplayTimeout
display_advanced_window() (*msl.equipment.resources.thorlabs.kinesis_structs.KMOT_Motors*)
(msl.equipment.resources.bentham.benhw32.BenthamAttribute), 583
method), 129
DisplayTimeout
display_range() (*msl.equipment.resources.thorlabs.kinesis_structs.KPZ_Motors*)
(msl.equipment.resources.greisinger.gmh3000.GMH3000Attribute), 587
method), 148
DisplayTimeout
display_setup_window() (*msl.equipment.resources.thorlabs.kinesis_structs.KSC_Motors*)
(msl.equipment.resources.bentham.benhw32.BenthamAttribute), 591
method), 129
DisplayTimeout
DisplayDimIntensity (*msl.equipment.resources.thorlabs.kinesis_structs.KMOT_ThermalParams*)
(attribute), 582
DISTANCE (*msl.equipment.resources.thorlabs.kinesis.enums.UnitType*)
attribute), 478
DisplayDimIntensity (*msl.equipment.resources.thorlabs.kinesis_structs.DISTRIBUTION*)
(attribute), 292
DISTRIBUTION (*msl.equipment.resources.picotech.picoscope.enums.DISTRIBUTION*)
(msl.equipment.resources.thorlabs.kinesis_structs.KSG_ThermalParams)
(attribute), 591
DISTRIBUTION (*msl.equipment.resources.picotech.picoscope.enums.DISTRIBUTION*)
(attribute), 303
DisplayDimIntensity (*msl.equipment.resources.thorlabs.kinesis_structs.DISTRIBUTION*)
(attribute), 592
do_averaging()
(msl.equipment.resources.thorlabs.kinesis_structs.KLDsManagement)
(attribute), 584
method), 207
displayIntensity (*msl.equipment.resources.thorlabs.kinesis_structs.KLSnMMIParams*)
(attribute), 585
method), 207
displayIntensity (*msl.equipment.resources.thorlabs.kinesis_structs.KMOT_ThermalParams*)
(attribute), 583
done() (*msl.equipment.resources.avantes.avaspec.Avantes*)
(method), 119
DisplayIntensity (*msl.equipment.resources.thorlabs.kinesis_structs.DONT_CARE_MMIParams*)
(attribute), 586
attribute), 249
DONT_CARE (*msl.equipment.resources.picotech.picoscope.enums.PS*)
(attribute), 249
DisplayIntensity (*msl.equipment.resources.thorlabs.kinesis_structs.KPZ_ThermalParams*)
(attribute), 587
DONT_CARE (*msl.equipment.resources.picotech.picoscope.enums.PS*)
(attribute), 239
DisplayIntensity (*msl.equipment.resources.thorlabs.kinesis_structs.DONT_CARE_MMIParams*)
(attribute), 591
attribute), 267
DONT_CARE (*msl.equipment.resources.picotech.picoscope.enums.PS*)
(attribute), 267
DisplayIntensity (*msl.equipment.resources.thorlabs.kinesis_structs.KSG_ThermalParams*)
(attribute), 592
DONT_CARE (*msl.equipment.resources.picotech.picoscope.enums.PS*)

attribute), 256

DONT_CARE (*msl.equipment.resources.picotech.picoscope.enums.PS2000AChDriverVersion*
attribute), 290

DRIVER_VERSION

DONT_CARE (*msl.equipment.resources.picotech.picoscope.enums.PS4000ATriggerState*
attribute), 278

DRIVER_VERSION

DONT_CARE (*msl.equipment.resources.picotech.picoscope.enums.PS4000ATriggerSources*
attribute), 254

DRIVER_VERSION

DONT_CARE (*msl.equipment.resources.picotech.picoscope.enums.PS3000ATriggerState*
attribute), 472

DRIVER_VERSION

DONT_CARE (*msl.equipment.resources.picotech.picoscope.enums.PS4000ATriggerSources*
attribute), 312

DRIVER_VERSION

DONT_CARE (*msl.equipment.resources.picotech.picoscope.enums.PS4000ATriggerState*
attribute), 472

DRIVER_VERSION

DONT_CARE (*msl.equipment.resources.picotech.picoscope.enums.PS6000ATriggerState*
attribute), 322

DRIVER_VERSION

DOWN (*msl.equipment.resources.picotech.picoscope.enums.PS2000ASweepType*
attribute), 245

DOWN (*msl.equipment.resources.picotech.picoscope.enums.PS3000ASweepType*
attribute), 237

DOWN (*msl.equipment.resources.picotech.picoscope.enums.PS4000ASweepType*
attribute), 263

DOWN (*msl.equipment.resources.picotech.picoscope.enums.PS4000ASweepType*
attribute), 285

DOWN (*msl.equipment.resources.picotech.picoscope.enums.PS4000ASweepType*
attribute), 274

DOWN (*msl.equipment.resources.picotech.picoscope.enums.PS5000ASweepType*
attribute), 301

DOWN (*msl.equipment.resources.picotech.picoscope.enums.PS6000ASweepType*
attribute), 299

DOWN (*msl.equipment.resources.picotech.picoscope.enums.PS6000ASweepType*
attribute), 320

DOWN (*msl.equipment.resources.picotech.picoscope.enums.PS6000ASweepType*
attribute), 318

DOWNLimit (*msl.equipment.resources.thorlabs.kinesis.structs.KNA_TIARangeParameters*
attribute), 586

downLimit (*msl.equipment.resources.thorlabs.kinesis.structs.NT_TIARangeParameters*
attribute), 578

E

DOWNUP (*msl.equipment.resources.picotech.picoscope.enums.PS4000ASweepType*
attribute), 245

E_MAX (*msl.equipment.resources.picotech.picoscope.enums.PS4000ASweepType*
attribute), 288

DOWNUP (*msl.equipment.resources.picotech.picoscope.enums.PS4000ASweepType*
attribute), 237

E_MIN (*msl.equipment.resources.picotech.picoscope.enums.PS4000ASweepType*
attribute), 282

DOWNUP (*msl.equipment.resources.picotech.picoscope.enums.PS4000ASweepType*
attribute), 263

EDGE_TRIGGER_SOURCE

DOWNUP (*msl.equipment.resources.picotech.picoscope.enums.PS4000ASweepType*
attribute), 285

Avantes

DOWNUP (*msl.equipment.resources.picotech.picoscope.enums.PS4000ASweepType*
attribute), 107

DOWNUP (*msl.equipment.resources.picotech.picoscope.enums.PS4000ASweepType*
attribute), 274

EF_10MHz4000ASweepType

DOWNUP (*msl.equipment.resources.picotech.picoscope.enums.PS4000ASweepType*
attribute), 308

EF_10MHz5000ASweepType

DOWNUP (*msl.equipment.resources.picotech.picoscope.enums.PS4000ASweepType*
attribute), 299

EF_10MHz5000ASweepType

DOWNUP (*msl.equipment.resources.picotech.picoscope.enums.PS4000ASweepType*
attribute), 318

EF_10MHz9600ASweepType

DRIVER_VERSION

(*msl.equipment.resources.picotech.picoscope.enums.PicotechPicoscopePulseSetupInfoApi*
attribute), 232

EF_MAX (*msl.equipment.resources.picotech.picoscope.enums.PS6000ASweepType*
attribute), 314

EF_OFF (*msl.equipment.resources.picotech.picoscope.enums.PS6000ASweepType*
attribute), 314

DRIVER_VERSION

```

EIGHT (msl.equipment.constants.DataBits at- property), 34
       tribute), 51 encoding_errors
enable_calibration_file() (msl.equipment.resources.optronic_laboratories.ol75600rib132)OL756
                           (method), 207 encoding_errors
enable_channel() (msl.equipment.resources.thorlabs.kinesis.benchtop_stepper)OL756
                  (method), 403
enable_channel() (msl.equipment.resources.thorlabs.kinesis.infrared_stereo_imaging)OL756
                  (method), 484
enable_channel() (msl.equipment.resources.thorlabs.kinesis.kcube_dc_method)KCubeDCServo
                  (method), 506
enable_channel() (msl.equipment.resources.thorlabs.kinesis.kcube_stepper)OL756
                  (method), 540
enable_dark_current() (msl.equipment.resources.optronic_laboratories.ol75600rib132)OL756
                      (method), 208
enable_last_msg_timer() (msl.equipment.resources.thorlabs.kinesis.benchtop_stereo_scope)OL756
                        (method), 404
enable_last_msg_timer() (msl.equipment.resources.thorlabs.kinesis.filter_flippert)OL756
                        (method), 482
enable_last_msg_timer() (msl.equipment.resources.thorlabs.kinesis.infrared_stereo_imaging)OL756
                        (method), 484
enable_last_msg_timer() (msl.equipment.resources.thorlabs.kinesis.kcube_dc_attribute)KCubeDCServo
                        (method), 506
enable_last_msg_timer() (msl.equipment.resources.thorlabs.kinesis.kcube_stepper)OL756
                        (method), 530
enable_last_msg_timer() (msl.equipment.resources.thorlabs.kinesis.kcube_stepper)OL756
                        (method), 540
enable_pmt_protection_mode() (msl.equipment.resources.optronic_laboratory)OL756
                           (method), 208
enable_sqr() (msl.equipment.connection_tcpip_vx)OL756
                           (method), 47
enable_standard_file() (msl.equipment.resources.optronic_laboratories.ol75600rib132)OL756
                           (method), 208
enabled(msl.equipment.resources.picotech.picoscope.channel)PicScope2Channel
                           (property), 231
encoding(msl.equipment.connection_message_based.ConnectionMessageBased)
                           (property), 29
encoding(msl.equipment.connection_prologix.ConnectionPrologix)
                           (tribute), 302

```

ENTER_OR_EXIT (*msl.equipment.resources.picotech.picoscope.enumerate_units*.~~enterOrExit~~^{ENTER_OR_EXIT} attribute), 321
enumerate_units() (in module *error* (*msl.equipment.hislip.AsyncStartTLSResponse* *msl.equipment.resources.picotech.picoscope.picoscope*.~~property~~^{property}), 70
325
enumerate_units() (in module *msl.equipment.resources.picotech.pt104*, *Error* (*msl.equipment.hislip.AuthenticationResult* *property*), 73
371
enums() (*msl.equipment.resources.utils.CHeader* method), 88
EQ99 (class in *msl.equipment.resources.energetiq.eq99*), 141
equipment (*msl.equipment.database.Database* property), 52
equipment_record (*msl.equipment.connection.Connection* property), 19
EquipmentRecord (class in *msl.equipment.record_types*), 79
errcheck() (*msl.equipment.resources.bentham.bentham*.~~method~~^{errcheck}), 130
errcheck_api() (*msl.equipment.resources.picotech.picoscope*.~~errcheck~~^{errcheck}), 331
errcheck_api() (*msl.equipment.resources.thorlabs.kinesis.motion_control*.~~errcheck~~^{errcheck}), 569
errcheck_code() (*msl.equipment.resources.thorlabs.fwxx2c.FilterWheel*.~~method~~^{errcheck}), 595
errcheck_negative() (*msl.equipment.resources.thorlabs.fwxx2c.FilterWheel*.~~method~~^{errcheck}), 595
errcheck_negative_one() (*msl.equipment.resources.picotech.picoscope.picoscaperiodik*.~~method~~^{errcheck}), 329
errcheck_non_zero() (*msl.equipment.resources.thorlabs.fwxx2c.FilterWheel*.~~method~~^{errcheck}), 595
errcheck_one() (*msl.equipment.resources.picotech.picoscope.picoscaperiodik*.~~method~~^{errcheck}), 329
errcheck_true() (*msl.equipment.resources.thorlabs.kinesis.motion_controller*.~~method~~^{errcheck}), 569
errcheck_zero() (*msl.equipment.resources.picotech.picoscope.picoscaperiodik*.~~method~~^{errcheck}), 329
Error, 59
error (*msl.equipment.hislip.AsyncEndTLSResponse* property), 71
error_CODE (*msl.equipment.resources.picotech.picoscope.enums.ERROR_CODE* attribute), 235
ERROR_CODE (*msl.equipment.resources.picotech.picoscope.enums.ERROR_CODE* attribute), 254
ERROR_CODES (*msl.equipment.resources.mks_instruments.pr4000b*.~~attribute~~^{attribute}), 151
error_to_english() (*msl.equipment.resources.princeton_instruments.arc_instrument*.~~static method~~^{static method}), 400
ErrorHandler (*msl.equipment.resources.nkt.nktpdll.ParameterSet*.~~attribute~~^{attribute}), 163
ErrorMessage (class in *msl.equipment.hislip*), 60
errcheck_stepopenapi (*msl.equipment.picoscope_nidaq.ConnectionNIDAQ* property), 32
ErrorType (class in *msl.equipment.hislip*), 58
ETH7010 (*msl.equipment.resources.avantes.avaspec.Avantes.InterfaceType* attribute), 108
ETH7010 (*msl.equipment.resources.avantes.avaspec.InterfaceType* attribute), 98
ETH_ALREADY_IN_USE_USB (*msl.equipment.resources.avantes.avaspec.Avantes.DeviceStatus* attribute), 108
ETH_ALREADY_IN_USE_USB (*msl.equipment.resources.avantes.avaspec.DeviceStatus* attribute), 98
ETH_AVAILABLE (*msl.equipment.resources.avantes.avaspec.Avante*.~~attribute~~^{attribute}), 108
ETH_AVAILABLE (*msl.equipment.resources.avantes.avaspec.DeviceStatus* attribute), 98
ETH_CONN_STATUS_CONNECTED (*msl.equipment.resources.avantes.avaspec.Avantes*.~~attribute~~^{attribute}), 108
ETH_CONN_STATUS_CONNECTED_NOMON (*msl.equipment.resources.avantes.avaspec.Avantes*.~~attribute~~^{attribute}), 108
ETH_CONN_STATUS_CONNECTING (*msl.equipment.resources.avantes.avaspec.Avantes*.~~attribute~~^{attribute}), 108
ETH_CONN_STATUS_NOCONNECTION (*msl.equipment.resources.avantes.avaspec.Avantes*.~~attribute~~^{attribute}), 108

ETH_IN_USE_BY_APPLICATION	attribute), 289 (msl.equipment.resources.avantes.avaspec. EXIT (byDeviceStatus .resources.picotech.picoscope.enums.PS4000A attribute), 108 attribute), 277
ETH_IN_USE_BY_APPLICATION	EXIT (msl.equipment.resources.picotech.picoscope.enums.PS5000A (msl.equipment.resources.avantes.avaspec.DeviceStatus attribute), 311 attribute), 302
ETH_IN_USE_BY_OTHER	EXIT (msl.equipment.resources.picotech.picoscope.enums.PS5000A (msl.equipment.resources.avantes.avaspec. EXIT (byDeviceStatus .resources.picotech.picoscope.enums.PS6000A attribute), 108 attribute), 321
ETH_IN_USE_BY_OTHER	export_config_file() (msl.equipment.resources.avantes.avaspec.DeviceStatus attribute), 208 method), 208
EthernetSettingsType	(class in export_registry() msl.equipment.resources.avantes.avaspec), (msl.equipment.resources.optronic_laboratories.ol756ocx method), 208 104
ETS (msl.equipment.resources.picotech.picoscope.enums.PS4000A) attribute), 296	EXT_PS4000APicosecondStringValue.picotech.picoscope.enums.PS2000AC attribute), 240
ETS_CYCLE (msl.equipment.resources.picotech.picoscope.enums.PS4000A) attribute), 296	EXT_PS4000APicosecondStringValue.picotech.picoscope.enums.PS2000AC attribute), 233
ETS_INTERLEAVE	EXT (msl.equipment.resources.picotech.picoscope.enums.PS3000AC (msl.equipment.resources.picotech.picoscope.enums.PS4000A)PicosecondStringValue attribute), 296 attribute), 258
ETS_SAMPLE_TIME_PICOSECONDS	EXT (msl.equipment.resources.picotech.picoscope.enums.PS4000A) (msl.equipment.resources.picotech.picoscope.enums.PS4000A)PicosecondStringValue.picotech.picoscope.enums.PS4000AC attribute), 296 attribute), 281
ETS_STATE (msl.equipment.resources.picotech.picoscope.enums.PS4000A) attribute), 296	EXT_PS4000APicosecondStringValue.picotech.picoscope.enums.PS4000AC attribute), 270
EVEN (msl.equipment.constants.Parity attribute), 50	EXT (msl.equipment.resources.picotech.picoscope.enums.PS5000AC attribute), 306
EVENT (msl.equipment.resources.picotech.picoscope.enums.PS5000C) attribute), 269	EXT_PS2000APicosecondStringValue.picotech.picoscope.enums.PS5000C attribute), 296
event_register()	EXT (msl.equipment.resources.picotech.picoscope.enums.PS6000C (msl.equipment.resources.energetiq.eq99.EQ99 attribute), 315 method), 143
event_status_register()	EXT_IN (msl.equipment.resources.picotech.picoscope.enums.PS2000A) (msl.equipment.resources.aim_tti.mx_series method), 91
excessEnergyLimit	EXT_IN (msl.equipment.resources.picotech.picoscope.enums.PS4000A) (msl.equipment.resources.thorlabs.kinesis.structs.MQFilterBranches attribute), 576
exclusive (msl.equipment.hislip.AsyncLockInfoResponse property), 64	EXT_IN (msl.equipment.resources.picotech.picoscope.enums.PS5000A) attribute), 276
EXIT (msl.equipment.resources.picotech.picoscope.enums.PS2000A) attribute), 249	EXT_IN (msl.equipment.resources.picotech.picoscope.enums.PS5000A) attribute), 249
EXIT (msl.equipment.resources.picotech.picoscope.enums.PS2000A) attribute), 238	EXT_IN (msl.equipment.resources.picotech.picoscope.enums.PS6000A) attribute), 238
EXIT (msl.equipment.resources.picotech.picoscope.enums.PS3000A) attribute), 266	EXT_MAX_VALUE (msl.equipment.resources.picotech.picoscope.ps2000A) attribute), 266
EXIT (msl.equipment.resources.picotech.picoscope.enums.PS3000A) attribute), 256	EXT_MAX_VALUE (msl.equipment.resources.picotech.picoscope.ps3000A) attribute), 256
EXIT (msl.equipment.resources.picotech.picoscope.enums.PS4000A) attribute), 244	EXT_MAX_VALUE (msl.equipment.resources.picotech.picoscope.ps4000A) attribute), 244

EXT_MAX_VALUE (*msl.equipment.resources.picotech.ps5000.PicoScope4000es.picotech.picoscope structs.PS4000AAttribute*), 350
attribute), 368

EXT_MAX_VALUE (*msl.equipment.resources.picotech.ps5000.PicoScope4000A.picotech.picoscope structs.PS4000AAttribute*), 351
attribute), 367

EXT_MAX_VALUE (*msl.equipment.resources.picotech.ps5000.PicoScope5000es.picotech.picoscope structs.PS5000AAttribute*), 353
attribute), 367

EXT_MAX_VALUE (*msl.equipment.resources.picotech.ps5000.PicoScope5000A.picotech.picoscope structs.PS5000AAttribute*), 354
attribute), 370

EXT_MAX_VOLTAGE
(msl.equipment.resources.picotech.picoscope.ps5000Attribute), 350
attribute), 355
external (*msl.equipment.resources.picotech.picoscope.structs.PS5000A*
external_input())

EXT_MIN_VALUE (*msl.equipment.resources.picotech.picoscope.ps2000.ipPicoscope2000anks_instruments.pr4000b.PR4000bAttribute*), 344
attribute), 344
method), 152

EXT_MIN_VALUE (*msl.equipment.resources.picotech.picoscope.pr4000.PicoScope3000A*
attribute), 347
EXTERNAL_TRIGGER (*RicoScope3000A*
(msl.equipment.resources.avantes.avaspec.Avantes

EXT_MIN_VALUE (*msl.equipment.resources.picotech.picoscope.pr4000.PicoScope4000*
attribute), 350
ExtSignalsMA (*msl.equipment.resources.thorlabs.kinesis.enums.HD4000Attribute*))

EXT_MIN_VALUE (*msl.equipment.resources.picotech.picoscope.pr4000.PicoScope4000A*
attribute), 352

EXT_MIN_VALUE (*msl.equipment.resources.picotech.picoscope.ps5000.PicoScope5000*
attribute), 353
F (*msl.equipment.resources.picotech.picoscope.enums.PS4000ACh*)

EXT_MIN_VALUE (*msl.equipment.resources.picotech.picoscope.pr5000.PicoScope5000A*
attribute), 354
F_MAX (*msl.equipment.resources.picotech.picoscope.enums.PS4000A*)

ExtBNC (*msl.equipment.resources.thorlabs.kinesis.enums.PPCat10Data*), 351
attribute), 448
F_MIN (*msl.equipment.resources.picotech.picoscope.enums.PS4000A*)

external (*msl.equipment.resources.picotech.picoscope.structs.PS2000AFWqConditions*
attribute), 359
factorForOutput

external (*msl.equipment.resources.picotech.picoscope.structs.PS2000ATriggerConditions*), 359
attribute), 574

external (*msl.equipment.resources.picotech.picoscope.FactoryValPS2000PmiCondition*), 359
attribute), 163

external (*msl.equipment.resources.picotech.picoscope.FactoryValPS2000PreqCondition*), 359
attribute), 236

external (*msl.equipment.resources.picotech.picoscope.FactoryValPS3000APreqCondition*), 362
attribute), 255

external (*msl.equipment.resources.picotech.picoscope.FactoryValPS3000PmiCondition*), 363
attribute), 63

external (*msl.equipment.resources.picotech.picoscope.FactoryValPS3000ATriggerCondition*), 362
attribute), 249

external (*msl.equipment.resources.picotech.picoscope.FactoryValPS3000V2Condition*), 362
attribute), 247

external (*msl.equipment.resources.picotech.picoscope.FactoryValPS3000PreqCondition*), 361
attribute), 249

external (*msl.equipment.resources.picotech.picoscope.FactoryValPS3000PmiCondition*), 361
attribute), 235

external (*msl.equipment.resources.picotech.picoscope.FactoryValPS4000PreqCondition*), 365
attribute), 267

external (*msl.equipment.resources.picotech.picoscope.FactoryValPS4000PmiCondition*), 364
attribute), 264

external (*msl.equipment.resources.picotech.picoscope.FactoryValPS5000PreqCondition*), 369
attribute), 266

FALLING (*msl.equipment.resources.picotech.picoscope.enums.PS3000AThresholdDirection*
attribute), 256 FALSE (*msl.equipment.resources.picotech.picoscope.enums.PS6000A*
attribute), 252

FALLING (*msl.equipment.resources.picotech.picoscope.enums.PS3000ATriggerDirection*
attribute), 254 FAST (*msl.equipment.resources.picotech.picoscope.enums.PS2000A*
attribute), 252

FALLING (*msl.equipment.resources.picotech.picoscope.enums.PS4000ASigGenTrigType*
attribute), 287 FAST (*msl.equipment.resources.picotech.picoscope.enums.PS2000A*
attribute), 254

FALLING (*msl.equipment.resources.picotech.picoscope.enums.PS4000ATHresholdDirection*
attribute), 289 FAST (*msl.equipment.resources.picotech.picoscope.enums.PS3000A*
attribute), 256

FALLING (*msl.equipment.resources.picotech.picoscope.enums.PS4000ASigGenTrigType*
attribute), 275 FAST (*msl.equipment.resources.picotech.picoscope.enums.PS3000A*
attribute), 257

FALLING (*msl.equipment.resources.picotech.picoscope.enums.PS4000ATHresholdDirection*
attribute), 277 FAST (*msl.equipment.resources.picotech.picoscope.enums.PS4000A*
attribute), 259

FALLING (*msl.equipment.resources.picotech.picoscope.enums.PS5000ASigGenTrigType*
attribute), 309 FAST (*msl.equipment.resources.picotech.picoscope.enums.PS4000A*
attribute), 259

FALLING (*msl.equipment.resources.picotech.picoscope.enums.PS5000ATHresholdDirection*
attribute), 311 FAST (*msl.equipment.resources.picotech.picoscope.enums.PS5000A*
attribute), 259

FALLING (*msl.equipment.resources.picotech.picoscope.enums.PS5000ASigGenTrigType*
attribute), 300 FAST (*msl.equipment.resources.picotech.picoscope.enums.PS5000A*
attribute), 259

FALLING (*msl.equipment.resources.picotech.picoscope.enums.PS5000ATHresholdDirection*
attribute), 302 FAST (*msl.equipment.resources.picotech.picoscope.enums.PS6000A*
attribute), 259

FALLING (*msl.equipment.resources.picotech.picoscope.enums.PS6000ASigGenTrigType*
attribute), 319 FAST (*msl.equipment.resources.thorlabs.fwxx2c.SpeedMode*
attribute), 259

FALLING (*msl.equipment.resources.picotech.picoscope.enums.PS6000ATHresholdDirection*
attribute), 321 Fatal Error, 59

FALLING_LOWER (*msl.equipment.resources.picotech.picoscope.enums.PS2000ATHresholdDirection*
attribute), 249 Fatal Error (*msl.equipment.hislip*), 57

FALLING_LOWER (*msl.equipment.resources.picotech.picoscope.enums.PS3000ATHresholdDirection* in
attribute), 266 *msl.equipment.hislip*), 60

FALLING_LOWER (*msl.equipment.resources.picotech.picoscope.enums.PS4000ATHresholdDirection* in
attribute), 289 *msl.equipment.hislip*), 454

FALLING_LOWER (*msl.equipment.resources.picotech.picoscope.enums.PS4000ATHresholdDirection*
attribute), 277 (*msl.equipment.thorlabs.kinesis.enums.KIM_F*
attribute), 259

FALLING_LOWER (*msl.equipment.resources.picotech.picoscope.enums.PS5000ATHresholdDirection*
attribute), 311 FC_20 (*msl.equipment.resources.picotech.picoscope.enums.PS4000A*
attribute), 259

FALLING_LOWER (*msl.equipment.resources.picotech.picoscope.enums.PS6000ATHresholdDirection*
attribute), 321 FC_20 (*msl.equipment.resources.picotech.picoscope.enums.PS4000A*
attribute), 259

FALSE (*msl.equipment.resources.picotech.picoscope.enums.PS2000ATrig200State*
attribute), 250 FC_200 (*msl.equipment.resources.picotech.picoscope.enums.PS4000A*
attribute), 259

FALSE (*msl.equipment.resources.picotech.picoscope.enums.PS2000ATrig200State*
attribute), 239 FC_200 (*msl.equipment.resources.picotech.picoscope.enums.PS4000A*
attribute), 259

FALSE (*msl.equipment.resources.picotech.picoscope.enums.PS2000ATrig200State*
attribute), 267 FC_20K (*msl.equipment.resources.picotech.picoscope.enums.PS4000A*
attribute), 259

FALSE (*msl.equipment.resources.picotech.picoscope.enums.PS2000ATrig200State*
attribute), 257 FC_20K (*msl.equipment.resources.picotech.picoscope.enums.PS4000A*
attribute), 259

FALSE (*msl.equipment.resources.picotech.picoscope.enums.PS4000ATrig200State*
attribute), 290 FC_2K (*msl.equipment.resources.picotech.picoscope.enums.PS4000A*
attribute), 259

FALSE (*msl.equipment.resources.picotech.picoscope.enums.PS4000ATrig200State*
attribute), 278 FC_2K (*msl.equipment.resources.picotech.picoscope.enums.PS4000A*
attribute), 259

FALSE (*msl.equipment.resources.picotech.picoscope.enums.PS5000ATrig200State*
attribute), 312 FC_MAX (*msl.equipment.resources.picotech.picoscope.enums.PS4000A*
attribute), 259

FALSE (*msl.equipment.resources.picotech.picoscope.enums.PS5000ATrig200State*

FC_MAX (*msl.equipment.resources.picotech.picoscope.FETLogistics*.~~counterRange~~
attribute), 280
feature_bitmap (*msl.equipment.hislip.AsyncDeviceClearAck*.~~filter1Fc~~
property), 65
feedbackSignal (*msl.equipment.resources.thorlabs.kinesis.structs.BNTHouseSettings*.
attribute), 577
feedbackSrc (*msl.equipment.resources.thorlabs.kinesis.structs.PPC_IOMode*).
attribute), 581
feedForward (*msl.equipment.resources.thorlabs.kinesis.structs.PPC_IOMode*).
attribute), 575
FF_InputButton (*msl.equipment.resources.thorlabs.kinesis.motion_control*.
enums.FF_SignalModes.
attribute), 451
FF_InputLogic (*msl.equipment.resources.thorlabs.kinesis.enums.FF_SignalModes*.
attribute), 451
FF_InputSwap (*msl.equipment.resources.thorlabs.kinesis.enums.FF_SignalModes*.
attribute), 451
FF_IOModes (class in *msl.equipment.resources.thorlabs.fwxx2c*),
msl.equipment.resources.thorlabs.kinesis.enums), 593
450
FF_IOSettings (class in *msl.equipment.resources.thorlabs.kinesis.filter_flipper*),
msl.equipment.resources.thorlabs.kinesis structs), 478
581
FF_OutputHighAtSetPosition (*msl.equipment.resources.thorlabs.kinesis.enums.FFTimerMode*).
attribute), 450
FF_OutputHighWhenMoving (*msl.equipment.resources.thorlabs.kinesis.enums.FFTimerMode*.
config.Config method), 18
attribute), 450
FF_OutputLevel (*msl.equipment.resources.thorlabs.kinesis.enums.FFTimerMode* in module *msl.equipment*),
attribute), 451
17
FF_OutputPulse (*msl.equipment.resources.thorlabs.kinesis.enums.FFTimerMode*.factory), 56
attribute), 451
FF_OutputSwap (*msl.equipment.resources.thorlabs.kinesis.enums.FFTimerMode*.
gpiB), 22
attribute), 451
FF_PositionError (*msl.equipment.resources.thorlabs.kinesis.enums.FF5Positions*.
attribute), 450
FF_Positions (class in *msl.equipment.connection_prologix*),
msl.equipment.resources.thorlabs.kinesis.enums), 36
450
FF_SetPositionOnPositiveEdge (*msl.equipment.resources.thorlabs.kinesis.enums.FFTimerMode* in module *msl.equipment.vxi11*),
attribute), 450
609
FF_SignalModes (class in *findall()* (*msl.equipment.config.Config*
msl.equipment.resources.thorlabs.kinesis.enums), *method*), 19
450
firmware_version()

(*msl.equipment.resources.greisinger.gmh3000.GMH3000*
method), 148 FS (*msl.equipment.resources.picotech.picoscope.enums.PS2000ATime*
FIRMWARE_VERSION_1 attribute), 245
(*msl.equipment.resources.picotech.picoscope.PS2000ATime*
attribute), 234
FIRMWARE_VERSION_2 FS (*msl.equipment.resources.picotech.picoscope.enums.PS3000ATime*
(*msl.equipment.resources.picotech.picoscope.enums.RetroSetup*
attribute), 233 FS (*msl.equipment.resources.picotech.picoscope.enums.PS3000ATime*
attribute), 253
firmwareVersion FS (*msl.equipment.resources.thorlabs.kinesis.FT_DeviceHardwareInfo*
attribute), 571 FS (*msl.equipment.resources.picotech.picoscope.enums.PS4000ATime*
attribute), 284
FIRST_USB FS (*msl.equipment.resources.picotech.picoscope.PS4000ATime*
attribute), 342 FS (*msl.equipment.resources.picotech.picoscope.PS4000ATime*
attribute), 273
FIRST_USB FS (*msl.equipment.resources.picotech.picoscope.PS5000ATime*
attribute), 345 FS (*msl.equipment.resources.picotech.picoscope.PS5000ATime*
attribute), 308
FIVE (*msl.equipment.constants.DataBits* at- FS (*msl.equipment.resources.picotech.picoscope.enums.PS5000ATime*
tribute), 51 attribute), 298
flash_led() (*msl.equipment.resources.picotech.picoscope.PicoScope2k3*
method), 329 FS (*msl.equipment.resources.picotech.picoscope.PicoScope2k3*
attribute), 317
flash_led() (*msl.equipment.resources.picotech.picoscope.PicoScope2k3*
method), 332 FS (*msl.equipment.resources.thorlabs.kinesis.enums.FT_Status*
attribute), 432
forceCalibration FS (*msl.equipment.resources.thorlabs.kinesis.FT_DeviceNotPresent*
attribute), 592 FS (*msl.equipment.resources.thorlabs.kinesis.enums.FT_Status*
attribute), 432
Forward (*msl.equipment.resources.thorlabs.kinesis.enums.KIMDirection*
attribute), 453 FT_DeviceNotPresent
Forward (*msl.equipment.resources.thorlabs.kinesis.enums.TIMDirection*
attribute), 475 FT_DeviceNotPresent
Forwards (*msl.equipment.resources.thorlabs.kinesis.FT_DeviceDirections*
attribute), 438 FT_DeviceNotPresent
FPBrightness (*msl.equipment.resources.thorlabs.kinesis.structs.DPIEIOSettings*
attribute), 581 FT_InsufficientResources
FREQUENCY_COUNTER (*msl.equipment.resources.picotech.picoscope.enums.PS4000APicoStringValue*
attribute), 294 FT_InvalidHandle
FREQUENCY_COUNTER_CHANNEL (*msl.equipment.resources.picotech.picoscope.enums.PS4000APicoStringValue*
attribute), 294 FT_InvalidHandle
FREQUENCY_COUNTER_ENABLED (*msl.equipment.resources.picotech.picoscope.enums.PS4000APicoStringValue*
attribute), 294 FT_IOError
FREQUENCY_COUNTER_RANGE (*msl.equipment.resources.picotech.picoscope.FT_Error*
attribute), 294 FT_Error
FREQUENCY_COUNTER_THRESHOLDMAJOR FT_Status (class in
(*msl.equipment.resources.picotech.picoscope.enums.PS4000APicoStringValue*
attribute), 294 431
FREQUENCY_COUNTER_THRESHOLDMINOR functions() (*msl.equipment.resources.utils.CHeader*
(*msl.equipment.resources.picotech.picoscope.enums.PS4000APicoStringValue*
attribute), 294 futureUse (*msl.equipment.resources.thorlabs.kinesis.structs.TSG_*
from_bytes() (in module *msl.equipment.utils*), attribute), 592

FW103 (*msl.equipment.resources.thorlabs.kinesis.enums.KST_S property*), 38
 attribute), 473
 GAUSSIAN (*msl.equipment.resources.picotech.picoscope.enums.PS2000ASigGenTrigType*)
 FW103 (*msl.equipment.resources.thorlabs.kinesis.enums.TST_S property*), 246
 attribute), 477
 GAUSSIAN (*msl.equipment.resources.picotech.picoscope.enums.PS2000ASigGenTrigType*)
 FW_FAIL (*msl.equipment.resources.picotech.errors.PS2000Error attribute*), 237
 attribute), 228
 GAUSSIAN (*msl.equipment.resources.picotech.picoscope.enums.PS3000ASigGenTrigType*)
 FW_FAIL (*msl.equipment.resources.picotech.errors.PS3000Error attribute*), 264
 attribute), 228
 GAUSSIAN (*msl.equipment.resources.picotech.picoscope.enums.PS4000ASigGenTrigType*)
G
 G (*msl.equipment.resources.picotech.picoscope.enums.PS4000ASigGenTrigType*), 275
 attribute), 281
 GAUSSIAN (*msl.equipment.resources.picotech.picoscope.enums.PS4000ASigGenTrigType*)
 G_MAX (*msl.equipment.resources.picotech.picoscope.enums.PS4000ASigGenTrigType*), 282
 attribute), 282
 GAUSSIAN (*msl.equipment.resources.picotech.picoscope.enums.PS4000ASigGenTrigType*)
 G_MIN (*msl.equipment.resources.picotech.picoscope.enums.PS4000ASigGenTrigType*), 282
 attribute), 282
 GAUSSIAN (*msl.equipment.resources.picotech.picoscope.enums.PS4000ASigGenTrigType*)
 GAIN (*msl.equipment.resources.cmi.sia3.SIA3 attribute*), 132
 attribute), 318
 GAUSSIAN_MAX_FREQUENCY
 gain (*msl.equipment.resources.thorlabs.kinesis structs.NT_Gain*), 579
 attribute), 345
 GARBLE_ARGS (*msl.equipment.vxi11.AcceptStatus attribute*), 602
 attribute), 602
 GATE_HIGH (*msl.equipment.resources.picotech.picoscope.enums.PS2000ASigGenTrigType*), 247
 attribute), 247
 GAUSSIAN_MAX_FREQUENCY
 GATE_HIGH (*msl.equipment.resources.picotech.picoscope.enums.PS3000ASigGenTrigType*), 264
 attribute), 264
 GAUSSIAN_MAX_FREQUENCY
 GATE_HIGH (*msl.equipment.resources.picotech.picoscope.enums.PS4000ASigGenTrigType*), 287
 attribute), 287
 GAUSSIAN_MAX_FREQUENCY
 GATE_HIGH (*msl.equipment.resources.picotech.picoscope.enums.PS5000ASigGenTrigType*), 309
 attribute), 309
 GAUSSIAN_MAX_FREQUENCY
 GATE_HIGH (*msl.equipment.resources.picotech.picoscope.enums.PS6000ASigGenTrigType*), 300
 attribute), 300
 GAUSSIAN_MAX_FREQUENCY
 GATE_HIGH (*msl.equipment.resources.picotech.picoscope.enums.PS6000ASigGenTrigType*), 319
 attribute), 319
 GD_Trig_High (*msl.equipment.resources.thorlabs.kinesis.enums.QD*)
 GATE_LOW (*msl.equipment.resources.picotech.picoscope.enums.PS2000ASigGenTrigType*), 247
 attribute), 247
 GD_Trig_Low (*msl.equipment.resources.thorlabs.kinesis.enums.QD*)
 GATE_LOW (*msl.equipment.resources.picotech.picoscope.enums.PS2000ASigGenTrigType*), 264
 attribute), 264
 GenericDCMotor (in module
 GATE_LOW (*msl.equipment.resources.picotech.picoscope.enums.PS4000ASigGenTrigType*), 288
 attribute), 288
 565
 GenericDCMotor (in module
 GATE_LOW (*msl.equipment.resources.picotech.picoscope.enums.PS4000ASigGenTrigType*), 275
 attribute), 275
 msl.equipment.resources.thorlabs.kinesis.messages), 565
 GenericMotor (in module
 GATE_LOW (*msl.equipment.resources.picotech.picoscope.enums.PS5000ASigGenTrigType*), 309
 attribute), 309
 GenericMotor (in module
 GATE_LOW (*msl.equipment.resources.picotech.picoscope.enums.PS5000ASigGenTrigType*), 300
 attribute), 300
 565
 GenericMotor (in module
 GATE_LOW (*msl.equipment.resources.picotech.picoscope.enums.PS6000ASigGenTrigType*), 319
 attribute), 319
 msl.equipment.resources.thorlabs.kinesis.messages), 565
 gateway (*msl.equipment.connection_sdk.ConnectionSDK*), 565

GenericSimpleMotor (in module `msl.equipment.resources.thorlabs.kinesis.motor`)
 get_bow_index() (method), 485
 566
 get() (method), 128
 (msl.equipment.resources.bentham.benhw32.Bentham3D)
 get_brightness()
 get() (method), 130
 (msl.equipment.resources.bentham.benhw64.Bentham)
 (msl.equipment.resources.energetiq.eq99.EQ99)
 method), 142
 get_acceleration() (method), 595
 (msl.equipment.resources.thorlabs.fwxx2c.FilterWheelXX2C)
 get_buffer() (msl.equipment.vxi11.RPCClient)
 (msl.equipment.resources.thorlabs.kinesis.integrated_step)
 get_button_params()
 get_access_channel() (method), 152
 (msl.equipment.resources.mks_instruments.pr4000b.PR4000B)
 get_button_params_block()
 get_actual_value() (method), 153
 (msl.equipment.resources.mks_instruments.pr4000b.PR4000B)
 get_c_group() (msl.equipment.resources.bentham.benhw32.Bentham)
 method), 128
 get_adaptive_int_time_index() (method), 209
 (msl.equipment.resources.optronic_laboratories.ol756ocx)
 get_array() (OL756)
 (msl.equipment.resources.optronic_laboratories.ol756ocx)
 get_cal_file_enabled()
 get_address() (method), 153
 (msl.equipment.resources.mks_instruments.pr4000b.PR4000B)
 get_cal_file_enabled()
 get_alarm() (method), 137
 (msl.equipment.resources.electron_dynamics.tc)
 (but this is a static method of msl.equipment.resources.optronic_laboratories.ol756ocx)
 method), 210
 get_all_ports() (method), 177
 (msl.equipment.resources.nkt.nktpdll.NKT)
 (static method), 210
 get_analog_in() (method), 116
 (msl.equipment.resources.avantes.avaspec.Avantes)
 (msl.equipment.resources.optronic_laboratories.ol756ocx)
 method), 211
 get_analogue_offset() (method), 332
 (msl.equipment.resources.picotech.picoscope.picoscope)
 (msl.equipment.resources.thorlabs.kinesis.benchtop_stepper)
 method), 404
 get_backlash() (method), 404
 (msl.equipment.resources.thorlabs.kinesis.benchtop_stepper)
 (but this is a static method of msl.equipment.resources.thorlabs.kinesis.integrated_step)
 method), 485
 get_backlash() (method), 484
 (msl.equipment.resources.thorlabs.kinesis.integrated_step)
 (but this is a static method of msl.equipment.resources.thorlabs.kinesis.integrated_step)
 method), 540
 get_backlash() (method), 506
 (msl.equipment.resources.thorlabs.kinesis.kcube_dc)
 (msl.equipment.resources.picotech.picoscope.picoscope)
 method), 332
 get_backlash() (method), 540
 (msl.equipment.resources.thorlabs.kinesis.kcube_step)
 (msl.equipment.resources.thorlabs.kinesis.integrated_step)
 method), 211
 get_beep() (method), 142
 (msl.equipment.resources.energetiq.eq99.EQ99)
 get_port_name()
 (msl.equipment.resources.avantes.avaspec.Avantes)
 method), 117
 get_bow_index() (method), 404
 (msl.equipment.resources.thorlabs.kinesis.benchtop_sterpe)
 (motor.BenchtopStepperMotor)
 (msl.equipment.resources.avantes.avaspec.Avantes)
 method), 117
 get_bow_index() (method), 485
 (msl.equipment.resources.thorlabs.kinesis.integrated_step)
 (motor.BenchtopStepperMotor)
 (msl.equipment.resources.thorlabs.kinesis.integrated_step)
 method), 540

```

(msl.equipment.resources.picotech.picoscope.get_4000a_hi_polar)(4000A
    method), 352
(msl.equipment.hislip.HiSLIPClient
    method), 74
get_component_list()
    (msl.equipment.resources.bentham.benhw32BenthamBipolar()
        method), 128
    (msl.equipment.resources.princeton_instruments.arc_instrument
        method), 380
get_component_list()
    (msl.equipment.resources.bentham.benhw64BenthamBipolar_str()
        method), 130
    (msl.equipment.resources.princeton_instruments.arc_instrument
        method), 380
get_control() (msl.equipment.resources.electron_dynamic.get_det_MXSeries()
    method), 137
    get_det_hv_on()
get_cri() (msl.equipment.resources.optronic_laboratories.ol75series.get_det_MXSeries()
    method), 209
    get_det_hv_on()
get_current() (msl.equipment.resources.aim_tti.get_det_MXSeries()
    method), 91
    (msl.equipment.resources.princeton_instruments.arc_instrument
        method), 380
get_current() (msl.equipment.resources.optronic_laboratories.ol75series.get_det_MXSeries()
    method), 225
    get_det_num_avg_read()
get_current_limit()
    (msl.equipment.resources.aim_tti.mx_series.MXSeries()
        method), 381
    get_det_range()
get_current_step_size()
    (msl.equipment.resources.aim_tti.mx_series.MXSeries()
        method), 381
    get_det_range_factor()
get_cycle_params()
    (msl.equipment.resources.thorlabs.kinesis.kcube_solenoid.get_det_KCubeSolenoid()
        method), 530
    get_det_type()
get_cycle_params_block()
    (msl.equipment.resources.thorlabs.kinesis.kcube_solenoid.get_det_KCubeSolenoid()
        method), 531
    get_det_type_str()
get_dark_current()
    (msl.equipment.resources.optronic_laboratories.ol75series.get_device_info()
        method), 212
    get_device_info()
get_dark_current_enable()
    (msl.equipment.resources.optronic_laboratories.ol75series.get_device_list()
        method), 212
    get_device_list()
get_dark_current_mode()
    (msl.equipment.resources.optronic_laboratories.ol75series.get_device_list_size()
        method), 212
    get_device_list_size()
get_dark_current_wavelength()
    (msl.equipment.resources.optronic_laboratories.ol75series.get_device_resolution()
        method), 212
    get_device_resolution()
get_dark_pixel_data()
    (msl.equipment.resources.avantes.avaspec.Avantes
        method), 355
    get_device_unit_from_real_value()
get_data() (msl.equipment.resources.avantes.avaspec.Avantes
    method), 120
    (msl.equipment.resources.thorlabs.kinesis.benchtop_stepper
        method), 404
get_dcpid_params()
    (msl.equipment.resources.thorlabs.kinesis.kcube_dc.get_device_KCpuID()
        method), 507
    get_device_unit_from_real_value()
get_dead_band()
    (msl.equipment.resources.mks_instruments.pr4000b.PR4000B
        method), 153
    get_device_unit_from_real_value()

```

get_device_unit_from_real_value() (msl.equipment.resources.energetiq.eq99.EQ99
 (msl.equipment.resources.thorlabs.kinesis.kcube stepper~~method~~) or KCubeStepperMotor
 method), 540 get_external_input_range()
get_dialog() (msl.equipment.resources.mks_instruments.pr4000b.PR4000B resources.mks_instruments.pr4000b.PR4000B
 method), 153 method), 153
get_digital_in() get_external_output_range()
 (msl.equipment.resources.avantes.avaspec.Avantes (msl.equipment.resources.mks_instruments.pr4000b.PR4000B
 method), 118 method), 153
get_digital_outputs() get_filter_max_pos()
 (msl.equipment.resources.thorlabs.kinesis.benchtop_kcube_dc_gearstepper~~method~~ or KCubeStepperMotor instruments.arc_instrument
 method), 405 method), 382
get_digital_outputs() get_filter_min_pos()
 (msl.equipment.resources.thorlabs.kinesis.kcube_dc_gearstepper~~method~~ or KCubeStepperMotor instruments.arc_instrument
 method), 507 method), 382
get_digital_outputs() get_filter_model()
 (msl.equipment.resources.thorlabs.kinesis.kcube_sole~~method~~ or KCubeStepperMotor instruments.arc_instrument
 method), 531 method), 382
get_digital_outputs() get_filter_position()
 (msl.equipment.resources.thorlabs.kinesis.kcube_step~~method~~ or KCubeStepperMotor instruments.arc_instrument
 method), 541 method), 382
get_display_text() get_filter_preopen_model()
 (msl.equipment.resources.mks_instruments.pr4000b.PR4000B pr4000b.PR4000B resources.princeton_instruments.arc_instrument
 method), 153 static method), 382
get_dll_version() get_filter_present()
 (msl.equipment.resources.avantes.avaspec.Avantes (msl.equipment.resources.princeton_instruments.arc_instrument
 method), 118 method), 382
get_encoder_counter() get_filter_serial()
 (msl.equipment.resources.thorlabs.kinesis.benchtop_kcube_dc_gearstepper~~method~~ or KCubeStepperMotor instruments.arc_instrument
 method), 405 method), 382
get_encoder_counter() get_firmware_version()
 (msl.equipment.resources.thorlabs.kinesis.kcube_dc_gearstepper~~method~~ or KCubeStepperMotor instruments.thorlabs.kinesis.benchtop_stepper
 method), 507 method), 405
get_encoder_counter() get_firmware_version()
 (msl.equipment.resources.thorlabs.kinesis.kcube_step~~method~~ or KCubeStepperMotor instruments.thorlabs.kinesis.filter_flipper.Filter
 method), 541 method), 479
get_ending_wavelength() get_firmware_version()
 (msl.equipment.resources.optronic_laboratories.ol75~~method~~ or KCubeStepperMotor instruments.thorlabs.kinesis.integrated_step
 method), 212 method), 486
get_enum_preopen_com() get_formula_relay()
 (msl.equipment.resources.princeton_instruments.arc_instrument or KCubeStepperMotor instruments.pr4000b.PR4000B
 static method), 383 method), 153
get_enum_preopen_model() get_formula_temporary()
 (msl.equipment.resources.princeton_instruments.arc_instrument or KCubeStepperMotor instruments.pr4000b.PR4000B
 static method), 382 method), 154
get_enum_preopen_serial() get_front_panel_locked()
 (msl.equipment.resources.princeton_instruments.arc_instrument or KCubeStepperMotor instruments.thorlabs.kinesis.kcube_dc_server
 static method), 383 method), 507
get_exposure_mode() get_front_panel_locked()
 (msl.equipment.resources.energetiq.eq99.EQ99 (msl.equipment.resources.thorlabs.kinesis.kcube stepper
 method), 144 method), 541
get_exposure_time() get_gain() (msl.equipment.resources.mks_instruments.pr4000b.PR4000B
 method), 153

```

        method), 154
get_gain_index()                               get_homing_params_block()
                                              (msl.equipment.resources.thorlabs.kinesis.integrated_step_
                                              (msl.equipment.resources.optronic_laboratories.ol756ocx.method), 508
                                              method), 212
                                              get_homing_params_block()
get_group() (msl.equipment.resources.bentham.benhw32.Bentham32.method), 128
                                              get_homing_params_block()
                                              method), 508
get_handle_from_serial()                      get_homing_params_block()
                                              (msl.equipment.resources.avantes.avaspec.Avantes.method), 118
                                              get_homing_params_block()
get_hardware_info()                           get_homing_velocity()
                                              (msl.equipment.resources.thorlabs.kinesis.benchtop_stepper.method), 405
                                              get_homing_velocity()
                                              method), 406
get_hardware_info()                           get_homing_velocity()
                                              (msl.equipment.resources.thorlabs.kinesis.filter_flipper.FilterFlipper.method), 479
                                              get_homing_velocity()
                                              method), 486
get_hardware_info()                           get_homing_velocity()
                                              (msl.equipment.resources.thorlabs.kinesis.integrated_stepper_motors.IntegratedStepperMotors.kcube_dc_servo.method), 486
                                              get_homing_velocity()
                                              method), 508
get_hardware_info()                           get_homing_velocity()
                                              (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo.KCubeDCServo.step_and_hold.method), 507
                                              get_homing_velocity()
                                              method), 541
get_hardware_info()                           get_hub_bay()
                                              (msl.equipment.resources.thorlabs.kinesis.kcube_solenoid.KCubeSolenoid.method), 531
                                              get_hub_bay()
                                              method), 531
get_hardware_info()                           get_hub_bay()
                                              (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo.KCubeDCServo.step_and_hold.method), 531
                                              get_hub_bay()
                                              method), 541
get_hardware_info()                           get_id()
                                              (msl.equipment.resources.thorlabs.fwxx2c.FilterWheelX2C.method), 505
                                              get_id()
                                              method), 542
get_hardware_info_block()                   get_increment()
                                              (msl.equipment.resources.optronic_laboratories.ol756ocx.method), 405
                                              get_increment()
                                              method), 405
get_hardware_info_block()                   get_increment_index()
                                              (msl.equipment.resources.thorlabs.kinesis.integrated_stepper_motors.IntegratedStepperMotors.method), 486
                                              get_increment_index()
                                              method), 508
get_hardware_info_block()                   get_input_range()
                                              (msl.equipment.resources.optronic_laboratories.ol756ocx.method), 508
                                              get_input_range()
                                              method), 508
get_hardware_info_block()                   get_input_status()
                                              (msl.equipment.resources.thorlabs.kinesis.kcube_solenoid.KCubeSolenoid.method), 531
                                              get_input_status()
                                              method), 531
get_hardware_info_block()                   get_input_voltage()
                                              (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo.KCubeDCServo.step_and_hold.method), 541
                                              get_input_voltage()
                                              method), 541
get_hardware_type()                          get_integration_time_index()
                                              (msl.equipment.resources.bentham.benhw32.Bentham32.method), 128
                                              get_integration_time_index()
                                              method), 406
get_hardware_type()                          get_interface_mode()
                                              (msl.equipment.resources.bentham.benhw64.Bentham64.method), 130
                                              get_interface_mode()
                                              method), 213
get_homing_params_block()                  get_io_settings()
                                              (msl.equipment.resources.thorlabs.kinesis.benchtop_stepper_motor.BenchtopStepperMotor.method), 405
                                              get_io_settings()
                                              method), 405

```

```

(msl.equipment.resources.thorlabs.kinesis.filter_flipperFilterFlipper.resources.thorlabs.kinesis.kcube_dc_servo
method), 480
method), 509
get_ip_config() get_jog_vel_params()
(msl.equipment.resources.avantes.avaspec.Avantes (msl.equipment.resources.thorlabs.kinesis.kcube_stepper_
method), 119 method), 542
get_ip_details() get_joystick_params()
(msl.equipment.resources.picotech.pt104.PT104 (msl.equipment.resources.thorlabs.kinesis.benchtop_stepp
method), 372 method), 407
get_jog_mode() get_lambda() (msl.equipment.resources.avantes.avaspec.Avantes
(msl.equipment.resources.thorlabs.kinesis.benchtop_steppe (msl.equipment.resources.thorlabs.kinesis.benchtop_stepper
method), 406 method), 420 or.BenchtopStepperMotor
get_jog_mode() get_lamptime()
(msl.equipment.resources.energetiq.eq99.EQ99
(msl.equipment.resources.thorlabs.kinesis.integrated_steppe (msl.equipment.resources.energetiq.eq99.EQ99
method), 486 method), 411 or.IntegratedStepperMotors
get_jog_mode() get_led_switches()
(msl.equipment.resources.thorlabs.kinesis.integrated_steppe
(msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo
method), 508 method), 517 or.KCubeDCServo
get_jog_mode() get_led_switches()
(msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo
(msl.equipment.resources.thorlabs.kinesis.kcube_steppe (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo
method), 542 method), 509 or.KCubeStepperMotor
get_jog_params_block() get_led_switches()
(msl.equipment.resources.thorlabs.kinesis.kcube_solenoid
(msl.equipment.resources.thorlabs.kinesis.benchtop_steppe (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo
method), 406 method), 521 or.BenchtopStepperMotor
get_jog_params_block() get_legacy_bus_scanning()
(msl.equipment.resources.nkt.nktpdll.NKT
(msl.equipment.resources.thorlabs.kinesis.integrated_steppe (msl.equipment.resources.nkt.nktpdll.NKT
method), 487 method), 418 or.IntegratedStepperMotors
get_jog_params_block() get_limit_mode()
(msl.equipment.resources.mks_instruments.pr4000b.PR4000b
(msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo
method), 508 method), 507 or.KCubeDCServo
get_jog_params_block() get_limit_switch_params()
(msl.equipment.resources.thorlabs.kinesis.benchtop_steppe
(msl.equipment.resources.thorlabs.kinesis.kcube_steppe (msl.equipment.resources.thorlabs.kinesis.benchtop_steppe
method), 542 method), 407 or.KCubeStepperMotor
get_jog_step_size() get_limit_switch_params()
(msl.equipment.resources.thorlabs.kinesis.benchtop_steppe (msl.equipment.resources.thorlabs.kinesis.integrated_steppe
method), 406 method), 487 or.IntegratedStepperMotors
get_jog_step_size() get_limit_switch_params()
(msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo
(msl.equipment.resources.thorlabs.kinesis.integrated_steppe (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo
method), 487 method), 509 or.IntegratedStepperMotors
get_jog_step_size() get_limit_switch_params()
(msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo
(msl.equipment.resources.thorlabs.kinesis.benchtop_steppe (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo
method), 508 method), 517 or.BenchtopStepperMotor
get_jog_step_size() get_limit_switch_params()
(msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo
(msl.equipment.resources.thorlabs.kinesis.benchtop_steppe (msl.equipment.resources.thorlabs.kinesis.benchtop_steppe
method), 542 method), 407 or.KCubeStepperMotor
get_jog_vel_params() get_limit_switch_params()
(msl.equipment.resources.thorlabs.kinesis.benchtop_steppe (msl.equipment.resources.thorlabs.kinesis.integrated_steppe
method), 407 method), 488 or.BenchtopStepperMotor
get_jog_vel_params() get_limit_switch_params()
(msl.equipment.resources.thorlabs.kinesis.benchtop_steppe (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo
method), 487 method), 509 or.KCubeDCServo
get_jog_vel_params() get_limit_switch_params()
(msl.equipment.resources.thorlabs.kinesis.benchtop_steppe

```

```

        method), 543
get_linearization_point()                               get_mmi_params_block()
                                                       (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo_
                                                       (msl.equipment.resources.mks_instruments.pr4000b.PR4000B) 510
                                                       method), 154
get_linearization_size()                            get_mmi_params_block()
                                                       (msl.equipment.resources.thorlabs.kinesis.kcube_solenoid_
                                                       (msl.equipment.resources.mks_instruments.pr4000b.PR4000B) 532
                                                       method), 155
get_lines()          (in      module      (msl.equipment.resources.thorlabs.kinesis.kcube_stepper_
      msl.equipment.resources.utils), 88           method), 543
get_lines() (msl.equipment.resources.utils.CHead) get_mmi_params_ext()
                                                       (method), 89
                                                       (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo_
get_log() (msl.equipment.resources.bentham.benhw32.Bentham) 510
                                                       (method), 129
get_log_size()                                get_mmi_params_ext()
                                                       (msl.equipment.resources.bentham.benhw32.Bentham) 532
                                                       (method), 129
get_lower_limit()                               get_mmi_params_ext()
                                                       (msl.equipment.resources.thorlabs.kinesis.kcube_stepper_
                                                       (msl.equipment.resources.mks_instruments.pr4000b.PR4000B) 543
                                                       method), 155
get_max_bw() (msl.equipment.resources.bentham.benhw32.Bentham) 3277
                                                       (method), 129
get_max_down_sample_ratio() (msl.equipment.resources.princeton_instruments.arc_instr_
                                                       (msl.equipment.resources.picotech.picoscope.picoscopehd.PicoScopeApi) 208
                                                       (method), 332
get_max_ets_values()                           get_mono_backlash_steps()
                                                       (msl.equipment.resources.princeton_instruments.arc_instr_
                                                       (msl.equipment.resources.picotech.picoscope.picoscopehd.PicoScopeApi) 283000A
                                                       (method), 348
get_max_segments()                            get_mono_diverter_pos()
                                                       (msl.equipment.resources.princeton_instruments.arc_instr_
                                                       (msl.equipment.resources.picotech.picoscope.picoscopehd.PicoScopeApi) 283000C
                                                       (method), 332
get_max_velocity()                           get_mono_diverter_pos_str()
                                                       (msl.equipment.resources.princeton_instruments.arc_instr_
                                                       (msl.equipment.resources.thorlabs.fwxx2c.FilterWheel) 384
                                                       (method), 595
get_message_buffer()                           get_mono_diverter_valid()
                                                       (msl.equipment.resources.princeton_instruments.arc_instr_
                                                       (msl.equipment.resources.energetiq.eq99.EQ99) 384
                                                       (method), 146
get_min_step()                                get_mono_double()
                                                       (msl.equipment.resources.princeton_instruments.arc_instr_
                                                       (msl.equipment.resources.bentham.benhw32.Bentham) 384
                                                       (method), 129
get_min_velocity()                           get_mono_double_intermediate_slit()
                                                       (msl.equipment.resources.princeton_instruments.arc_instr_
                                                       (msl.equipment.resources.thorlabs.fwxx2c.FilterWheel) 385
                                                       (method), 595
get_mmi_params()                            get_mono_double_subtractive()
                                                       (msl.equipment.resources.princeton_instruments.arc_instr_
                                                       (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo) 510
                                                       method), 510
get_mmi_params()                            get_mono_exit_slit()
                                                       (msl.equipment.resources.princeton_instruments.arc_instr_
                                                       (msl.equipment.resources.thorlabs.kinesis.kcube_solenoid) 531
                                                       method), 531
get_mmi_params()                            get_mono_filter_max_pos()
                                                       (msl.equipment.resources.princeton_instruments.arc_instr_
                                                       (msl.equipment.resources.thorlabs.kinesis.kcube_stepper) 538
                                                       method), 543
get_mmi_params()                            get_mono_filter_min_pos()
                                                       (msl.equipment.resources.princeton_instruments.arc_instr_
                                                       (msl.equipment.resources.thorlabs.kinesis.kcube_stepper) 543
                                                       method), 543

```

(*msl.equipment.resources.princeton_instruments.arc* (*instaquipmPrinrestbenthm.benhw32.Bentham32*
method), 385
get_mono_filter_position() get_mono_items()
(*msl.equipment.resources.princeton_instruments.arc* (*instaquipmPrinrestbenthm.benhw64.Bentham*
method), 386
get_mono_filter_present() get_mono_model()
(*msl.equipment.resources.princeton_instruments.arc* (*instaquipmPrinrestprinoneton_instruments.arc_instr*
method), 386
get_mono_focal_length() get_mono_motor_int()
(*msl.equipment.resources.princeton_instruments.arc* (*instaquipmPrinrestprinoneton_instruments.arc_instr*
method), 386
get_mono_gear_steps() get_mono_nm_rev_ratio()
(*msl.equipment.resources.princeton_instruments.arc* (*instaquipmPrinrestprinoneton_instruments.arc_instr*
method), 386
get_mono_grating() get_mono_precision()
(*msl.equipment.resources.princeton_instruments.arc* (*instaquipmPrinrestprinoneton_instruments.arc_instr*
method), 386
get_mono_grating_blaze() get_mono_preopen_model()
(*msl.equipment.resources.princeton_instruments.arc* (*instaquipmPrinrestprinoneton_instruments.arc_instr*
method), 386
get_mono_grating_density() get_mono_scan_rate_nm_min()
(*msl.equipment.resources.princeton_instruments.arc* (*instaquipmPrinrestprinoneton_instruments.arc_instr*
method), 386
get_mono_grating_gadjust() get_mono_serial()
(*msl.equipment.resources.princeton_instruments.arc* (*instaquipmPrinrestprinoneton_instruments.arc_instr*
method), 387
get_mono_grating_installed() get_mono_shutter_open()
(*msl.equipment.resources.princeton_instruments.arc* (*instaquipmPrinrestprinoneton_instruments.arc_instr*
method), 387
get_mono_grating_max() get_mono_shutter_valid()
(*msl.equipment.resources.princeton_instruments.arc* (*instaquipmPrinrestprinoneton_instruments.arc_instr*
method), 387
get_mono_grating_offset() get_mono_sine_drive()
(*msl.equipment.resources.princeton_instruments.arc* (*instaquipmPrinrestprinoneton_instruments.arc_instr*
method), 387
get_mono_half_angle() get_mono_slit_type()
(*msl.equipment.resources.princeton_instruments.arc* (*instaquipmPrinrestprinoneton_instruments.arc_instr*
method), 387
get_mono_init_grating() get_mono_slit_type_str()
(*msl.equipment.resources.princeton_instruments.arc* (*instaquipmPrinrestprinoneton_instruments.arc_instr*
method), 387
get_mono_init_scan_rate_nm() get_mono_slit_width()
(*msl.equipment.resources.princeton_instruments.arc* (*instaquipmPrinrestprinoneton_instruments.arc_instr*
method), 387
get_mono_init_wave_nm() get_mono_slit_width_max()
(*msl.equipment.resources.princeton_instruments.arc* (*instaquipmPrinrestprinoneton_instruments.arc_instr*
method), 387
get_mono_int_led_on() get_mono_turret()
(*msl.equipment.resources.princeton_instruments.arc* (*instaquipmPrinrestprinoneton_instruments.arc_instr*
method), 388
get_mono_items() get_mono_turret_gratings()

```
(msl.equipment.resources.princeton_instruments.arc(instaquipmPrinrestnresthodats.kinesis.kcube_dc_servo
method), 391                                              method), 511
get_mono_turret_max()                                     get_motor_params_ext()
                                                       (msl.equipment.resources.princeton_instruments.arc(instaquipmPrinrestnresthodats.kinesis.kcube_stepper
method), 390                                              method), 544
get_mono_wavelength_abs_cm()                                get_motor_travel_limits()
                                                       (msl.equipment.resources.princeton_instruments.arc(instaquipmPrinrestnresthodats.kinesis.benchtop_stopp
method), 391                                              method), 408
get_mono_wavelength_ang()                                  get_motor_travel_limits()
                                                       (msl.equipment.resources.princeton_instruments.arc(instaquipmPrinrestnresthodats.kinesis.integrated_stopp
method), 391                                              method), 488
get_mono_wavelength_cutoff_nm()                            get_motor_travel_limits()
                                                       (msl.equipment.resources.princeton_instruments.arc(instaquipmPrinrestnresthodats.kinesis.kcube_dc_servo
method), 391                                              method), 511
get_mono_wavelength_ev()                                   get_motor_travel_limits()
                                                       (msl.equipment.resources.princeton_instruments.arc(instaquipmPrinrestnresthodats.kinesis.kcube_stepper
method), 391                                              method), 544
get_mono_wavelength_micron()                             get_motor_travel_mode()
                                                       (msl.equipment.resources.princeton_instruments.arc(instaquipmPrinrestnresthodats.kinesis.benchtop_stopp
method), 391                                              method), 408
get_mono_wavelength_min_nm()                            get_motor_travel_mode()
                                                       (msl.equipment.resources.princeton_instruments.arc(instaquipmPrinrestnresthodats.kinesis.integrated_stopp
method), 391                                              method), 488
get_mono_wavelength_nm()                                 get_motor_travel_mode()
                                                       (msl.equipment.resources.princeton_instruments.arc(instaquipmPrinrestnresthodats.kinesis.kcube_dc_servo
method), 391                                              method), 511
get_mono_wavelength_rel_cm()                            get_motor_travel_mode()
                                                       (msl.equipment.resources.princeton_instruments.arc(instaquipmPrinrestnresthodats.kinesis.kcube_stepper
method), 392                                              method), 545
get_mono_wheel_int()                                    get_motor_velocity_limits()
                                                       (msl.equipment.resources.princeton_instruments.arc(instaquipmPrinrestnresthodats.kinesis.benchtop_stopp
method), 392                                              method), 409
get_motor_params()                                       get_motor_velocity_limits()
                                                       (msl.equipment.resources.thorlabs.kinesis.benchtop_<del>step</del>equipmtonBreschtrpeSteppdmMotkinesis.integrated_stopp
method), 408                                              method), 488
get_motor_params()                                       get_motor_velocity_limits()
                                                       (msl.equipment.resources.thorlabs.kinesis.integrated(stshpqnmotats:dnagatedSorplrs:Motosis.kcube_dc_servo
method), 488                                              method), 511
get_motor_params()                                       get_motor_velocity_limits()
                                                       (msl.equipment.resources.thorlabs.kinesis.kcube_dc_<del>serv</del>KCPmheDGSsources.thorlabs.kinesis.kcube_stepper_
method), 510                                              method), 545
get_motor_params()                                       get_move_absolute_position()
                                                       (msl.equipment.resources.thorlabs.kinesis.kcube_step(psl.equipment.KCubesStoppers.Motabbs.kinesis.benchtop_stopp
method), 544                                              method), 409
get_motor_params_ext()                                    get_move_absolute_position()
                                                       (msl.equipment.resources.thorlabs.kinesis.benchtop_<del>step</del>equipmtonBreschtrpeSteppdmMotkinesis.integrated_stopp
method), 408                                              method), 489
get_motor_params_ext()                                    get_move_absolute_position()
                                                       (msl.equipment.resources.thorlabs.kinesis.integrated(stshpqnmotats:dnagatedSorplrs:Motosis.kcube_dc_servo
method), 488                                              method), 511
get_motor_params_ext()                                    get_move_absolute_position()
```

(*msl.equipment.resources.thorlabs.kinesis.kcube_step*(*pos*, *equipment.KCubesStopper.Motor*, *kinesis.benchtop_stepper.method*), 545
get_move_relative_distance() get_next_message()
(*msl.equipment.resources.thorlabs.kinesis.benchtop_stop*(*equipment.BenchtopStepper.Motor*, *kinesis.filter_flipper.Filter*, *method*), 409
method), 409
get_move_relative_distance() get_next_message()
(*msl.equipment.resources.thorlabs.kinesis.integrated_stripper*(*stripper*, *integrated_Stripper.Motor*, *integrated_step*, *method*), 489
method), 489
get_move_relative_distance() get_next_message()
(*msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo*(*servo*, *KCubesStopper.Motor*, *kinesis.kcube_dc_servo.method*), 512
method), 512
get_move_relative_distance() get_next_message()
(*msl.equipment.resources.thorlabs.kinesis.kcube_step*(*pos*, *equipment.KCubesStopper.Motor*, *kinesis.kcube_solenoid.method*), 545
method), 532
get_n_groups() get_next_message()
(*msl.equipment.resources.bentham.benhw32.Bentham32.equipment.resources.thorlabs.kinesis.kcube stepper.method*), 129
method), 545
get_ncl_filter_max_pos() get_no_of_captures()
(*msl.equipment.resources.princeton_instruments.arc_instrument*(*PrincetonInstruments.picoscope.picoscope_ar*, *method*), 392
method), 332
get_ncl_filter_min_pos() get_no_of_dark_currents()
(*msl.equipment.resources.princeton_instruments.arc_instrument*(*PrincetonInstruments.picoscope.picoscope_ar*, *method*), 392
method), 128
get_ncl_filter_position() get_num_channels()
(*msl.equipment.resources.princeton_instruments.arc_instrument*(*PrincetonInstruments.picoscope.picoscope_ar*, *method*), 392
method), 410
get_ncl_filter_present() get_num_det() (*msl.equipment.resources.princeton_instruments.arc_instrument*(*PrincetonInstruments.picoscope.picoscope_ar*, *method*), 392
method), 392
get_ncl_mono_enum() get_num_devices()
(*msl.equipment.resources.avantes.avaspec.Avantes*(*method*), 393
method), 393
get_ncl_mono_setup() get_num_found_inst_ports()
(*msl.equipment.resources.princeton_instruments.arc_instrument*(*PrincetonInstruments.picoscope.picoscope_ar*, *method*), 393
method), 393
get_ncl_shutter_open() get_num_processed_captures()
(*msl.equipment.resources.princeton_instruments.arc_instrument*(*PrincetonInstruments.picoscope.picoscope_ar*, *method*), 393
method), 393
get_ncl_shutter_valid() get_num_pixels()
(*msl.equipment.resources.avantes.avaspec.Avantes*(*method*), 393
method), 393
get_ncl_ttl_in() get_number_positions()
(*msl.equipment.resources.thorlabs.kinesis.benchtop_step*(*method*), 393
method), 393
get_ncl_ttl_out() get_number_positions()
(*msl.equipment.resources.thorlabs.kinesis.filter_flipper.Filter*(*method*), 393
method), 393
get_ncl_ttl_valid() get_number_positions()
(*msl.equipment.resources.thorlabs.kinesis.integrated_step*(*method*), 393
method), 393
get_next_message() (*msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo*(*servo*, *KCubesStopper.Motor*, *kinesis.kcube_dc_servo.method*), 512
method), 512

method), 512
get_number_positions()
(msl.equipment.resources.thorlabs.kinesis.kcube_stepper
method), 545
get_ocx_version()
(msl.equipment.resources.optronic_laboratories.ol756ocx
method), 214
get_oem_parameter()
(msl.equipment.resources.avantes.avaspec.A
method), 120
get_offset()
(msl.equipment.resources.mks_instruments.pr4000bdyR4000B
method), 155
get_open_ports()
(msl.equipment.resources.nkt.nktpdll.NKT
static method), 178
get_operating_mode()
(msl.equipment.resources.thorlabs.kinesis.k
method), 532
get_operating_state()
(msl.equipment.resources.thorlabs.kinesis.k
method), 533
get_option()
(msl.equipment.resources.optronic_laboratory
method), 225
get_output()
(msl.equipment.resources.electron_dynamics.t
method), 137
get_output()
(msl.equipment.resources.energetiq.e
method), 144
get_output_range()
(msl.equipment.resources.mks_instruments.pr4000bsPR4000B
method), 155
get_over_current_protection()
(msl.equipment.resources.aim_tti.mx_seriesMX
method), 91
get_over_voltage_protection()
(msl.equipment.resources.aim_tti.mx_seriesMX
method), 91
get_parameter()
(msl.equipment.resources.avantes.avaspec.A
method), 120
get_pid_loop_encoder_coeff()
(msl.equipment.resources.thorlabs.kinesis.b
method), 410
get_pid_loop_encoder_coeff()
(msl.equipment.resources.thorlabs.kinesis.k
method), 546
get_pid_loop_encoder_params()
(msl.equipment.resources.thorlabs.kinesis.b
method), 410
get_pid_loop_encoder_params()
(msl.equipment.resources.thorlabs.kinesis.k
method), 546
get_pmt_flux_overload()
(msl.equipment.resources.optronic_laboratories.ol756ocx
method), 545
get_pmt_voltage()
(msl.equipment.resources.optronic_laboratories.ol756ocx
method), 604
get_port()
(msl.equipment.vxi11.RPCClient
method), 604
get_port_error_msg()
(msl.equipment.resources.nkt.nktpdll.NKT
method), 178
get_ports()
(msl.equipment.resources.thorlabs.fwxx2c.FilterWheelXX
method), 595
get_potentiometer()
(msl.equipment.resources.thorlabs.kinesis.benchtop_st
method), 595
get_potentiometer()
(msl.equipment.resources.thorlabs.kinesis.benchtop_st
method), 595
get_position()
(msl.equipment.resources.thorlabs.kinesis.benchtop_st
method), 489
get_position()
(msl.equipment.resources.thorlabs.kinesis.kcube_dc_serve
method), 512
get_position()
(msl.equipment.resources.thorlabs.kinesis.kcube_dc_serve
method), 546
get_position_count()
(msl.equipment.resources.thorlabs.fwxx2c.FilterWheelXX
method), 596
get_position_counter()
(msl.equipment.resources.thorlabs.kinesis.benchtop_st
method), 410
get_position_counter()
(msl.equipment.resources.thorlabs.kinesis.benchtop_st
method), 490
get_position_counter()
(msl.equipment.resources.thorlabs.kinesis.kcube_dc_serve
method), 512
get_position_counter()
(msl.equipment.resources.thorlabs.kinesis.benchtop_st
method), 410
get_position_counter()
(msl.equipment.resources.thorlabs.kinesis.kcube_dc_serve
method), 546

method), 490
get_potentiometer_params_block()
 (*msl.equipment.resources.thorlabs.kinesis.integrated_stacks_implemented_for_motion_instruments.pr4000b.PR4000b*.
 method), 490
get_power_params()
 (*msl.equipment.resources.thorlabs.kinesis.benchtop_stacks_implemented_for_motion_instruments.pr4000b.PR4000b*.
 method), 411
get_power_params()
 (*msl.equipment.resources.thorlabs.kinesis.integrated_stacks_implemented_for_motion_instruments.pr4000b.PR4000b*.
 method), 490
get_power_params()
 (*msl.equipment.resources.thorlabs.kinesis.kcube_stacks_implemented_for_motion_instruments.pr4000b.PR4000b*.
 method), 546
get_probe() (*msl.equipment.resources.picotech.picotest4000.PicoScope4000*.
 method), 350
get_quick_scan_rate()
 (*msl.equipment.resources.optronic_laboratories.classic_scanner.list*).
 method), 214
get_quick_scan_rate_index()
 (*msl.equipment.resources.optronic_laboratories.classic_scanner.gets_scanner_pixels*).
 method), 214
get_rack_digital_outputs()
 (*msl.equipment.resources.thorlabs.kinesis.benchtop_stacks_implemented_for_motion_instruments.pr4000b.PR4000b*.
 method), 411
get_rack_status_bits()
 (*msl.equipment.resources.thorlabs.kinesis.benchtop_stacks_implemented_for_motion_instruments.pr4000b.PR4000b*.
 method), 411
get_range() (*msl.equipment.resources.mks_instruments.gets_per1000_PR4000B*.
 method), 155
get_readout_itime_ms()
 (*msl.equipment.resources.princeton_instruments.arc_instrumentation.readouts.fwxx2c.FilterWheelXX*.
 method), 394
get_readout_model()
 (*msl.equipment.resources.princeton_instruments.arc_instrumentation.readouts.dynamics.tc_series.TC*.
 method), 394
get_readout_preopen_model()
 (*msl.equipment.resources.princeton_instruments.arc_instrumentation.readouts.mks_instruments.pr4000b.PR4000b*.
 static method), 394
get_readout_serial()
 (*msl.equipment.resources.princeton_instruments.arc_instrumentation.readouts.raicol_tec.RaicolTEC*.
 method), 394
get_real_value_from_device_unit()
 (*msl.equipment.resources.thorlabs.kinesis.benchtop_stacks_implemented_for_motion_instruments.pr4000b.PR4000b*.
 method), 411
get_real_value_from_device_unit()
 (*msl.equipment.resources.thorlabs.kinesis.integrated_stacks_implemented_for_motion_instruments.pr4000b.PR4000b*.
 method), 490
get_real_value_from_device_unit()
 (*msl.equipment.resources.thorlabs.kinesis.kcube_dc_gated_kappa_energetiq.eq99.EQ99*.
 method), 512
get_real_value_from_device_unit()
 (*msl.equipment.resources.thorlabs.kinesis.gets_kappa_energetiq.eq99.EQ99*.
 method), 145
get_signal_array()

```

(msl.equipment.resources.optronic_laboratories.ol756ocx.get_stage_axis_max_pos(), 513)
    method), 215
get_signal_mode() (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo.get_stage_axis_max_pos(), 513)
    method), 156
get_soft_limit_mode() (msl.equipment.resources.thorlabs.kinesis.benchtop_stepper_motor.get_stage_axis_max_pos(), 517)
    method), 411
get_soft_limit_mode() (msl.equipment.resources.thorlabs.kinesis.integrated_stepper_motors.get_stage_axis_min_pos(), 412)
    method), 491
get_soft_limit_mode() (msl.equipment.resources.thorlabs.kinesis.integrated_dc_servo.get_stage_axis_min_pos(), 513)
    method), 513
get_soft_limit_mode() (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo.get_stage_axis_min_pos(), 513)
    method), 547
get_software_version() (msl.equipment.resources.thorlabs.kinesis.benchtop_stepper_motor.get_standard_file(), 517)
    method), 412
get_software_version() (msl.equipment.resources.optronic_laboratories.ol756ocx.get_start_wavelength(), 515)
    method), 479
get_software_version() (msl.equipment.resources.thorlabs.kinesis.integrated_stepper_motors.get_status(), 412)
    method), 491
get_software_version() (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo.get_status(), 513)
    method), 533
get_software_version() (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo.get_status(), 513)
    method), 533
get_software_version() (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo.get_status(), 513)
    method), 547
get_solenoid_state() (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo.get_status(), 513)
    method), 533
get_speed() (msl.equipment.resources.optosigma.shot702.SHOT702.get_status_bits(), 513)
    method), 200
get_speed_mode() (msl.equipment.resources.thorlabs.kinesis.kcube_solenoid.get_status_bits(), 513)
    method), 596
get_speed_origin() (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo.get_status_by_serial(), 547)
    method), 200
get_stage_axis_max_pos() (msl.equipment.resources.avantes.avaspec.Avantes.get_std_file_enabled(), 513)
    method), 412
get_stage_axis_max_pos() (msl.equipment.resources.thorlabs.kinesis.integrated_stepper_motors.get_std_file_enabled(), 513)
    method), 412

```

get_steps() (*msl.equipment.resources.optosigma.shot702.SHOT702Equipment.resources.picotech.picoscope.ps4000.PicoScope4000A.get_steps()*, 200
method), 200
get_str() (*msl.equipment.resources.bentham.benhvga.BenHvga2.get_BenHvga2channel_time_offset64()*,
method), 129
get_streaming_last_values()
(*msl.equipment.resources.picotech.picoscope.get_trigger_config_params()*,
method), 329
get_streaming_latest_values()
(*msl.equipment.resources.picotech.picoscope.get_trigger_config_params()*,
method), 333
get_streaming_values()
(*msl.equipment.resources.picotech.picoscope.get_trigger_config_params()*,
method), 329
get_streaming_values_no_aggregation()
(*msl.equipment.resources.picotech.picoscope.get_trigger_config_params()*,
method), 329
get_string() (*msl.equipment.resources.picotech.picoscope.pic1000a.PicoScope4000A.get_trigger_config_params_block()*,
method), 352
get_struct_imports()
(*msl.equipment.resources.utils.CHeader.get_struct_imports()*,
method), 89
get_termination()
(*msl.equipment.resources.energetiq.eq99.EQ99.get_termination()*,
method), 146
get_text_between_brackets()
(*msl.equipment.resources.utils.CHeader.get_text_between_brackets()*,
static method), 89
get_time_to_current_pos()
(*msl.equipment.resources.thorlabs.fwxx2c.FilterWheelXX2C.get_time_to_current_pos()*,
method), 596
get_timebase()
(*msl.equipment.resources.picotech.picoscope.picoscopeapi.PicoScope4000A.get_timebase()*,
method), 330
get_timebase()
(*msl.equipment.resources.picotech.picoscope.picoscopeapi.PicoScope4000A.get_timebase()*,
method), 333
get_timebase2()
(*msl.equipment.resources.picotech.picoscope.picoscopeapi.PicoScope4000A.get_timebase2()*,
method), 333
get_timeout() (*msl.equipment.hislip.HiSLIPClient.get_timeout()*,
method), 74
get_times_and_values()
(*msl.equipment.resources.picotech.picoscope.picoscopeapi.PicoScope4000A.get_times_and_values()*,
method), 330
get_transit_time()
(*msl.equipment.resources.thorlabs.kinesis.filter_flipper.Flipper.get_transit_time()*,
method), 481
get_travel_per_pulse()
(*msl.equipment.resources.optosigma.shot702.SHOT702Equipment.resources.thorlabs.kinesis.benchtop_stepper.get_travel_per_pulse()*,
method), 201
get_trigger_channel_time_offset()
(*msl.equipment.resources.picotech.picoscope.get_trigger_channel_time_offset64()*,
method), 350
get_trigger_config_params()
(*msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo.get_trigger_config_params()*,
method), 533
get_trigger_config_params_block()
(*msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo.get_trigger_config_params_block()*,
method), 533
get_trigger_info_bulk()
(*msl.equipment.resources.picotech.picoscope.ps3000a.PicoScope3000A.get_trigger_info_bulk()*,
method), 348
get_trigger_info_bulk()
(*msl.equipment.resources.picotech.picoscope.ps5000a.PicoScope5000A.get_trigger_info_bulk()*,
method), 548
get_trigger_mode()
(*msl.equipment.resources.energetiq.eq99.EQ99.get_trigger_mode()*,
method), 548
get_trigger_params_params()
(*msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo.get_trigger_params_params()*,
method), 533
get_trigger_params_params_block()
(*msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo.get_trigger_params_params_block()*,
method), 514
get_trigger_switches()
(*msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo.get_trigger_switches()*,
method), 412
get_trigger_switches()
(*msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo.get_trigger_switches()*,
method), 548

```

(msl.equipment.resources.thorlabs.kinesis.integrated_steppid157.IntegratedStepperMotors
method), 491
get_trigger_time_offset() (msl.equipment.resources.thorlabs.kinesis.benchtop_steppid113
(msl.equipment.resources.picotech.picoscope.picoscopepid149.PicoScopeApi
method), 333
get_trigger_time_offset64() (msl.equipment.resources.thorlabs.kinesis.integrated_steppid149
(msl.equipment.resources.picotech.picoscope.picoscopepid149.PicoScopeApi
method), 333
get_unit_info() (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo
(msl.equipment.resources.picotech.picoscope.picoscopepid50.S50
method), 326
get_unit_info() (msl.equipment.resources.thorlabs.kinesis.kcube_steerer_
(msl.equipment.resources.picotech.pt104.PT104
method), 548
method), 372
get_upper_limit() (msl.equipment.resources.thorlabs.kinesis.benchtop_steppid413
(msl.equipment.resources.mks_instruments.pr4000b.PR4000B
method), 156
get_values() (msl.equipment.resources.picotech.pt104.PT104
method), 372
get_values() (msl.equipment.resources.picotech.picoscopepidblock.PicoScope2k3k
method), 330
get_values() (msl.equipment.resources.picotech.picoscopepid515.PicoScopeApi
method), 333
get_values_async() (msl.equipment.resources.thorlabs.kinesis.kcube_steerer_
(msl.equipment.resources.picotech.picoscope.picoscopepid518.PicoScopeApi
method), 334
get_values_bulk() (msl.equipment.resources.picotech.picoscopepidversion.PicScopeApaint.resources.optosigma.shot702.SH
method), 334
get_values_bulk_asyc() (msl.equipment.resources.picotech.picoscope.ps6000(PicScope6000
method), 357
get_values_overlapped() (msl.equipment.resources.picotech.picoscope.picoscopepid92.PicoScopeApi
method), 334
get_values_overlapped_bulk() (msl.equipment.resources.picotech.picoscopepidversion.PicScopeApaint
method), 335
get_values_trigger_channel_time_offset_bulk() (msl.equipment.resources.picotech.picoscopepid92.PicScopeApaint()
method), 350
get_values_trigger_channel_time_offset_bulk64() (msl.equipment.resources.picotech.picoscopepid92.PicScopeApaint()
method), 350
get_values_trigger_time_offset_bulk() (msl.equipment.resources.picotech.picoscopepid92.PicScopeApaint()
method), 335
get_values_trigger_time_offset_bulk64() (msl.equipment.resources.picotech.picoscopepid92.PicScopeApaint()
method), 335
get_valves() (msl.equipment.resources.mks_instrumentx.pr4000h
get_valvepid92.Pr4000h_info()

```

(*msl.equipment.resources.bentham.benhw32.Bentham32* (*msl.equipment.resources.picotech.picoscope.enums.PS4000ACh*
 method), 128
attribute), 286

GetDescriptors (*class* in *msl.equipment.hislip*), 67
GreisingerError, 55

group_add() (*msl.equipment.resources.bentham.benhw32.Bentham32*
 method), 128

group_execute_trigger()
 (*msl.equipment.connection_prologix.ConnectionPrologix*
 method), 35

group_remove()
 (*msl.equipment.resources.bentham.benhw32.Bentham32*
 method), 128

H

HALF_SINE (*msl.equipment.resources.picotech.picoscope.enums.PS4000ACh*
 attribute), 281

H_MAX (*msl.equipment.resources.picotech.picoscope.enums.PS4000ACh*
 attribute), 282

H_MIN (*msl.equipment.resources.picotech.picoscope.enums.PS4000ACh*
 attribute), 282

HALF_SINE (*msl.equipment.resources.picotech.picoscope.enums.PS4000ACh*
 attribute), 246

HALF_SINE (*msl.equipment.resources.picotech.picoscope.enums.PS4000ACh*
 attribute), 238

HALF_SINE (*msl.equipment.resources.picotech.picoscope.enums.PS4000ACh*
 attribute), 264

HALF_SINE (*msl.equipment.resources.picotech.picoscope.enums.PS4000ACh*
 attribute), 285

HALF_SINE (*msl.equipment.resources.picotech.picoscope.enums.PS4000ACh*
 attribute), 275

HALF_SINE (*msl.equipment.resources.picotech.picoscope.enums.PS4000ACh*
 attribute), 309

HALF_SINE (*msl.equipment.resources.picotech.picoscope.enums.PS4000ACh*
 attribute), 299

HALF_SINE (*msl.equipment.resources.picotech.picoscope.enums.PS4000ACh*
 attribute), 318

HALF_SINE_MAX_FREQUENCY (*msl.equipment.resources.picotech.picoscope.ps2000a.PulseWidthType*
 attribute), 309

HALF_SINE_MAX_FREQUENCY
 (*msl.equipment.resources.picotech.picoscope.ps2000a.PulseWidthType*
 attribute), 309

HALF_SINE_MAX_FREQUENCY
 (*msl.equipment.resources.picotech.picoscope.ps3000a.PulseWidthType*
 attribute), 348

HALF_SINE_MAX_FREQUENCY
 (*msl.equipment.resources.picotech.picoscope.ps4000a.PulseWidthType*
 attribute), 348

HALF_SINE_MAX_FREQUENCY
 (*msl.equipment.resources.picotech.picoscope.ps4000a.PulseWidthType*
 attribute), 348

HALF_SINE_MAX_FREQUENCY
 (*msl.equipment.resources.picotech.picoscope.ps5000a.PulseWidthType*
 attribute), 354

HALF_SINE_MAX_FREQUENCY
 (*msl.equipment.resources.picotech.picoscope.ps5000a.PulseWidthType*
 attribute), 354

HALF_SINE_MAX_FREQUENCY
 (*msl.equipment.resources.picotech.picoscope.ps5000a.PulseWidthType*
 attribute), 354

HALF_SINE_MAX_FREQUENCY
 (*msl.equipment.resources.picotech.picoscope.ps6000a.PulseWidthType*
 attribute), 354

(*msl.equipment.resources.picotech.picoscope.HARDWARE_VERSION*)
attribute), 357

HAMS10420_FIRST_USED_DARK_PIXEL
(*msl.equipment.resources.avantes.avaspec.Avantes*.*hardwareVersion*
attribute), 107

HAMS10420_TOTAL_DARK_PIXELS
(*msl.equipment.resources.avantes.avaspec.Avantes*.*hardwareVersion*
attribute), 107

HAMS10420_USED_DARK_PIXELS
(*msl.equipment.resources.avantes.avaspec.Avantes*.*hardwareVersion*
attribute), 107

HAMS11071_FIRST_USED_DARK_PIXEL
(*msl.equipment.resources.avantes.avaspec.Avantes*.*hardwareVersion*
attribute), 107

HAMS11071_TOTAL_DARK_PIXELS
(*msl.equipment.resources.avantes.avaspec.Avantes*.*hardwareVersion*
attribute), 107

HAMS11071_USED_DARK_PIXELS
(*msl.equipment.resources.avantes.avaspec.Avantes*.*hardwareVersion*
attribute), 107

HAMS11155_TOTAL_DARK_PIXELS
(*msl.equipment.resources.avantes.avaspec.Avantes*.*hardwareVersion*
attribute), 107

HAMS7031_FIRST_USED_DARK_PIXEL
(*msl.equipment.resources.avantes.avaspec.Avantes*.*header*
attribute), 107

HAMS7031_TOTAL_DARK_PIXELS
(*msl.equipment.resources.avantes.avaspec.Avantes*.*header*
attribute), 107

HAMS7031_USED_DARK_PIXELS
(*msl.equipment.resources.avantes.avaspec.Avantes*.*header*
attribute), 107

HAMS9840_FIRST_USED_DARK_PIXEL
(*msl.equipment.resources.avantes.avaspec.Avantes*.*HIGH*
attribute), 107

HAMS9840_TOTAL_DARK_PIXELS
(*msl.equipment.resources.avantes.avaspec.Avantes*.*HIGH*
attribute), 107

HAMS9840_USED_DARK_PIXELS
(*msl.equipment.resources.avantes.avaspec.Avantes*.*HIGH*
attribute), 107

handle (*msl.equipment.connection_gpib.Connection*.*KINDB*
property), 25

handle (*msl.equipment.resources.picotech.picoscope.picoscope*.*pathFluence*
property), 325

HARDWARE_VERSION
(*msl.equipment.resources.picotech.picoscope.enums.RetroSata*).*ES666Api*
attribute), 233

HARDWARE_VERSION
(*msl.equipment.resources.picotech.picoscope.enums.RetroSata*).*ES666Api*
attribute), 233

HARDWAREVERSION
(*msl.equipment.resources.picotech.picoscope.enums.PS3000A*)
attribute), 254

hardwareVersion
(*msl.equipment.resources.thorlabs.kinesis structs.TLI_HardwareVersion*)
attribute), 571

hasLastMsgTimerOverrun()
(*msl.equipment.resources.thorlabs.kinesis.benchtop_steppe*
method), 413

hasLastMsgTimerOverrun()
(*msl.equipment.resources.thorlabs.kinesis.filter_flipper.Filter*
method), 482

hasLastMsgTimerOverrun()
(*msl.equipment.resources.thorlabs.kinesis.integrated_steppe*
method), 492

hasLastMsgTimerOverrun()
(*msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo*
method), 515

hasLastMsgTimerOverrun()
(*msl.equipment.resources.thorlabs.kinesis.kcube_solenoid*
method), 533

heartbeat()
(*msl.equipment.resources.avantes.avaspec.Avantes*.*method*), 121

HeartbeatResponseType (class in
msl.equipment.resources.avantes.avaspec), 104

HIGH (*msl.equipment.resources.picotech.picoscope.enums.PS2000A*)
attribute), 249

highGearAcceleration
(*msl.equipment.resources.thorlabs.kinesis.structs.MOT_Jerk*)
attribute), 574

highGearMaxVelocity
(*msl.equipment.resources.thorlabs.kinesis.structs.MOT_Jerk*)
attribute), 574

KINDB (*msl.equipment.resources.thorlabs.kinesis.structs.KNA_IC*)
VoltageOutputRoute

pathFluence (class in *msl.equipment.picoscope*), 586

highVoltageOutRange
(*msl.equipment.resources.thorlabs.kinesis.structs.KNA_IC*)

HiSLIPClient (class in *msl.equipment.hislip*), 73

HiSLIPEException, 59

holdOnPS2000 (*msl.equipment.resources.picotech.picoscope.picoscope*.*method*), 335

home() (*msl.equipment.resources.optosigma.shot702.SHOT702*
 method), 201
 id_number() (*msl.equipment.resources.greisinger.gmh3000.GMH3000*
 method), 148
 home() (*msl.equipment.resources.thorlabs.kinesis.benchtop_stepper_motor.BenchtopStepperMotor*
 method), 414
 identify() (*msl.equipment.resources.thorlabs.kinesis.benchtop_stepper_motor*
 method), 415
 home() (*msl.equipment.resources.thorlabs.kinesis.filter_flipper.FilterFlipper*
 method), 480
 identify() (*msl.equipment.resources.thorlabs.kinesis.filter_flipper*
 method), 481
 home() (*msl.equipment.resources.thorlabs.kinesis.integrated_stepper_motors.IntegratedStepperMotors*
 method), 492
 identify() (*msl.equipment.resources.thorlabs.kinesis.integrated_stepper_motors*
 method), 493
 home() (*msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo.KCubeDCServo*
 method), 515
 identify() (*msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo*
 method), 516
 home() (*msl.equipment.resources.thorlabs.kinesis.kcube stepper_motor.KCubeStepperMotor*
 method), 549
 identify() (*msl.equipment.resources.thorlabs.kinesis.kcube_sole*
 method), 534
 horizontalComponent
 (msl.equipment.resources.thorlabs.kinesis.structs.NTIFY_HYDRAEquipment
 attribute), 577
 IdentifyHYDRAEquipment()
 method), 549
 host (*msl.equipment.connection_socket.ConnectionSocket*
 property), 42
 Identity() (*msl.equipment.resources.energetiq.eq99.EQ99*
 method), 141
 host (*msl.equipment.connection_tcpip_hislip.ConnectionTCPPIPHISLIP*
 property), 43
 IdentityTCPPIPHISLIP()
 method), 157
 host (*msl.equipment.connection_tcpip_vxi11.ConnectionTCPPIPVXI11*
 property), 46
 Ignore()
 attribute), 454
 host (*msl.equipment.connection_zeromq.ConnectionZeroMQ*
 property), 49
 ZeroMQ()
 attribute), 455
 Hour (*msl.equipment.resources.nkt.nktpdll.DateTimeType*
 attribute), 162
 ILX_TOTAL_DARK_PIXELS
 hubAnalogOutput
 (msl.equipment.resources.avantes.avaspec.Avantes
 attribute), 592
 TSG_IOSettings()
 attribute), 597
 HubAnalogueModes
 (class in (msl.equipment.resources.avantes.avaspec.Avantes
 msl.equipment.resources.thorlabs.kinesis.enums), attribute), 107
 467
 import_config_file()
 humidity() (*msl.equipment.resources.omega.ithx.iTHX*
 method), 196
 (msl.equipment.resources.optronic_laboratories.ol756ocx
 method), 216
 HV (*msl.equipment.resources.thorlabs.kinesis.enums.PPC_IQOutputMode*
 attribute), 448
 Import_IQSetup()
 attribute), 216
 HW_TRIGGER_MODE
 (msl.equipment.resources.avantes.avaspec.AnalogIO
 attribute), 107
 IN_RANGE (*msl.equipment.resources.picotech.picoscope.enums.PS2000TriggerChannelProperties*
 attribute), 251
 hysteresis (*msl.equipment.resources.picotech.picoscope.enums.PS2000TriggerChannelProperties*
 attribute), 358
 IN_RANGE (*msl.equipment.resources.picotech.picoscope.enums.PS2000TriggerChannelProperties*
 attribute), 239
 hysteresis (*msl.equipment.resources.picotech.picoscope.enums.PS3000TriggerChannelProperties*
 attribute), 360
 IN_RANGE (*msl.equipment.resources.picotech.picoscope.enums.PS3000TriggerChannelProperties*
 attribute), 268
 hysteresis (*msl.equipment.resources.picotech.picoscope.enums.PS5000TriggerChannelProperties*
 attribute), 367
 IN_RANGE (*msl.equipment.resources.picotech.picoscope.enums.PS5000TriggerChannelProperties*
 attribute), 257
 hysteresisLower
 (msl.equipment.resources.picotech.picoscope.structs.PS6000TriggerChannelProperties
 attribute), 370
 IN_RANGE (*msl.equipment.resources.picotech.picoscope.enums.PS4000TriggerChannelProperties*
 attribute), 282
 hysteresisUpper
 (msl.equipment.resources.picotech.picoscope.structs.PS6000TriggerChannelProperties
 attribute), 370
 IN_RANGE (*msl.equipment.resources.picotech.picoscope.enums.PS5000TriggerChannelProperties*
 attribute), 313
 IN_RANGE (*msl.equipment.resources.picotech.picoscope.enums.PS5000TriggerChannelProperties*
 attribute), 314

attribute), 303
IN_RANGE (*msl.equipment.resources.picotech.picoscope.integralGain.PS2000APulseWidthType*.resources.thorlabs.kinesis.structs.M
attribute), 323
increment_current() integralGain (*msl.equipment.resources.thorlabs.kinesis.structs.M*
(*msl.equipment.resources.aim_tti.mx_series.MXSeries*.attribute), 574
method), 92 integralGain (*msl.equipment.resources.thorlabs.kinesis.structs.M*
increment_voltage() attribute), 576
(*msl.equipment.resources.aim_tti.mx_series.MXSeries*.attribute), 576
method), 93 integralGain (*msl.equipment.resources.thorlabs.kinesis.structs.M*
init() integralGain (*msl.equipment.resources.thorlabs.kinesis.structs.Q*
method), 121 attribute), 588
init() integralGain (*msl.equipment.resources.princeton_instrument.integralGain.emslPerkinElmerResources.thorlabs.kinesis.structs.Q*
static method), 400 attribute), 589
init() (*msl.equipment.vxi11.RPCClient* method), integralGain (*msl.equipment.resources.thorlabs.kinesis.structs.T*
604 attribute), 593
initial_encryption integralLimit (*msl.equipment.resources.thorlabs.kinesis.structs.*
(*msl.equipment.hislip.InitializeResponse* attribute), 575
property), 61 integralLimit (*msl.equipment.resources.thorlabs.kinesis.structs.*
initialise() (*msl.equipment.resources.bentham.benhw32.Benhw32*.attribute), 574
method), 128 integralLimit (*msl.equipment.resources.thorlabs.kinesis.structs.*
initialise() (*msl.equipment.resources.bentham.benhw64.Benhw64*.attribute), 576
method), 131 integralTerm (*msl.equipment.resources.thorlabs.kinesis.structs.K*
Initialize (class in *msl.equipment.hislip*), 61 attribute), 587
Initialize (*msl.equipment.hislip.MessageType* integralTerm (*msl.equipment.resources.thorlabs.kinesis.structs.F*
attribute), 57 attribute), 579
initialize() (*msl.equipment.hislip.SyncClient* IntegratedStepperMotors (class in
method), 75 *msl.equipment.resources.thorlabs.kinesis.integrated_stepp*
InitializeResponse (class in 483
msl.equipment.hislip), 61 IntegrationTime (class in
attribute), 57 *msl.equipment.resources.cmi.sia3*),
InitializeResponse 131 Interface (class in *msl.equipment.constants*), 50
(*msl.equipment.hislip.MessageType* Interface (class in *msl.equipment.record_types.ConnectionRecord*
attribute), 57 attribute), 86
INPUT (*msl.equipment.resources.thorlabs.fwxx2c.TriggerInterface*.attribute), 594
INSIDE (*msl.equipment.resources.picotech.picoscope.integralGain.PS2000APulseWidthType*.thresholdDirection
attribute), 249 (msl.equipment.connection_gpib.ConnectionGPIB
INSIDE (*msl.equipment.resources.picotech.picoscope.enums.PS2000APulseWidthType*.thresholdDirection
attribute), 238 InterfaceType (class in
INSIDE (*msl.equipment.resources.picotech.picoscope.enums.PS3000APulseWidthType*.thresholdDirection.avaspec),
attribute), 266 98
INSIDE (*msl.equipment.resources.picotech.picoscope.integralGain.PS2000APulseWidthType*.thresholdDirection.sources.avantes.avaspec.Avante
attribute), 256 attribute), 110
INSIDE (*msl.equipment.resources.picotech.picoscope.integralGain.PS2000APulseWidthType*.thresholdDirection.sources.avantes.avaspec.Broad
attribute), 289 attribute), 99
INSIDE (*msl.equipment.resources.picotech.picoscope.integralGain.PS4000APulseWidthType*.thresholdDirection
attribute), 277 (msl.equipment.vxi11.RPCClient
INSIDE (*msl.equipment.resources.picotech.picoscope.enums.PS5000ATime*.thresholdDirection
attribute), 311 Interrupted (*msl.equipment.hislip.MessageType*
INSIDE (*msl.equipment.resources.picotech.picoscope.enums.PS5000ATime*.thresholdDirection
attribute), 302 INVALID_AVIS_HANDLE_VALUE
INSIDE (*msl.equipment.resources.picotech.picoscope.enums.PS6000ATime*.thresholdDirection.avantes.avaspec.Avantes

attribute), 106

INVALID_INIT_SEQUENCE (*msl.equipment.hislip.ErrorType attribute*), 58

ipv4_addresses() (*in module msl.equipment.utils*), 601

IrradianceType (*class in msl.equipment.resources.avantes.avaspec*), 101

is_calibration_active() (*msl.equipment.resources.thorlabs.kinesis.bogus_impossible_msl_equip_and_seepppe_Motor*), 414

is_calibration_active() (*msl.equipment.resources.thorlabs.kinesis.integrated_stepper_motors*), 492

is_calibration_active() (*msl.equipment.resources.thorlabs.kinesis.kdogstep*), 549

is_calibration_due() (*msl.equipment.record_types.EquipmentRecord method*), 81

is_channel_valid() (*msl.equipment.resources.thorlabs.kinesis.bogus_impossible_msl_equip_and_seepppe_Motor*), 414

is_inited() (*msl.equipment.resources.princeton_instrument*), 400

is_led_flashing() (*msl.equipment.resources.picotech.picoscope*), 335

is_moving() (*msl.equipment.resources.optosigma.shot702.SHOT702*), 201

is_open() (*msl.equipment.resources.thorlabs.fwxx2c.FilterWheelXX*), 596

is_operable (*msl.equipment.record_types.EquipmentRecord attribute*), 80

is_output_on() (*msl.equipment.resources.aim_tti.mx_series*), 93

is_ready() (*msl.equipment.resources.picotech.picoscope*), 326

is_trigger_or_pulse_width_qualifier_enabled() (*msl.equipment.resources.picotech.picoscope*), 335

isCustomType (*msl.equipment.resources.thorlabs.kinesis.structs.IEquipmentInfo*), 571

isKnownType (*msl.equipment.resources.thorlabs.kinesis.structs.IEquipmentInfo*), 571

isLaser (*msl.equipment.resources.thorlabs.kinesis.structs.TLI_DeviceInfo*), 571

isPiezoDevice (*msl.equipment.resources.thorlabs.kinesis.structs.IEquipmentInfo*), 571

isRack (*msl.equipment.resources.thorlabs.kinesis.structs.TLI_DeviceInfo*), 571

at- **iTHX** (*class in msl.equipment.resources.omega.ithx*), 195

J

jerk (*msl.equipment.resources.thorlabs.kinesis.structs.MOT_Velocity*), 572

Jog (*msl.equipment.resources.thorlabs.kinesis.enums.TIM_Buttons attribute*), 475

JogContinuous (*msl.equipment.resources.thorlabs.kinesis.enums.TIM_JogMode*), 475

JogStep (*msl.equipment.resources.thorlabs.kinesis.enums.KIM_JogMode attribute*), 453

Joystick (*msl.equipment.resources.thorlabs.kinesis.enums.PPC_Inputs attribute*), 448

JoystickBnc (*msl.equipment.resources.thorlabs.kinesis.enums.PPC_Inputs attribute*), 448

JoystickDirectionSense (*msl.equipment.resources.thorlabs.kinesis.structs.KPZ_Motor*), 587

JoystickMode (*msl.equipment.resources.thorlabs.kinesis.structs.KPZ_Motor attribute*), 587

JS_Jog (*msl.equipment.resources.thorlabs.kinesis.enums.KIM_JoystickMode*), 456

JS_Velocity (*msl.equipment.resources.thorlabs.kinesis.enums.KIM_Velocity*), 456

K

KCube_Brushless_Motor (*msl.equipment.resources.thorlabs.kinesis.motion_control*), 567

KCube_DC_Servo (*msl.equipment.resources.thorlabs.kinesis.motion_control*), 567

KCube_Inertial_Motor (*msl.equipment.resources.thorlabs.kinesis.motion_control*), 567

KCube_Picoscope (*msl.equipment.resources.thorlabs.kinesis.motion_control*), 567

KCube_NanoTrak (*msl.equipment.resources.thorlabs.kinesis.motion_control*), 567

KCube_Piezo (<i>msl.equipment.resources.thorlabs.kinesis.motion_sensors.KCubePiezoMotionControl</i> , <i>msl.equipment.resources.thorlabs.kinesis.structs</i>), attribute), 567	584
KCube_Solenoid	KIM_HomeParameters (class in <i>msl.equipment.resources.thorlabs.kinesis.motion_commands.KCubeSolenoid</i> , <i>msl.equipment.resources.thorlabs.kinesis.structs</i>), attribute), 567
KCube_Stepper_Motor	KIM_JogMode (class in <i>msl.equipment.resources.thorlabs.kinesis.motion_commands.KCubeStepperMotor</i> , <i>msl.equipment.resources.thorlabs.kinesis.enums</i>), attribute), 567
KCubeDCServo	(class in KIM_JogParameters (class in <i>msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo</i> , <i>msl.equipment.resources.thorlabs.kinesis.structs</i>), 505
KCubeSolenoid	(class in KIM_JoystickModes (class in <i>msl.equipment.resources.thorlabs.kinesis.kcube_solenoid</i> , <i>msl.equipment.resources.thorlabs.kinesis.enums</i>), 530
KCubeStepperMotor	(class in KIM_LimitSwitchModes (class in <i>msl.equipment.resources.thorlabs.kinesis.kcube_stepper</i> , <i>msl.equipment.resources.thorlabs.kinesis.enums</i>), 539
KD_TrigOut_Diff	KIM_LimitSwitchParameters (class in <i>msl.equipment.resources.thorlabs.kinesis.enums.QDnKIPtrigMode</i> , <i>msl.equipment.resources.thorlabs.kinesis.structs</i>), attribute), 470
KD_TrigOut_GPO	KIM_MMICChannelParameters (class in <i>msl.equipment.resources.thorlabs.kinesis.enums.QDnKIPtrigMode</i> , <i>msl.equipment.resources.thorlabs.kinesis.structs</i>), attribute), 470
KD_TrigOut_Sum	KIM_MMIPParameters (class in <i>msl.equipment.resources.thorlabs.kinesis.enums.QDnKIPtrigMode</i> , <i>msl.equipment.resources.thorlabs.kinesis.structs</i>), attribute), 470
KD_TrigOut_SumDiff	KIM_Status (class in <i>msl.equipment.resources.thorlabs.kinesis.enums.QDnKIPtrigMode</i> , <i>msl.equipment.resources.thorlabs.kinesis.structs</i>), attribute), 470
KERNEL_DRIVER_VERSION	KIM_TravelDirection (class in <i>msl.equipment.resources.picotech.picoscope.enums.PS2000Ipfront</i> , <i>msl.equipment.resources.thorlabs.kinesis.enums</i>), attribute), 235
KERNEL_DRIVER_VERSION	KIM_TrigIOConfig (class in <i>msl.equipment.resources.picotech.picoscope.enums.PS2000Ipfront</i> , <i>msl.equipment.resources.thorlabs.kinesis.structs</i>), attribute), 254
KERNEL_VERSION	KIM_TrigModes (class in <i>msl.equipment.resources.picotech.picoscope.enums.PicotechEquipmentApis</i> , <i>msl.equipment.resources.thorlabs.kinesis.enums</i>), attribute), 233
KIM_Channels	(class in KIM_TrigParamsParameters (class in <i>msl.equipment.resources.thorlabs.kinesis.enums</i> , <i>msl.equipment.resources.thorlabs.kinesis.structs</i>), 453
KIM_DirectionSense	(class in KIM_TrigPolarities (class in <i>msl.equipment.resources.thorlabs.kinesis.enums</i> , <i>msl.equipment.resources.thorlabs.kinesis.enums</i>), 454
KIM_DriveOPParameters	(class in KLD_Disabled (<i>msl.equipment.resources.thorlabs.kinesis.enums.KLD_TriggerOpType</i>), <i>msl.equipment.resources.thorlabs.kinesis.structs</i>), attribute), 458
KIM_FBSignalMode	(class in KLD_HighStability (<i>msl.equipment.resources.thorlabs.kinesis.enums.KLD_TriggerOpType</i>), <i>msl.equipment.resources.thorlabs.kinesis.enums</i>), attribute), 458
KIM_FeedbackSigParams	(class in KLD_Input (<i>msl.equipment.resources.thorlabs.kinesis.enums.KLD_TriggerOpType</i>), attribute), 458

KLD_InterlockEnabled
(*msl.equipment.resources.thorlabs.kinesis.enums.KLSDaTriggerMode* (class in
attribute), 458)

KLD_LaserOn (*msl.equipment.resources.thorlabs.kinesis.enums.KLD_TriggerMode*
attribute), 458

KLD_LowStability
(*msl.equipment.resources.thorlabs.kinesis.enums.KLD5TriggerMode*
attribute), 458

KLD_MMIParams (class in attribute), 460
msl.equipment.resources.thorlabs.kinesis.structs.KLS_SetPointChange
584

KLD_Output (*msl.equipment.resources.thorlabs.kinesis.enums.KLSDTiggerMode*
attribute), 458

KLD_SetPointChange
(*msl.equipment.resources.thorlabs.kinesis.enums.KLD5TriggerMode*
attribute), 458

KLD_TriggerMode (class in *msl.equipment.resources.thorlabs.kinesis structs*),
msl.equipment.resources.thorlabs.kinesis.enums, 585
458

KLD_TrigIOParams (class in (*msl.equipment.resources.thorlabs.kinesis.enums.KLS_TrigIOParams*
msl.equipment.resources.thorlabs.kinesis.structs), attribute), 460
585

KLD_TrigPol_High
(*msl.equipment.resources.thorlabs.kinesis.enums.KLDTrigPolarity*
attribute), 459

KLD_TrigPol_Low
(*msl.equipment.resources.thorlabs.kinesis.enums.KLDTrigPolarity*
attribute), 459

KLD_TrigPolarity (class in *msl.equipment.resources.thorlabs.kinesis.enums.KLSDTiggerMode*
msl.equipment.resources.thorlabs.kinesis.structs), 582
458

KMOT_MMIParams (class in *msl.equipment.resources.thorlabs.kinesis structs*),
msl.equipment.resources.thorlabs.kinesis.enums, 582

KMOT_TrigDisabled

KLS_ConstantCurrent
(*msl.equipment.resources.thorlabs.kinesis.enums.KLSDOpMode*
attribute), 459

KLS_ConstantPower
(*msl.equipment.resources.thorlabs.kinesis.enums.KLS8OpMode*
attribute), 459

KLS_Disabled (*msl.equipment.resources.thorlabs.kinesis.enums.KLSDTiggerMode*
attribute), 460

KLS_HighStability
(*msl.equipment.resources.thorlabs.kinesis.enums.KLSDTiggerMode*
attribute), 460

KLS_Input (*msl.equipment.resources.thorlabs.kinesis.enums.KMOTTrigKdygeTriggerPolarity*
attribute), 460

KLS_InterlockEnabled
(*msl.equipment.resources.thorlabs.kinesis.enums.KMOTKdygeAbsoluteMove*
attribute), 460

KLS_LaserOn (*msl.equipment.resources.thorlabs.kinesis.enums.KMOTTriggerMode*
attribute), 452

KLS_LowStability
(*msl.equipment.resources.thorlabs.kinesis.enums.KLSDTiggerMode*

KMOT_TrigIn_Home	KNA_Channels	(class in (<i>msl.equipment.resources.thorlabs.kinesis.enums.KMOT_TrigIn_PortMode</i>), attribute), 452	465
KMOT_TrigIn_RelativeMove	KNA_ChannelUndefined	(<i>msl.equipment.resources.thorlabs.kinesis.enums.KMOT_TrigIn_PortMode</i>), attribute), 452	465
KMOT_TrigOut_AtMaxVelocity	KNA_Default_Range	(<i>msl.equipment.resources.thorlabs.kinesis.enums.KMOT_TrigOut_PortMode</i>), attribute), 452	462
KMOT_TrigOut_AtPositionSteps	KNA_Default_Route	(<i>msl.equipment.resources.thorlabs.kinesis.enums.KMOT_TrigOut_PortMode</i>), attribute), 452	463
KMOT_TrigOut_GPO	KNA_EnableInputBoost	(<i>msl.equipment.resources.thorlabs.kinesis.enums.KMOT_TrigOut_PortMode</i>), attribute), 452	463
KMOT_TrigOut_InMotion	KNA_ExtIn_I01	(<i>msl.equipment.resources.thorlabs.kinesis.enums.KMOT_TrigOut_PortMode</i>), attribute), 452	463
KMOT_TrigOut_Synch	KNA_ExtIn_PIN	(<i>msl.equipment.resources.thorlabs.kinesis.enums.KMOT_TrigOut_PortMode</i>), attribute), 452	463
KMOT_TrigPolarityHigh	KNA_EnableLoopPolarity	(<i>msl.equipment.resources.thorlabs.kinesis.enums.KMOT_TrigPortMode</i>), attribute), 452	463
KMOT_TrigPolarityLow	KNA_EnableLoopPriority	(<i>msl.equipment.resources.thorlabs.kinesis.enums.KMOT_TrigPortMode</i>), attribute), 452	463
KMOT_WheelDirectionSense	KNA_FeedbackModeTypes	(class in <i>msl.equipment.resources.thorlabs.kinesis.enums</i>), 451	587
KMOT_WheelMode	KNA_FeedbackSource	(class in <i>msl.equipment.resources.thorlabs.kinesis.enums</i>), 451	465
KMOT_WM_Jog	KNA_HighOutputVoltageRoute	(class in <i>msl.equipment.resources.thorlabs.kinesis.enums</i>), 452	462
KMOT_WM_MoveAbsolute	KNA_HighVoltageRange	(class in <i>msl.equipment.resources.thorlabs.kinesis.enums</i>), 452	462
KMOT_WM_Negative	KNA_I01only	(<i>msl.equipment.resources.thorlabs.kinesis.enums.KMOT_WheelMode</i>), attribute), 451	462
KMOT_WM_Positive	KNA_Io1Wires	(<i>msl.equipment.resources.thorlabs.kinesis.enums.KMOT_WheelMode</i>), attribute), 451	462
KMOT_WM_Velocity	KNA_LowOutputVoltageRoute	(class in <i>msl.equipment.resources.thorlabs.kinesis.enums</i>), 451	462
KNA_Channel1	KNA_Channels	(class in <i>msl.equipment.resources.thorlabs.kinesis.enums</i>), 465	462
KNA_Channel2	KNA_LowVoltageRange	(class in <i>msl.equipment.resources.thorlabs.kinesis.enums</i>), 465	461

KNA_MMIParams	(class in KNA_TrigDisabled <i>msl.equipment.resources.thorlabs.kinesis structs</i>), (msl.equipment.resources.thorlabs.kinesis.enums.KNA_TrigDisabled attribute), 464 586
KNA_TIARange	(class in KNA_TriggerConfig <i>msl.equipment.resources.thorlabs.kinesis.enums</i>), (msl.equipment.resources.thorlabs.kinesis.structs), 587 461
KNA_TIARange10_166uA	KNA_TriggerPortMode (class in <i>msl.equipment.resources.thorlabs.kinesis.enums.KNA_TriggerPortMode</i>), (msl.equipment.resources.thorlabs.kinesis.enums.KNA_TriggerPortMode attribute), 464 461
KNA_TIARange11_500uA	KNA_TriggerPortPolarity (class in <i>msl.equipment.resources.thorlabs.kinesis.enums.KNA_TriggerPortPolarity</i>), (msl.equipment.resources.thorlabs.kinesis.enums.KNA_TriggerPortPolarity attribute), 464 461
KNA_TIARange12_1_66mA	KNA_TrigIn_GPI (msl.equipment.resources.thorlabs.kinesis.enums.KNA_TrigIn_GPI), (msl.equipment.resources.thorlabs.kinesis.enums.KNA_TrigIn_GPI attribute), 464 461
KNA_TIARange13_5mA	KNA_TrigIn_VoltageStepDown (msl.equipment.resources.thorlabs.kinesis.enums.KNA_TrigIn_VoltageStepDown), (msl.equipment.resources.thorlabs.kinesis.enums.KNA_TrigIn_VoltageStepDown attribute), 464 461
KNA_TIARange1_5nA	KNA_TrigIn_VoltageStepUp (msl.equipment.resources.thorlabs.kinesis.enums.KNA_TrigIn_VoltageStepUp), (msl.equipment.resources.thorlabs.kinesis.enums.KNA_TrigIn_VoltageStepUp attribute), 464 461
KNA_TIARange2_16_6nA	KNA_TrigOut_GPO (msl.equipment.resources.thorlabs.kinesis.enums.KNA_TrigOut_GPO), (msl.equipment.resources.thorlabs.kinesis.enums.KNA_TrigOut_GPO attribute), 464 461
KNA_TIARange3_50nA	KNA_TrigPolarityHigh (msl.equipment.resources.thorlabs.kinesis.enums.KNA_TrigPolarityHigh), (msl.equipment.resources.thorlabs.kinesis.enums.KNA_TrigPolarityHigh attribute), 465 461
KNA_TIARange4_166nA	KNA_TrigPolarityLow (msl.equipment.resources.thorlabs.kinesis.enums.KNA_TrigPolarityLow), (msl.equipment.resources.thorlabs.kinesis.enums.KNA_TrigPolarityLow attribute), 465 461
KNA_TIARange5_500nA	KNA_VoltageRange_10v (msl.equipment.resources.thorlabs.kinesis.enums.KNA_VoltageRange_10v), (msl.equipment.resources.thorlabs.kinesis.enums.KNA_VoltageRange_10v attribute), 462 461
KNA_TIARange6_1_66uA	KNA_VoltageRange_CH1_150v (msl.equipment.resources.thorlabs.kinesis.enums.KNA_VoltageRange_CH1_150v), (msl.equipment.resources.thorlabs.kinesis.enums.KNA_VoltageRange_CH1_150v attribute), 462 461
KNA_TIARange7_5uA	KNA_VoltageRange_CH1_75v (msl.equipment.resources.thorlabs.kinesis.enums.KNA_VoltageRange_CH1_75v), (msl.equipment.resources.thorlabs.kinesis.enums.KNA_VoltageRange_CH1_75v attribute), 462 461
KNA_TIARange8_16_6uA	KNA_VoltageRange_CH2_150v (msl.equipment.resources.thorlabs.kinesis.enums.KNA_VoltageRange_CH2_150v), (msl.equipment.resources.thorlabs.kinesis.enums.KNA_VoltageRange_CH2_150v attribute), 462 461
KNA_TIARange9_50uA	KNA_VoltageRange_CH2_75v (msl.equipment.resources.thorlabs.kinesis.enums.KNA_VoltageRange_CH2_75v), (msl.equipment.resources.thorlabs.kinesis.enums.KNA_VoltageRange_CH2_75v attribute), 462 461
KNA_TIARangeParameters	(class in KNA_WheelAdjustRate <i>msl.equipment.resources.thorlabs.kinesis structs</i>), (msl.equipment.resources.thorlabs.kinesis.enums.KNA_WheelAdjustRate attribute), 463 585
KNA_TIAReading	(class in KNA_WM_High(msl.equipment.resources.thorlabs.kinesis.enums.KNA_WM_High <i>msl.equipment.resources.thorlabs.kinesis structs</i>), (attribute), 464 586 KNA_WM_Low(msl.equipment.resources.thorlabs.kinesis.enums.KNA_WM_Low <i>msl.equipment.resources.thorlabs.kinesis structs</i>), (attribute), 464 586

<i>attribute), 464</i>	<i>attribute), 467</i>
KNA_WM_Medium (<i>msl.equipment.resources.thorlabs.KPZ_WM_Medium</i>)	<i>KNA_WM_Medium</i> (<i>msl.equipment.resources.thorlabs.kinesis.enums.KNA_WM_Medium</i>)
<i>attribute), 464</i>	<i>attribute), 467</i>
KPZ_MMIParams	(class <i>in</i> KPZ_WM_MoveAtVoltage
	<i>msl.equipment.resources.thorlabs.kinesis structs), (msl.equipment.resources.thorlabs.kinesis.enums.KPZ_WM_MoveAtVoltage</i>
587	<i>attribute), 466</i>
KPZ_TrigDisabled	KPZ_WM_Negative
	<i>(msl.equipment.resources.thorlabs.kinesis.enums.KPZ_WM_TrigDisabled)</i>
	<i>attribute), 467</i>
KPZ_TriggerConfig	(class <i>in</i> KPZ_WM_Positive
	<i>msl.equipment.resources.thorlabs.kinesis structs), (msl.equipment.resources.thorlabs.kinesis.enums.KPZ_WM_Positive</i>
588	<i>attribute), 466</i>
KPZ_TriggerPortMode	(class <i>in</i> KPZ_WM_SetVoltage
	<i>msl.equipment.resources.thorlabs.kinesis enums), (msl.equipment.resources.thorlabs.kinesis.enums.KPZ_WM_SetVoltage</i>
467	<i>attribute), 466</i>
KPZ_TriggerPortPolarity	(class <i>in</i> KSC_MMIParams) (class <i>in</i>
	<i>msl.equipment.resources.thorlabs.kinesis enums), msl.equipment.resources.thorlabs.kinesis structs),</i>
467	<i>591</i>
KPZ_TrigIn_GPI	KSC_TrigDisabled
	<i>(msl.equipment.resources.thorlabs.kinesis enums.KPZ_TrigIn_GPI)</i>
	<i>attribute), 467</i>
KPZ_TrigIn_VoltageStepDown	KSC_TriggerConfig (class <i>in</i>
	<i>(msl.equipment.resources.thorlabs.kinesis enums.KPZ_TrigIn_VoltageStepDown)</i>
	<i>attribute), 467</i>
KPZ_TrigIn_VoltageStepUp	KSC_TriggerPortMode (class <i>in</i>
	<i>(msl.equipment.resources.thorlabs.kinesis enums.KPZ_TrigIn_VoltageStepUp)</i>
	<i>attribute), 467</i>
KPZ_TrigOut_GPO	KSC_TriggerPortPolarity (class <i>in</i>
	<i>(msl.equipment.resources.thorlabs.kinesis enums.KPZ_TrigOut_GPO)</i>
	<i>attribute), 467</i>
KPZ_TrigPolarityHigh	KSC_TrigIn_GPI
	<i>(msl.equipment.resources.thorlabs.kinesis enums.KPZ_TrigPolarityHigh)</i>
	<i>attribute), 467</i>
KPZ_TrigPolarityLow	KSC_TrigOut_GPO
	<i>(msl.equipment.resources.thorlabs.kinesis enums.KPZ_TrigPolarityLow)</i>
	<i>attribute), 467</i>
KPZ_WheelChangeRate	(class <i>in</i> KSC_TrigPolarityHigh
	<i>msl.equipment.resources.thorlabs.kinesis enums), (msl.equipment.resources.thorlabs.kinesis enums.KSC_TrigPolarityHigh</i>
466	<i>attribute), 472</i>
KPZ_WheelDirectionSense	(class <i>in</i> KSC_TrigPolarityLow
	<i>msl.equipment.resources.thorlabs.kinesis enums), (msl.equipment.resources.thorlabs.kinesis enums.KSC_TrigPolarityLow</i>
466	<i>attribute), 472</i>
KPZ_WheelMode	(class <i>in</i> KSG_MMIParams) (class <i>in</i>
	<i>msl.equipment.resources.thorlabs.kinesis enums), msl.equipment.resources.thorlabs.kinesis structs),</i>
466	<i>592</i>
KPZ_WM_High (<i>msl.equipment.resources.thorlabs.kin</i>	KSG_CTimingDRab_WheelChangeRate
	<i>(msl.equipment.resources.thorlabs.kinesis enums.KSG_CTimingDRab_WheelChangeRate</i>
	<i>attribute), 467</i>
KPZ_WM_JogVoltage	KSG_CTimingDRab_WheelChangeRate (class <i>in</i>
	<i>msl.equipment.resources.thorlabs.kinesis enums.KSG_CTimingDRab_WheelChangeRate</i>
	<i>attribute), 466</i>
KPZ_WM_Low (<i>msl.equipment.resources.thorlabs.kinesis enums.KPZ_WheelChangeRate</i>	KSG_CTimingDRab_WheelChangeRate

KSG_TriggerPortMode (class in LAST_USB (*msl.equipment.resources.picotech.picoscope.ps2000.Picoscope*), *msl.equipment.resources.thorlabs.kinesis.enums*), attribute), 342
473 LAST_USB (*msl.equipment.resources.picotech.picoscope.ps3000.Picoscope*), *msl.equipment.resources.thorlabs.kinesis.enums*, attribute), 345
KSG_TriggerPortPolarity (class in LAST_NotUsed (*msl.equipment.resources.thorlabs.kinesis.structs.MSL_Equipment*), *msl.equipment.resources.thorlabs.kinesis.enums*), attribute), 345
474 lastNotUsed (*msl.equipment.resources.thorlabs.kinesis.structs.MSL_Equipment*), attribute), 575
KSG_TrigIn_GPI (lastNotUsed (*msl.equipment.resources.thorlabs.kinesis.structs.MSL_Equipment*), *msl.equipment.resources.thorlabs.kinesis.enums.KSG_TriggerPortMode*), attribute), 474 lastNotUsed (*msl.equipment.resources.thorlabs.kinesis.structs.MSL_Equipment*), attribute), 574
KSG_TrigOut_BetweenLimits (lastNotUsed (*msl.equipment.resources.thorlabs.kinesis.structs.MSL_Equipment*), *msl.equipment.resources.thorlabs.kinesis.enums.KSG_TriggerPortMode*), attribute), 474 latest_RampUpLimit (*msl.equipment.resources.thorlabs.kinesis.structs.MSL_Equipment*), attribute), 574
KSG_TrigOut_GPO (lastNotUsed (*msl.equipment.resources.thorlabs.kinesis.structs.MSL_Equipment*), *msl.equipment.resources.thorlabs.kinesis.enums.KSG_TriggerPortMode*), attribute), 474 latest_calibration
KSG_TrigOut_LessThanLowerLimit (latest_calibration (*msl.equipment.record_types.EquipmentRecord*), *msl.equipment.resources.thorlabs.kinesis.enums.KSG_TriggerPortMode*), attribute), 474 LD_AnodeGrounded
KSG_TrigOut_LessThanUpperLimit (LD_AnodeGrounded (*msl.equipment.resources.thorlabs.kinesis.enums.LD_POL*), *msl.equipment.resources.thorlabs.kinesis.enums.KSG_TriggerPortMode*), attribute), 474 LD_CathodeGrounded
KSG_TrigOut_MoreThanLowerLimit (LD_CathodeGrounded (*msl.equipment.resources.thorlabs.kinesis.enums.LD_POL*), *msl.equipment.resources.thorlabs.kinesis.enums.KSG_TriggerPortMode*), attribute), 474 LD_DisplayUnits (class in *msl.equipment.resources.thorlabs.kinesis.enums*),
KSG_TrigOut_MoreThanUpperLimit (LD_DisplayUnits (*msl.equipment.resources.thorlabs.kinesis.enums.KSG_TriggerPortMode*), attribute), 474 LD_ExternalSignal
KSG_TrigOut_OutsideLimits (LD_ExternalSignal (*msl.equipment.resources.thorlabs.kinesis.enums.LD_Input*), *msl.equipment.resources.thorlabs.kinesis.enums.KSG_TriggerPortMode*), attribute), 474 LD_ILD (*msl.equipment.resources.thorlabs.kinesis.enums.LD_Displa*), attribute), 457
KSG_TrigPolarityHigh (LD_ILD (*msl.equipment.resources.thorlabs.kinesis.enums.LD_Displa*), *msl.equipment.resources.thorlabs.kinesis.enums.KSG_TriggerPortPolarity*), attribute), 457
KSG_TrigPolarityLow (attribute), 457 LD_InputSourceFlags (class in *msl.equipment.resources.thorlabs.kinesis.enums*),
KST_Stages (LD_InputSourceFlags (*msl.equipment.resources.thorlabs.kinesis.enums.KSG_TriggerPortPolarity*), attribute), 456
472 LD_PLD (*msl.equipment.resources.thorlabs.kinesis.enums.LD_Displa*), attribute), 457
L LD_POLARITY (class in *msl.equipment.resources.thorlabs.kinesis.enums*),
LabJack_050 (attribute), 458 LabJack_050 (*msl.equipment.resources.thorlabs.kinesis.motions*), attribute), 458
attribute), 568 LabJack_490 (*msl.equipment.resources.thorlabs.kinesis.motions*), attribute), 458
attribute), 568 LD_DisplacementMotionControl (attribute), 457
Laser (attribute), 457 LD_DisplacementMotionControl (*msl.equipment.resources.thorlabs.kinesis.enums.LD_Displa*),
module (attribute), 457 msl.equipment.resources.thorlabs.kinesis.motions LD_SoftwareOnly
565 (attribute), 457 last_button_press () (attribute), 457
(msl.equipment.resources.picotech.picoscope), attribute), 343 EDPS2000_EDPS2000 (attribute), 458
method), 343 attribute), 458

LD_TIA_10mA (*msl.equipment.resources.thorlabs.kinesis.enum.LD_TIA_RANGES*)
 attribute), 458
 LD_TIA_10uA (*msl.equipment.resources.thorlabs.kinesis.enum.LD_TIA_RANGES*)
 attribute), 457
 LD_TIA_1mA (*msl.equipment.resources.thorlabs.kinesis.enum.LD_TIA_RANGES*)
 attribute), 458
 LD_TIA_RANGES (class in *LIMIT_MODES* (*msl.equipment.resources.mks_instruments.pr4000b*))
msl.equipment.resources.thorlabs.kinesis.enums), attribute), 151
 457
 limitSwitch (*msl.equipment.resources.thorlabs.kinesis.structs.M*)
 leftButtonPosition
msl.equipment.resources.thorlabs.kinesis.structs.M
msl.equipment.resources.thorlabs.kinesis.enums.M
 attribute), 572
 attribute), 582
 length_payload (*msl.equipment.hislip.Message*)
 lines () (*msl.equipment.connection_gpib.ConnectionGPIB*)
 property), 60
 method), 25
 LESS_THAN (*msl.equipment.resources.picotech.picoscope.enums.PS2000APulseWidthType*)
 attribute), 251
 LESS_THAN (*msl.equipment.resources.picotech.picoscope.enums.PS2000APulseWidthType*)
 attribute), 239
 LESS_THAN (*msl.equipment.resources.picotech.picoscope.enums.PS2000APulseWidthType*)
 attribute), 268
 LESS_THAN (*msl.equipment.resources.picotech.picoscope.enums.PS2000APulseWidthType*)
 attribute), 257
 load_named_settings ()
 LESS_THAN (*msl.equipment.resources.picotech.picoscope.enums.PS4000APulseWidthType*)
 attribute), 292
 method), 414
 LESS_THAN (*msl.equipment.resources.picotech.picoscope.enums.PS4000APulseWidthType*)
 attribute), 279
 LESS_THAN (*msl.equipment.resources.picotech.picoscope.enums.PS5000APulseWidthType*)
 attribute), 313
 load_named_settings ()
 LESS_THAN (*msl.equipment.resources.picotech.picoscope.enums.PS5000APulseWidthType*)
 attribute), 303
 method), 492
 LESS_THAN (*msl.equipment.resources.picotech.picoscope.enums.PS6000APulseWidthType*)
 attribute), 323
 (msl.equipment.resources.thorlabs.kinesis.filter_flipper.Fil
 LEVEL (*msl.equipment.resources.picotech.picoscope.enums.PS2000AThresholdMode*)
 attribute), 248
 load_named_settings ()
 LEVEL (*msl.equipment.resources.picotech.picoscope.enums.PS2000AThresholdMode*)
 attribute), 238
 method), 534
 LEVEL (*msl.equipment.resources.picotech.picoscope.enums.PS2000AThresholdMode*)
 attribute), 266
 (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo
 LEVEL (*msl.equipment.resources.picotech.picoscope.enums.PS2000AThresholdMode*)
 attribute), 256
 load_sdk () (msl.equipment.resources.nkt.nkpdll.NKT
 LEVEL (*msl.equipment.resources.picotech.picoscope.enums.PS4000AThresholdMode*)
 attribute), 289
 load_settings ()
 LEVEL (*msl.equipment.resources.picotech.picoscope.enums.PS4000AThresholdMode*)
 attribute), 277
 method), 414
 LEVEL (*msl.equipment.resources.picotech.picoscope.enums.PS5000AThresholdMode*)
 attribute), 310
 (msl.equipment.resources.thorlabs.kinesis.filter_flipper.Fil
 LEVEL (*msl.equipment.resources.picotech.picoscope.enums.PS5000AThresholdMode*)
 attribute), 301
 load_settings ()
 LEVEL (*msl.equipment.resources.picotech.picoscope.enums.PS6000AThresholdMode*)
 attribute), 320
 method), 492
 LEVEL_TRIGGER_SOURCE
 load_settings ()

(*msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo*.**method**), 515
load_settings() (*msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo*.**method**), 534
load_settings() (*msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo*.**attribute**), 568
load_settings() (*msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo*.**attribute**), 581
LOST_DATA (*msl.equipment.resources.picotech.picoscope.ps2000.PS2000A*.**method**), 128
LOST_DATA (*msl.equipment.resources.picotech.picoscope.ps3000.PS3000A*.**method**), 128
load_setup() (*msl.equipment.resources.bentham.benhw32.Bentham*), 2346
load_setup() (*msl.equipment.resources.bentham.benhw32.Bentham*).**method**, 128
load_setup() (*msl.equipment.resources.bentham.benhw64.Bentham*), 350
load_standard_file() (*msl.equipment.resources.optronic_laboratory*.**method**), 216
local() (*msl.equipment.connection_gpib.Connection*).**method**, 26
local() (*msl.equipment.connection_tcpip_vxi11.Connection*).**method**, 47
LocalIp (*msl.equipment.resources.avantes.avaspec.AVANTES*.**attribute**), 110
LocalIp (*msl.equipment.resources.avantes.avaspec.BorrowedLimits*.**attribute**), 99
lock() (*msl.equipment.connection_tcpip_hislip.Connection*).**method**, 44
lock() (*msl.equipment.connection_tcpip_yxi11.Connection*).**method**, 47
lock() (*msl.equipment.resources.mks_instruments.pr4000b.PR4000B*).**method**, 157
lock_status() (*msl.equipment.connection_tcpip_hislip.Connection*).**method**, 44
lock_timeout() (*msl.equipment.connection_tcpip_hislip.Connection*).**property**, 44
lock_timeout() (*msl.equipment.connection_tcpip_yxi11.Connection*).**property**, 47
log_critical() (*msl.equipment.connection.Connection*).**static method**, 21
log_debug() (*msl.equipment.connection.Connection*).**static method**, 20
log_errcheck() (*msl.equipment.connection_sdk.Connection*).**method**, 39
log_error() (*msl.equipment.connection.Connection*).**static method**, 21
log_info() (*msl.equipment.connection.Connection*).**static method**, 20
log_warning() (*msl.equipment.connection.Connection*).**static method**, 20
LONG_PRESS (*msl.equipment.resources.picotech.picoscope.enums.PS2000B*).**attribute**, 237
lowVoltageOutputRoute (*msl.equipment.resources.thorlabs.kinesis.KNA_IOC*.**attribute**), 586
lowVoltageOutputRoute (*msl.equipment.resources.thorlabs.kinesis.KNA_IOC*.**attribute**), 586
lowVoltageOutputRoute (*msl.equipment.resources.thorlabs.kinesis.QD_Lock*.**attribute**), 589
lowPassFilterEnabled (*msl.equipment.resources.thorlabs.kinesis.QD_Lock*.**attribute**), 588
lowPassFilterCutOffFreq (*msl.equipment.resources.thorlabs.kinesis.QD_Lock*.**attribute**), 589
lowVoltageOutputRoute (*msl.equipment.resources.thorlabs.kinesis.KNA_IOC*.**attribute**), 586
lowVoltageOutputRoute (*msl.equipment.resources.thorlabs.kinesis.QD_PowerSupply*.**attribute**), 586
lowVoltageOutputRoute (*msl.equipment.resources.thorlabs.kinesis.QD_PowerSupply*.**attribute**), 586
lowVoltageOutRange (*msl.equipment.resources.thorlabs.kinesis.QD_PowerSupply*.**attribute**), 586

<code>(msl.equipment.resources.thorlabs.kinesis.structs.KNAndIOSetup::getattribute),</code>	<code>586</code>	<code>m_aHighNLCounts</code>	<code>(msl.equipment.resources.avantes.avaspec.Avantes.DetectorType::getattribute),</code>	<code>111</code>
<code>lowVoltageOutRange</code>		<code>m_aHighNLCounts</code>	<code>(msl.equipment.resources.thorlabs.kinesis.structs.NT::getattribute),</code>	<code>101</code>
<code>(msl.equipment.resources.thorlabs.kinesis.enums),</code>			<code>(msl.equipment.resources.avantes.avaspec.Avantes.DetectorType::getattribute),</code>	<code>101</code>
<code>LS_DisplayUnits</code>	<code>(class in m_aLowNLCounts</code>			
<code>msl.equipment.resources.thorlabs.kinesis.enums),</code>			<code>(msl.equipment.resources.avantes.avaspec.Avantes.DetectorType::getattribute),</code>	<code>111</code>
<code>475</code>				
<code>LS_ExternalSignal</code>		<code>m_aLowNLCounts</code>		
<code>(msl.equipment.resources.thorlabs.kinesis.enums.LSImpl::getattribute),</code>	<code>459</code>		<code>(msl.equipment.resources.avantes.avaspec.Avantes.DetectorType::getattribute),</code>	<code>101</code>
<code>LS_InputSourceFlags</code>	<code>(class in m_AnalogHigh(msl.equipment.resources.avantes.avaspec.Avantes.DetectorType::getattribute),</code>			
<code>msl.equipment.resources.thorlabs.kinesis.enums),</code>			<code>114</code>	
<code>459</code>			<code>m_AnalogHigh(msl.equipment.resources.avantes.avaspec.ProcessorType::getattribute),</code>	<code>101</code>
<code>LS_mAmps</code>	<code>(msl.equipment.resources.thorlabs.kinesis.enums.LSDisplayUnits::getattribute),</code>			
<code>476</code>		<code>m_AnalogLow(msl.equipment.resources.avantes.avaspec.Avantes.ProcessorType::getattribute),</code>		
<code>LS_mDbs</code>	<code>(msl.equipment.resources.thorlabs.kinesis.enums.LSDisplayUnits::getattribute),</code>			
<code>476</code>		<code>m_AnalogLow(msl.equipment.resources.avantes.avaspec.ProcessorType::getattribute),</code>		
<code>LS_mWatts</code>	<code>(msl.equipment.resources.thorlabs.kinesis.enums.LSDisplayUnits::getattribute),</code>			
<code>476</code>		<code>m_aNLCorrect(msl.equipment.resources.avantes.avaspec.Avantes.ProcessorType::getattribute),</code>		
<code>LS_Potentiometer</code>	<code>(msl.equipment.resources.thorlabs.kinesis.enums.mslEquipment::getattribute),</code>			
<code>459</code>		<code>m_aNLCorrect(msl.equipment.resources.avantes.avaspec.DetectorType::getattribute),</code>		
<code>LS_SoftwareOnly</code>	<code>(msl.equipment.resources.thorlabs.kinesis.enums.LSDisplayUnits::getattribute),</code>			
<code>459</code>		<code>m_aReserved(msl.equipment.resources.avantes.avaspec.Avantes.DeviceConfigType::getattribute),</code>		
<code>LUTdiameter</code>	<code>(msl.equipment.resources.thorlabs.kinesis.structsNT::getattribute),</code>			
<code>578</code>		<code>m_aSpectrumCorrect</code>		
<code>LUTValueDelay</code>	<code>(msl.equipment.resources.thorlabs.kinesis.structsNT::getattribute),</code>			
<code>579</code>		<code>m_aSpectrumCorrect</code>		
M			<code>(msl.equipment.resources.avantes.avaspec.SpectrumCorrect::getattribute),</code>	
<code>m_aCalibConvers</code>	<code>(msl.equipment.resources.avantes.avaspec.AvantTempCalibrationType::getattribute),</code>			
<code>112</code>		<code>m_aUserFriendlyId</code>		
<code>m_aCalibConvers</code>	<code>(msl.equipment.resources.avantes.avaspec.SpecTempCalibrationType::getattribute),</code>			
<code>101</code>		<code>(msl.equipment.resources.avantes.avaspec.DeviceConfigType::getattribute),</code>		
<code>m_aFit</code>	<code>(msl.equipment.resources.avantes.avaspec.Avantes.DetectorType::getattribute),</code>			
<code>111</code>		<code>m_aUserFriendlyId</code>		
<code>m_aFit</code>	<code>(msl.equipment.resources.avantes.avaspec.TecControlType::getattribute),</code>			
<code>114</code>		<code>(msl.equipment.resources.avantes.avaspec.DeviceConfigType::getattribute),</code>		
<code>m_aFit</code>	<code>(msl.equipment.resources.avantes.avaspec.AvantUserTempFriendlyType::getattribute),</code>			
<code>114</code>		<code>(msl.equipment.resources.avantes.avaspec.DeviceConfigType::getattribute),</code>		
<code>m_aFit</code>	<code>(msl.equipment.resources.avantes.avaspec.DetectorType::getattribute),</code>			
<code>101</code>		<code>m_BitMatrix(msl.equipment.resources.avantes.avaspec.Avantes.IrradianceType::getattribute),</code>		
<code>m_aFit</code>	<code>(msl.equipment.resources.avantes.avaspec.TecControlType::getattribute),</code>			
<code>104</code>		<code>m_BitMatrix(msl.equipment.resources.avantes.avaspec.HeartbeatType::getattribute),</code>		
<code>m_aFit</code>	<code>(msl.equipment.resources.avantes.avaspec.TempSensorType::getattribute),</code>			
<code>104</code>		<code>m_CalibrationType</code>		
<code>m_aHighNLCounts</code>			<code>(msl.equipment.resources.avantes.avaspec.Avantes.IrradianceType::getattribute),</code>	

attribute), 112
m_CalibrationType
(msl.equipment.resources.avantes.avaspec.IrradianceType
attribute), 101
m_CalInttime (msl.equipment.resources.avantes.avaspec.DynamicStorage
attribute), 112
m_CalInttime (msl.equipment.resources.avantes.avaspec.SpectralCalibrationType
attribute), 101
m_ConfigVersion
(msl.equipment.resources.avantes.avaspec.AvanteConfigType
attribute), 115
m_ConfigVersion
(msl.equipment.resources.avantes.avaspec.DeviceConfigType
attribute), 105
m_Control (msl.equipment.resources.avantes.avaspec.AvanteMeasConfigType
attribute), 112
m_Control (msl.equipment.resources.avantes.avaspec.MeasConfigType), 103
m_CorDynDark (msl.equipment.resources.avantes.avaspec.AvanteMeasConfigType
attribute), 112
m_CorDynDark (msl.equipment.resources.avantes.avaspec.MeasConfigType
attribute), 102
m_data (msl.equipment.resources.avantes.avaspec.AvanteEthernetSettings
attribute), 115
m_data (msl.equipment.resources.avantes.avaspec.OemDataType
attribute), 105
m_Date (msl.equipment.resources.avantes.avaspec.TimeStampType1
attribute), 113
m_Date (msl.equipment.resources.avantes.avaspec.TimeStampType1
attribute), 103
m_DefectivePixels
(msl.equipment.resources.avantes.avaspec.DetectorType)
attribute), 111
m_DefectivePixels
(msl.equipment.resources.avantes.avaspec.DetectorType
attribute), 100
m_Detector (msl.equipment.resources.avantes.avaspec.AvanteDeviceConfigType
attribute), 115
m_Detector (msl.equipment.resources.avantes.avaspec.meDefragPercentageType
attribute), 105
m_DhcpEnabled (msl.equipment.resources.avantes.avaspec.AvanteEthernetSettingsType
attribute), 114
m_DhcpEnabled (msl.equipment.resources.avantes.avaspec.EthernetSettingsType
attribute), 104
m_DigitalHigh (msl.equipment.resources.avantes.avaspec.AvanteProtocolControlType
attribute), 114
m_DigitalHigh (msl.equipment.resources.avantes.avaspec.ProtocolControlType
attribute), 104
m_DigitalLow (msl.equipment.resources.avantes.avaspec.AvanteProtocolControlType
attribute), 114
m_DigitalLow (msl.equipment.resources.avantes.avaspec.ProtocolControlType
attribute), 104
attribute), 104
m_DynamicStorage
attribute), 115
m_DynamicStorage
attribute), 104
m_Enable (msl.equipment.resources.avantes.avaspec.Avantes.Dark
attribute), 110
m_Enable (msl.equipment.resources.avantes.avaspec.AvanteStandAloneType
attribute), 113
m_Enable (msl.equipment.resources.avantes.avaspec.TecControlType
attribute), 114
m_Enable (msl.equipment.resources.avantes.avaspec.DarkCorrect
attribute), 105
m_Enable (msl.equipment.resources.avantes.avaspec.StandAloneType
attribute), 112
m_Enable (msl.equipment.resources.avantes.avaspec.TecControlType
attribute), 102
m_EthernetSettings
attribute), 115
m_EthernetSettings
attribute), 115
m_EthernetSettings
attribute), 115
m_EthernetSettings
attribute), 115
m_FiberDiameter
attribute), 12
m_FiberDiameter
attribute), 101
m_FiberDiameter
attribute), 101
m_ForgetPercentage
attribute), 110
m_Gain (msl.equipment.resources.avantes.avaspec.Avantes.DarkCorr
attribute), 110
m_Gain (msl.equipment.resources.avantes.avaspec.DetectorType
attribute), 100
m_Gateway (msl.equipment.resources.avantes.avaspec.Avantes.Ether
attribute), 100
m_Gateway (msl.equipment.resources.avantes.avaspec.EthernetSett
attribute), 100
m_IntegrationDelay
attribute), 100
m_ProgressEquipmentType
resources.avantes.avaspec.Avantes.MeasC

attribute), 113
m_IntegrationDelay
(msl.equipment.resources.avantes.avaspec.MeasConfigType), 102
attribute), 102
m_IntegrationTime
(msl.equipment.resources.avantes.avaspec.AnalogInputType), 113
attribute), 113
m_IntegrationTime
(msl.equipment.resources.avantes.avaspec.MeasConfigType), 111
attribute), 102
m_IntensityCalib
(msl.equipment.resources.avantes.avaspec.AnalogInputType), 112
attribute), 112
m_IntensityCalib
(msl.equipment.resources.avantes.avaspec.IrradianceType), 113
attribute), 102
m_IpAddr
(msl.equipment.resources.avantes.avaspec.Avantes.EthernetSettingsType)
attribute), 114
m_IpAddr
(msl.equipment.resources.avantes.avaspec.EthernetSettingsType03)
attribute), 104
m_Irradiance
(msl.equipment.resources.avantes.avaspec.AvantesDeviceConfigType)
attribute), 115
m_Irradiance
(msl.equipment.resources.avantes.avaspec.DevConfigType)
attribute), 105
m_LaserDelay
(msl.equipment.resources.avantes.avaspec.ControlSettingsType)
attribute), 110
m_LaserDelay
(msl.equipment.resources.avantes.avaspec.ControlSettingsType)
attribute), 100
m_LaserWaveLength
(msl.equipment.resources.avantes.avaspec.AnalogDataSettingsType)
attribute), 110
m_LaserWaveLength
(msl.equipment.resources.avantes.avaspec.ControlSettingsType), 111
attribute), 100
m_LaserWidth
(msl.equipment.resources.avantes.avaspec.ControlSettingsType)
attribute), 110
m_LaserWidth
(msl.equipment.resources.avantes.avaspec.ControlSettingsType)
attribute), 100
m_Len
(msl.equipment.resources.avantes.avaspec.AvantesDeviceConfigType)
attribute), 115
m_Len
(msl.equipment.resources.avantes.avaspec.DeviceConfigType), 105
attribute), 105
m_LinkStatus
(msl.equipment.resources.avantes.avaspec.AvantesEthernetSettingsType)
attribute), 114
m_LinkStatus
(msl.equipment.resources.avantes.avaspec.EthernetSettingsType)
attribute), 104
m_Meas
(msl.equipment.resources.avantes.avaspec.AvantesStandAloneType)
attribute), 113
m_Meas
(msl.equipment.resources.avantes.avaspec.StandAloneType), 114
attribute), 103
m_Mode
(msl.equipment.resources.avantes.avaspec.TriggerType03), 115

<code>(msl.equipment.resources.optronic_laboratory.ol7560rribid2).OL756</code>	<code>MAX(msl.equipment.picotech.picoscope.enums.PS2000ASource, attribute)</code> , 245
<code>manual_get_pmt_voltage()</code>	<code>MAX(msl.equipment.picotech.picoscope.enums.PS2000ATime, attribute)</code> , 245
<code>(msl.equipment.resources.optronic_laboratories.ol7560rribid2).OL756</code>	<code>MAX(msl.equipment.picotech.picoscope.enums.PS2000ATime, attribute)</code> , 245
<code>manual_get_settling_time()</code>	<code>MAX(msl.equipment.picotech.picoscope.enums.PS2000AWaveform, attribute)</code> , 246
<code>(msl.equipment.resources.optronic_laboratory.ol7560rribid2).OL756</code>	<code>MAX(msl.equipment.picotech.picoscope.enums.PS2000AWaveform, attribute)</code> , 246
<code>manual_get_signal()</code>	<code>MAX(msl.equipment.picotech.picoscope.enums.PS2000Data, attribute)</code> , 246
<code>(msl.equipment.resources.optronic_laboratories.ol7560rribid2).OL756</code>	<code>MAX(msl.equipment.picotech.picoscope.enums.PS2000Data, attribute)</code> , 246
<code>manual_home_ol756()</code>	<code>MAX(msl.equipment.picotech.picoscope.enums.PS2000Source, attribute)</code> , 236
<code>(msl.equipment.resources.optronic_laboratory.ol7560rribid2).OL756</code>	<code>MAX(msl.equipment.picotech.picoscope.enums.PS2000Source, attribute)</code> , 237
<code>manual_move_to_wavelength()</code>	<code>MAX(msl.equipment.picotech.picoscope.enums.PS2000Time, attribute)</code> , 245
<code>(msl.equipment.resources.optronic_laboratories.ol7560rribid2).OL756</code>	<code>MAX(msl.equipment.picotech.picoscope.enums.PS2000Time, attribute)</code> , 245
<code>manual_set_gain()</code>	<code>MAX(msl.equipment.picotech.picoscope.enums.PS2000Trigger, attribute)</code> , 235
<code>(msl.equipment.resources.optronic_laboratory.ol7560rribid2).OL756</code>	<code>MAX(msl.equipment.picotech.picoscope.enums.PS2000Trigger, attribute)</code> , 239
<code>manual_set_integration_time()</code>	<code>MAX(msl.equipment.picotech.picoscope.enums.PS3000ACount, attribute)</code> , 245
<code>(msl.equipment.resources.optronic_laboratories.ol7560rribid2).OL756</code>	<code>MAX(msl.equipment.picotech.picoscope.enums.PS3000ACount, attribute)</code> , 245
<code>manual_set_pmt_overload()</code>	<code>MAX(msl.equipment.picotech.picoscope.enums.PS3000AOverload, attribute)</code> , 267
<code>(msl.equipment.resources.optronic_laboratory.ol7560rribid2).OL756</code>	<code>MAX(msl.equipment.picotech.picoscope.enums.PS3000AOverload, attribute)</code> , 259
<code>manual_set_pmt_voltage()</code>	<code>MAX(msl.equipment.picotech.picoscope.enums.PS3000AVoltage, attribute)</code> , 245
<code>(msl.equipment.resources.optronic_laboratories.ol7560rribid2).OL756</code>	<code>MAX(msl.equipment.picotech.picoscope.enums.PS3000AVoltage, attribute)</code> , 245
<code>manual_set_settling_time()</code>	<code>MAX(msl.equipment.picotech.picoscope.enums.PS3000AWaveform, attribute)</code> , 269
<code>(msl.equipment.resources.optronic_laboratory.ol7560rribid2).OL756</code>	<code>MAX(msl.equipment.picotech.picoscope.enums.PS3000AWaveform, attribute)</code> , 265
<code>manufacturer(msl.equipment.record_types.ConnectedEquipment, attribute)</code>	<code>MAX(msl.equipment.picotech.picoscope.enums.PS3000ASource, attribute)</code> , 263
<code>manufacturer(msl.equipment.record_types.Equipment, attribute)</code>	<code>MAX(msl.equipment.picotech.picoscope.enums.PS3000ATime, attribute)</code> , 263
<code>mapped_logging()</code>	<code>MAX(msl.equipment.picotech.picoscope.enums.PS3000ATime, attribute)</code> , 268
<code>(msl.equipment.resources.bentham.benhw32.Bentham, attribute)</code>	<code>MAX(msl.equipment.picotech.picoscope.enums.PS3000AWaveform, attribute)</code> , 268
<code>method), 129</code>	<code>MAX(msl.equipment.picotech.picoscope.enums.PS3000AWaveform, attribute)</code> , 268
<code>MARK (msl.equipment.constants.Parity, attribute), 50</code>	<code>MAX(msl.equipment.picotech.picoscope.enums.PS3000Error, attribute)</code> , 264
<code>MAX(msl.equipment.picotech.picoscope.enums.PS2000ABitmask, attribute), 240</code>	<code>MAX(msl.equipment.picotech.picoscope.enums.PS3000Event, attribute)</code> , 264
<code>MAX(msl.equipment.picotech.picoscope.enums.PS2000ADigitalDirection, attribute), 250</code>	<code>MAX(msl.equipment.picotech.picoscope.enums.PS3000Event, attribute)</code> , 264
<code>MAX(msl.equipment.picotech.picoscope.enums.PS2000AEinsMode, attribute), 244</code>	<code>MAX(msl.equipment.picotech.picoscope.enums.PS3000Event, attribute)</code> , 264
<code>MAX(msl.equipment.picotech.picoscope.enums.PS2000AHabit, attribute), 251</code>	<code>MAX(msl.equipment.picotech.picoscope.enums.PS3000Event, attribute)</code> , 264
<code>MAX(msl.equipment.picotech.picoscope.enums.PS2000ABitmask, attribute), 248</code>	<code>MAX(msl.equipment.picotech.picoscope.enums.PS4000AClock, attribute)</code> , 264

attribute), 282
 MAX (msl.equipment.resources.picotech.picoscope.enums.PS5000Ti
 attribute), 284
 MAX (msl.equipment.resources.picotech.picoscope.enums.PS4000AEinsMode), 209
 attribute), 288
 MAX (msl.equipment.resources.picotech.picoscope.enums.PS4000ABmode), Mode
 attribute), 285
 MAX (msl.equipment.resources.picotech.picoscope.enums.PS4000ABmedrType
 attribute), 285
 MAX (msl.equipment.resources.picotech.picoscope.enums.PS4000ATimeUnits
 attribute), 284
 MAX (msl.equipment.resources.picotech.picoscope.enums.PS6000C
 attribute), 284
 MAX (msl.equipment.resources.picotech.picoscope.enums.PS6000ET
 attribute), 284
 MAX (msl.equipment.resources.picotech.picoscope.enums.PS6000ITriggerState
 attribute), 290
 MAX (msl.equipment.resources.picotech.picoscope.enums.PS6000IWateType
 attribute), 286
 MAX (msl.equipment.resources.picotech.picoscope.enums.PS6000IBufferIndex
 attribute), 269
 MAX (msl.equipment.resources.picotech.picoscope.enums.PS6000IBMode), 217
 attribute), 273
 MAX (msl.equipment.resources.picotech.picoscope.enums.PS6000ITimeMode
 attribute), 276
 MAX (msl.equipment.resources.picotech.picoscope.enums.PS6000W
 attribute), 279
 MAX (msl.equipment.resources.picotech.picoscope.enums.PS4000Rb4000HoldOffType
 attribute), 279
 MAX (msl.equipment.resources.picotech.picoscope.enums.PS4000SwaptType), resources.picotech.picoscope.enums.PS4000Sw
 attribute), 274
 MAX (msl.equipment.resources.picotech.picoscope.enums.PS4000TtriggerState
 attribute), 274
 MAX (msl.equipment.resources.picotech.picoscope.enums.PS4000VaqetType), resources.picotech.picoscope.ps2000a.Pic
 attribute), 275
 MAX (msl.equipment.resources.picotech.picoscope.enums.PS4000VmaxOffsetType), 209
 attribute), 306
 MAX (msl.equipment.resources.picotech.picoscope.enums.PS5000AEinsMode), 248
 attribute), 307
 MAX (msl.equipment.resources.picotech.picoscope.enums.PS5000AflexMode), resources.picotech.picoscope.ps4000a.Pic
 attribute), 310
 MAX (msl.equipment.resources.picotech.picoscope.enums.PS5000AflexOffsetType), 209
 attribute), 308
 MAX (msl.equipment.resources.picotech.picoscope.enums.PS5000ATimeUnits
 attribute), 308
 MAX (msl.equipment.resources.picotech.picoscope.enums.PS5000ATimeUnits
 attribute), 308
 MAX (msl.equipment.resources.picotech.picoscope.enums.PS5000ATriggerState), resources.picotech.picoscope.ps6000.Pico
 attribute), 312
 MAX (msl.equipment.resources.picotech.picoscope.enums.PS5000ATriggerState), resources.picotech.picoscope.ps2000a.Pic
 attribute), 309
 MAX (msl.equipment.resources.picotech.picoscope.enums.PS5000IBufferIndex
 attribute), 297
 MAX (msl.equipment.resources.picotech.picoscope.enums.PS5000IBMode), 214
 attribute), 298
 MAX (msl.equipment.resources.picotech.picoscope.enums.PS5000IBMode), 214
 attribute), 348
 MAX (msl.equipment.resources.picotech.picoscope.enums.PS5000IVmaxOffsetType), 209
 attribute), 301
 MAX (msl.equipment.resources.picotech.picoscope.enums.PS5000IVmaxOffsetType), 209
 attribute), 352
 MAX (msl.equipment.resources.picotech.picoscope.enums.PS5000IVmaxOffsetType), 209
 attribute), 299
 MAX (msl.equipment.resources.picotech.picoscope.enums.PS5000IVmaxOffsetType), 209
 attribute), 348

(<i>msl.equipment.resources.picotech.picoscope.ps5000attribute</i>), 355	<i>ps5000A</i>
	MAX_DELAY_COUNT
MAX_ANALOGUE_OFFSET_50MV_200MV	(<i>msl.equipment.resources.picotech.picoscope.ps5000.PicoScope25000A</i>)
(<i>msl.equipment.resources.picotech.picoscope.ps6000attribute</i>), 356	<i>PicoScope26000</i>
	MAX_DELAY_COUNT
MAX_ANALOGUE_OFFSET_5V_20V	(<i>msl.equipment.resources.picotech.picoscope.ps5000a.PicoScope25000A</i>)
(<i>msl.equipment.resources.picotech.picoscope.ps2000attribute</i>), 344	<i>PicoScope22000A</i>
	MAXETS_CYCLES
MAX_ANALOGUE_OFFSET_5V_20V	(<i>msl.equipment.resources.picotech.picoscope.ps6000a.PicoScope25000A</i>)
(<i>msl.equipment.resources.picotech.picoscope.ps3000attribute</i>), 348	<i>PicoScope23000A</i>
	MAXETS_CYCLES_INTERLEAVE_RATIO
MAX_ANALOGUE_OFFSET_5V_20V	(<i>msl.equipment.resources.picotech.picoscope.ps2000a.PicoScope24000A</i>)
(<i>msl.equipment.resources.picotech.picoscope.ps4000attribute</i>), 352	<i>PicoScope24000A</i>
	MAXETS_CYCLES_INTERLEAVE_RATIO
MAX_ANALOGUE_OFFSET_5V_20V	(<i>msl.equipment.resources.picotech.picoscope.ps3000a.PicoScope25000A</i>)
(<i>msl.equipment.resources.picotech.picoscope.ps5000attribute</i>), 355	<i>PicoScope25000A</i>
	MAX_EXTRA_RESISTANCES
MAX_ANALOGUE_OFFSET_5V_20V	(<i>msl.equipment.resources.picotech.picoscope.enums.PS4000Attribute</i>)
(<i>msl.equipment.resources.picotech.picoscope.ps6000attribute</i>), 357	<i>PicoScope26000</i>
	MAX_HOLDOFF_COUNT
max_channel_count()	(<i>msl.equipment.resources.thorlabs.kinesis.benchtop.BenchtopStepperMotor</i>)
(<i>msl.equipment.resources.thorlabs.kinesis.benchtop_attribute</i>), 415	<i>BenchtopStepperMotor</i>
	MAX_INTERLEAVE
MAX_CHANNELS (<i>msl.equipment.resources.picotech.picoscope.enums.PS2000AChannel</i>)	(<i>msl.equipment.resources.picotech.picoscope.ps6000Attribute</i>)
(<i>msl.equipment.resources.picotech.picoscope.attribute</i>), 240	<i>PS2000AChannel</i>
	MAX_ATTRIBUTE
MAX_CHANNELS (<i>msl.equipment.resources.picotech.picoscope.enums.PS2000AChannel</i>)	(<i>msl.equipment.resources.picotech.picoscope.ps2000a.Pic</i>)
(<i>msl.equipment.resources.picotech.picoscope.attribute</i>), 233	<i>PS2000AChannel</i>
	MAX_LOGIC_LEVEL
MAX_CHANNELS (<i>msl.equipment.resources.picotech.picoscope.enums.PS3000AChannel</i>)	(<i>msl.equipment.resources.picotech.picoscope.ps3000a.Pic</i>)
(<i>msl.equipment.resources.picotech.picoscope.attribute</i>), 252	<i>PS3000AChannel</i>
	MAX_ATTRIBUTE
MAX_CHANNELS (<i>msl.equipment.resources.picotech.picoscope.enums.PS4000AChannel</i>)	(<i>msl.equipment.resources.picotech.picoscope.ps4000a.Pic</i>)
(<i>msl.equipment.resources.picotech.picoscope.attribute</i>), 281	<i>PS4000AChannel</i>
	MAX_ATTRIBUTE
MAX_CHANNELS (<i>msl.equipment.resources.picotech.picoscope.enums.PS4000AChannel</i>)	(<i>msl.equipment.resources.picotech.picoscope.ps2000a.Pic</i>)
(<i>msl.equipment.resources.picotech.picoscope.attribute</i>), 270	<i>PS4000AChannel</i>
	MAX_OVERSAMPLE
MAX_CHANNELS (<i>msl.equipment.resources.picotech.picoscope.enums.PS5000AChannel</i>)	(<i>msl.equipment.resources.picotech.picoscope.ps2000a.Pic</i>)
(<i>msl.equipment.resources.picotech.picoscope.attribute</i>), 306	<i>PS5000AChannel</i>
	MAX_ATTRIBUTE
MAX_CHANNELS (<i>msl.equipment.resources.picotech.picoscope.enums.PS5000AChannel</i>)	(<i>msl.equipment.resources.picotech.picoscope.ps3000a.Pic</i>)
(<i>msl.equipment.resources.picotech.picoscope.attribute</i>), 296	<i>PS5000AChannel</i>
	MAX_ATTRIBUTE
MAX_CHANNELS (<i>msl.equipment.resources.picotech.picoscope.enums.PS6000AChannel</i>)	(<i>msl.equipment.resources.picotech.picoscope.ps3000a.Pic</i>)
(<i>msl.equipment.resources.picotech.picoscope.attribute</i>), 315	<i>PS6000AChannel</i>
	MAX_ATTRIBUTE
MAX_CLIENTS (<i>msl.equipment.hislip.ErrorType</i>)	(<i>msl.equipment.resources.picotech.picoscope.ps3000a.Pic</i>)
(<i>msl.equipment.resources.picotech.picoscope.attribute</i>), 347	<i>ErrorType</i>
	MAX_ATTRIBUTE
MAX_DATA_TYPES	(<i>msl.equipment.resources.picotech.picoscope.ps4000a.Pic</i>)
(<i>msl.equipment.resources.picotech.pt104.Pt104DataType</i>), 349	<i>DataTypes</i>
(<i>msl.equipment.resources.picotech.picoscope.attribute</i>), 371	MAX_ATTRIBUTE
	MAX_OVERSAMPLE_8BIT
MAX_DELAY_COUNT	(<i>msl.equipment.resources.picotech.picoscope.ps4000a.Pic</i>)
(<i>msl.equipment.resources.picotech.picoscope.ps4000attribute</i>), 350	<i>PicoScope24000</i>
	MAX_ATTRIBUTE
	MAX_OVERSAMPLE_8BIT
MAX_DELAY_COUNT	(<i>msl.equipment.resources.picotech.picoscope.ps5000a.Pic</i>)
(<i>msl.equipment.resources.picotech.picoscope.ps4000attribute</i>), 350	<i>PicoScope25000A</i>

MAX_OVERSAMPLE_8BIT	MAX_SIG_GEN_BUFFER_SIZE
(<i>msl.equipment.resources.picotech.picoscope.ps6000</i> (<i>PicoScope6000</i> resources.picotech.picoscope.ps5000.PicoScope6000 attribute), 356	attribute), 353
MAX_PROBES (<i>msl.equipment.resources.picotech.picoscope.ps6000</i> .MAXSIGGENBUFFERSIZE attribute), 272	(<i>msl.equipment.resources.picotech.picoscope.ps6000</i> .Pic attribute), 356
MAX_PULSE_WIDTH	MAXSIGGENFREQ
(<i>msl.equipment.resources.thorlabs.kinesis.filMaxSigGenFreq</i> attribute), 478	(<i>msl.equipment.resources.picotech.picoscope.ps2000a.Pico attribute), 344</i>
MAX_PULSE_WIDTH_QUALIFIER_COUNT	MAXSIGGENFREQ3000
(<i>msl.equipment.resources.picotech.picoscope.ps3000a.Pico attribute), 346</i>	(<i>msl.equipment.resources.picotech.picoscope.ps3000a.Pico attribute), 347</i>
MAX_PULSE_WIDTH_QUALIFIER_COUNT	MAXSIGGENFREQ4000
(<i>msl.equipment.resources.picotech.picoscope.ps4000.Pico attribute), 350</i>	(<i>msl.equipment.resources.picotech.picoscope.ps4000.Pico attribute), 350</i>
MAX_PULSE_WIDTH_QUALIFIER_COUNT	MAXSIGGENFREQ4000A
(<i>msl.equipment.resources.picotech.picoscope.ps5000</i> (<i>Pico attribute), 352</i>	(<i>msl.equipment.resources.picotech.picoscope.ps2000.Pico attribute), 350</i>
MAX_PULSE_WIDTH_QUALIFIER_COUNT	MAXSIGGENFREQ5000
(<i>msl.equipment.resources.picotech.picoscope.ps5000A</i> (<i>Pico attribute), 353</i>	(<i>msl.equipment.resources.picotech.picoscope.ps2000a.Pico attribute), 343</i>
MAX_PULSE_WIDTH_QUALIFIER_COUNT	MAXSWEEPSHOTS3000
(<i>msl.equipment.resources.picotech.picoscope.ps2000a.Pico attribute), 356</i>	(<i>msl.equipment.resources.picotech.picoscope.ps2000a.Pico attribute), 346</i>
MAX_RANGES (<i>msl.equipment.resources.picotech.picoscope.enMaxSweepsShots</i> attribute), 271	MAX_SWEEPS_SHOTS
max_read_size (<i>msl.equipment.connection_message_based</i> . <i>OnMessageReceived</i> attribute), 30	(<i>msl.equipment.resources.picotech.picoscope.ps3000a.Pico attribute), 348</i>
max_read_size (<i>msl.equipment.connection_prolog</i> . <i>OnSweepStart</i> attribute), 34	MAX_SWEEPS_SHOTS
max_read_size (<i>msl.equipment.connection_tcpip_hislip</i> . <i>OnSweepEnd</i> attribute), 44	(<i>msl.equipment.resources.picotech.picoscope.ps4000.Pico attribute), 352</i>
max_read_size (<i>msl.equipment.connection_zeromq</i> . <i>OnZeroMQEvent</i> attribute), 49	MAX_SWEEPS_SHOTS
MAX_INSTANCES	MAX_SWEEPS_SHOTS
(<i>msl.equipment.resources.picotech.picoscope.enums.PS4000Range</i> attribute), 271	(<i>msl.equipment.resources.picotech.picoscope.ps5000.Pico attribute), 354</i>
MAX_SIG_GEN_BUFFER_SIZE	MAX_SWEEPS_SHOTS
(<i>msl.equipment.resources.picotech.picoscope.ps2000</i> (<i>PicoScope2000</i> resources.picotech.picoscope.ps5000a.Pico attribute), 344	(<i>msl.equipment.resources.picotech.picoscope.ps5000a.Pico attribute), 355</i>
MAX_SIG_GEN_BUFFER_SIZE	MAX_SWEEPS_SHOTS
(<i>msl.equipment.resources.picotech.picoscope.ps3000</i> (<i>PicoScope3000</i> resources.picotech.picoscope.ps6000.Pico attribute), 348	(<i>msl.equipment.resources.picotech.picoscope.ps6000.Pico attribute), 356</i>
MAX_SIG_GEN_BUFFER_SIZE	MAX_TEMP_SENSORS
(<i>msl.equipment.resources.picotech.picoscope.ps4000</i> (<i>PicoScope4000</i> resources.avantes.avaspec.Avantes attribute), 350	(<i>msl.equipment.resources.picotech.picoscope.ps4000</i> (<i>PicoScope4000</i> resources.picotech.picoscope.enums.PS4000 attribute), 106
MAX_SIG_GEN_BUFFER_SIZE	MAX_TEMPERATURES
(<i>msl.equipment.resources.picotech.picoscope.ps4000</i> (<i>PicoScope4000</i> resources.picotech.picoscope.enums.PS4000 attribute), 352	(<i>msl.equipment.resources.picotech.picoscope.ps4000</i> (<i>PicoScope4000</i> resources.picotech.picoscope.enums.PS4000 attribute), 272

MAX_TIMEBASE (<i>msl.equipment.resources.picotech.picoscope.PS2000Bla</i> attribute), 342	<i>msl.equipment.resources.picotech.picoscope.ps2000Bla</i> method), 148
MAX_TRANSIT_TIME (<i>msl.equipment.resources.thorlabs.kinesis.filter_flipper</i> attribute), 478	MAX_VALUE_16BIT <i>msl.equipment.resources.picotech.picoscope.ps5000a.Pic</i> attribute), 354
MAX_TRIGGER_SOURCES (<i>msl.equipment.resources.picotech.picoscope.enums.PS2000ACh</i> attribute), 241	MAX_VALUE_8BIT <i>msl.equipment.resources.picotech.picoscope.ps5000a.Pic</i> attribute), 354
MAX_TRIGGER_SOURCES (<i>msl.equipment.resources.picotech.picoscope.enums.PS3000ACh</i> attribute), 259	MAX_VIDEO_CHANNELS <i>msl.equipment.resources.picotech.picoscope.avantes.avaspec.Avant</i> attribute), 106
MAX_TRIGGER_SOURCES (<i>msl.equipment.resources.picotech.picoscope.enums.PS3000ACh</i> attribute), 252	MAX_WAVEFORMS_PER_SECOND <i>msl.equipment.resources.picotech.picoscope.ps6000.Pico</i> attribute), 356
MAX_TRIGGER_SOURCES (<i>msl.equipment.resources.picotech.picoscope.enums.PS4000ACh</i> attribute), 281	MAX WIRES (<i>msl.equipment.resources.picotech.pt104.PT104</i> maxAcceleration <i>msl.equipment.resources.thorlabs.kinesis.structs.MOT_S</i> attribute), 571
MAX_TRIGGER_SOURCES (<i>msl.equipment.resources.picotech.picoscope.enums.PS4000ACh</i> attribute), 270	maxChannels (<i>msl.equipment.resources.thorlabs.kinesis.structs.TL</i> attribute), 571
MAX_TRIGGER_SOURCES (<i>msl.equipment.resources.picotech.picoscope.PS5000ACh</i> attribute), 306	maxDemand (<i>msl.equipment.resources.picotech.picoscope.PS5000ACh</i> attribute), 573
MAX_TRIGGER_SOURCES (<i>msl.equipment.resources.picotech.picoscope.PS5000ACh</i> attribute), 296	maximum_message_size <i>msl.equipment.resources.picotech.picoscope.PS6000Ch</i> attribute), 67
MAX_TRIGGER_SOURCES (<i>msl.equipment.resources.picotech.picoscope.PS6000Ch</i> attribute), 315	maximumMessageSize <i>msl.equipment.hislip.HiSLIPClient</i> maximum_message_size
MAX_UNITS (<i>msl.equipment.resources.picotech.picoscope.PS2000Pic</i> attribute), 342	maximumMessageSize <i>msl.equipment.hislip.HiSLIPClient</i> maximum_message_size
MAX_UNITS (<i>msl.equipment.resources.picotech.picoscope.ps3000Pic</i> attribute), 345	maximum_value() <i>msl.equipment.resources.picotech.picoscope.picoscope.P</i> maximum_value()
MAX_UNITS_OPENED (<i>msl.equipment.resources.picotech.errors.PS2000Er</i> attribute), 228	maximum_value() <i>msl.equipment.resources.picotech.picoscope.picoscope.P</i> maximum_value()
MAX_UNITS_OPENED (<i>msl.equipment.resources.picotech.errors.PS2000Er</i> attribute), 228	maxPosition (<i>msl.equipment.resources.thorlabs.kinesis.structs.M</i> attribute), 573
MAX_UNITS_OPENED (<i>msl.equipment.resources.picotech.errors.PS2000Er</i> attribute), 228	maxPosition (<i>msl.equipment.resources.thorlabs.kinesis.structs.M</i> attribute), 573
MAX_VALUE (<i>msl.equipment.resources.picotech.picoscope.ps2000Pic</i> attribute), 343	maxVelocity (<i>msl.equipment.resources.thorlabs.kinesis.structs.M</i> attribute), 575
MAX_VALUE (<i>msl.equipment.resources.picotech.picoscope.ps3000Pic</i> attribute), 346	maxVelocity (<i>msl.equipment.resources.thorlabs.kinesis.structs.M</i> attribute), 575
MAX_VALUE (<i>msl.equipment.resources.picotech.picoscope.ps4000Pic</i> attribute), 349	maxXdemand (<i>msl.equipment.resources.thorlabs.kinesis.structs.QD</i> attribute), 575
MAX_VALUE (<i>msl.equipment.resources.picotech.picoscope.ps4000Pic</i> attribute), 351	maxYdemand (<i>msl.equipment.resources.thorlabs.kinesis.structs.QD</i> attribute), 575
MAX_VALUE (<i>msl.equipment.resources.picotech.picoscope.ps5000Pic</i> attribute), 353	MeasConfigType (class in
MAX_VALUE (<i>msl.equipment.resources.picotech.picoscope.ps6000Pic</i> attribute), 356	102

MeasurandRecord (class in *msl.equipment.record_types*), 83 message_queue_size()
measurands (*msl.equipment.record_types.CalibrationRecord*(*msl.equipment.resources.thorlabs.kinesis.kcube_solenoid* attribute), 83 method), 534
measure() (*msl.equipment.resources.avantes.avaspec*.~~MESSAGE_TOO_LARGE~~ message_queue_size()
method), 121 (msl.equipment.resources.thorlabs.kinesis.kcube_stepper_ measure_callback()
method), 550 (msl.equipment.resources.avantes.avaspec.~~MESSAGE_TOO_LARGE~~ MESSAGE_TOO_LARGE
method), 122 (msl.equipment.hislip.ErrorType attribute), 58
MeasureCallback (in module *msl.equipment.resources.avantes.avaspec*), MessageID (in module
105 *msl.equipment.resources.thorlabs.kinesis.messages*),
measurement() (*msl.equipment.resources.bentham.benhw32*.~~Bentham32~~ MessageType (class in *msl.equipment.hislip*), 57
method), 128 MessageType (class in *msl.equipment.vxi11*), 602
measurement_range() (*msl.equipment.resources.greisinger.gmh300*.~~MESSAGE_TYPES~~ MESSAGE_TYPES (in module
method), 148 *msl.equipment.resources.thorlabs.kinesis.messages*),
MEM_FAIL (*msl.equipment.resources.picotech.errors.PS2000Error* MILLI_TO_MICRO
attribute), 228
MEM_FAIL (*msl.equipment.resources.picotech.errors.PS3000Error* attribute), 107
MEMORY (*msl.equipment.resources.picoscope.picoscopy*.~~MinP&4000APriceStringValue~~.nktpdll.DateTimeType
attribute), 293 attribute), 162
MEMORY_MAX_SAMPLES MIN_ANALOGUE_OFFSET_500MV_2V
(*msl.equipment.resources.picotech.picoscope.enums.PS4000APriceStringValues*.picotech.picoscope.ps2000a.Pic attribute), 293 attribute), 344
MEMORY_NO_OF_SEGMENTS MIN_ANALOGUE_OFFSET_500MV_2V
(*msl.equipment.resources.picotech.picoscope.enums.PS4000APriceStringValues*.picotech.picoscope.ps3000a.Pic attribute), 293 attribute), 348
memory_segments() MIN_ANALOGUE_OFFSET_500MV_2V
(*msl.equipment.resources.picotech.picoscope.picoscopy*.~~ps4000PicoscopeAPIprices~~.picotech.picoscope.ps4000a.Pic method), 335 attribute), 352
Message (class in *msl.equipment.hislip*), 59 MIN_ANALOGUE_OFFSET_500MV_2V
message (*msl.equipment.hislip.HiSLIPException* attribute), 355
property), 59
message_id (*msl.equipment.hislip.SyncClient* MIN_ANALOGUE_OFFSET_500MV_2V
property), 75 (msl.equipment.resources.picotech.picoscope.ps6000.Pico attribute), 357
message_id_received prop- MIN_ANALOGUE_OFFSET_50MV_200MV
erty), 75 (msl.equipment.resources.picotech.picoscope.ps2000a.Pico attribute), 357
message_queue_size() MIN_ANALOGUE_OFFSET_50MV_200MV
(*msl.equipment.resources.thorlabs.kinesis.bentham32*.~~MIN_ANALOGUE_OFFSET_50MV_200MVUpperMotor~~ attribute), 415 (msl.equipment.resources.picotech.picoscope.ps3000a.Pico attribute), 344
message_queue_size() MIN_ANALOGUE_OFFSET_50MV_200MV
(*msl.equipment.resources.thorlabs.kinesis.fim*.~~MIN_ANALOGUE_OFFSET_50MV_200MVUpperMotors~~ attribute), 482 (msl.equipment.resources.picotech.picoscope.ps4000a.Pico attribute), 352
message_queue_size() MIN_ANALOGUE_OFFSET_50MV_200MV
(*msl.equipment.resources.thorlabs.kinesis.inertial*.~~MIN_ANALOGUE_OFFSET_50MV_200MVUpperMotors~~ attribute), 493 (msl.equipment.resources.picotech.picoscope.ps5000a.Pico attribute), 355
message_queue_size() MIN_ANALOGUE_OFFSET_50MV_200MV
(*msl.equipment.resources.thorlabs.kinesis.kcube*.~~MIN_ANALOGUE_OFFSET_50MV_200MV~~ attribute), 885 (msl.equipment.resources.picotech.picoscope.ps6000.Pico attribute), 355

<code>(msl.equipment.resources.picotech.picoscope.ps6000a.PicoScope2000 attribute), 356</code>	<code>MIN_FREQUENCY (msl.equipment.resources.picotech.picoscope.ps5000a.PicoScope2000A attribute), 355</code>
<code>MIN_ANALOGUE_OFFSET_5V_20V (msl.equipment.resources.picotech.picoscope.ps6000a.PicoScope2000A attribute), 344</code>	<code>MIN2_FREQUENCY (msl.equipment.resources.picotech.picoscope.ps6000a.PicoScope2000A attribute), 357</code>
<code>MIN_ANALOGUE_OFFSET_5V_20V (msl.equipment.resources.picotech.picoscope.ps3000a.PicoScope3000A attribute), 348</code>	<code>MIN_ILX_INNTIME (msl.equipment.resources.picotech.picoscope.ps3000a.PicoScope3000A attribute), 107</code>
<code>MIN_ANALOGUE_OFFSET_5V_20V (msl.equipment.resources.picotech.picoscope.ps4000a.PicoScope4000A attribute), 352</code>	<code>MIN_LOGIC_LEVEL (msl.equipment.resources.picotech.picoscope.ps5000a.PicoScope5000A attribute), 344</code>
<code>MIN_ANALOGUE_OFFSET_5V_20V (msl.equipment.resources.picotech.picoscope.ps6000a.PicoScope6000A attribute), 355</code>	<code>MIN_LOGIC_LEVEL (msl.equipment.resources.picotech.picoscope.ps5000a.PicoScope5000A attribute), 347</code>
<code>MIN_DWELL_COUNT (msl.equipment.resources.picotech.picoscope.ps2000a.PicoScope2000A attribute), 344</code>	<code>MIN_PULSE_WIDTH (msl.equipment.resources.picotech.picoscope.ps6000a.PicoScope6000A attribute), 478</code>
<code>MIN_DWELL_COUNT (msl.equipment.resources.picotech.picoscope.ps3000a.PicoScope3000A attribute), 348</code>	<code>MIN_SIG_GEN_BUFFER_SIZE (msl.equipment.resources.picotech.picoscope.ps2000a.PicoScope2000A attribute), 344</code>
<code>MIN_DWELL_COUNT (msl.equipment.resources.picotech.picoscope.ps4000a.PicoScope4000A attribute), 350</code>	<code>MIN_SIG_GEN_BUFFER_SIZE (msl.equipment.resources.picotech.picoscope.ps3000a.PicoScope3000A attribute), 348</code>
<code>MIN_DWELL_COUNT (msl.equipment.resources.picotech.picoscope.ps4000a.PicoScope4000A attribute), 352</code>	<code>MIN_SIG_GEN_BUFFER_SIZE (msl.equipment.resources.picotech.picoscope.ps4000a.PicoScope4000A attribute), 352</code>
<code>MIN_DWELL_COUNT (msl.equipment.resources.picotech.picoscope.ps5000a.PicoScope5000A attribute), 354</code>	<code>MIN_SIG_GEN_BUFFER_SIZE (msl.equipment.resources.picotech.picoscope.ps5000a.PicoScope5000A attribute), 353</code>
<code>MIN_DWELL_COUNT (msl.equipment.resources.picotech.picoscope.ps5000a.PicoScope5000A attribute), 355</code>	<code>MIN_SIG_GEN_BUFFER_SIZE (msl.equipment.resources.picotech.picoscope.ps6000a.PicoScope6000A attribute), 355</code>
<code>MIN_DWELL_COUNT (msl.equipment.resources.picotech.picoscope.ps6000a.PicoScope6000A attribute), 356</code>	<code>MIN_SIG_GEN_BUFFER_SIZE (msl.equipment.resources.picotech.picoscope.ps2000a.PicoScope2000A attribute), 344</code>
<code>MIN_ETS_CYCLES_INTERLEAVE_RATIO (msl.equipment.resources.picotech.picoscope.ps2000a.PicoScope2000A attribute), 343</code>	<code>MIN_SIG_GEN_FREQ (msl.equipment.resources.picotech.picoscope.ps3000a.PicoScope3000A attribute), 344</code>
<code>MIN_ETS_CYCLES_INTERLEAVE_RATIO (msl.equipment.resources.picotech.picoscope.ps3000a.PicoScope3000A attribute), 346</code>	<code>MIN_SIG_GEN_FREQ (msl.equipment.resources.picotech.picoscope.ps4000a.PicoScope4000A attribute), 347</code>
<code>MIN_FREQUENCY (msl.equipment.resources.picotech.picoscope.ps2000a.PicoScope2000A attribute), 345</code>	<code>MIN_SIGGEN_FREQ (msl.equipment.resources.picotech.picoscope.ps4000a.PicoScope4000A attribute), 345</code>
<code>MIN_FREQUENCY (msl.equipment.resources.picotech.picoscope.ps3000a.PicoScope3000A attribute), 348</code>	<code>MIN_SIGGEN_FREQ (msl.equipment.resources.picotech.picoscope.ps2000a.PicoScope2000A attribute), 343</code>
<code>MIN_FREQUENCY (msl.equipment.resources.picotech.picoscope.ps4000a.PicoScope4000A attribute), 352</code>	<code>MIN_SIGGEN_FREQ (msl.equipment.resources.picotech.picoscope.ps5000a.PicoScope5000A attribute), 343</code>
<code>MIN_FREQUENCY (msl.equipment.resources.picotech.picoscope.ps5000a.PicoScope5000A attribute), 355</code>	

<code>(msl.equipment.resources.picotech.picoscope.ps2000.PicoScope2000Sources.thorlabs.kinesis structs.PZ_LUTWa attribute), 346</code>	<code>mode1(msl.equipment.resources.thorlabs.kinesis.structs.KLD_Trig attribute), 579</code>
<code>MIN_TRANSIT_TIME (msl.equipment.resources.thorlabs.kinesis.filter_flippattFilterFlipper attribute), 478</code>	<code>mode1(msl.equipment.resources.thorlabs.kinesis.structs.KLS_Trig attribute), 585</code>
<code>MIN_VALUE(msl.equipment.resources.picotech.picoscope.ps2000.PicoScope2000 attribute), 343</code>	<code>mode2(msl.equipment.resources.thorlabs.kinesis.structs.KLD_Trig attribute), 585</code>
<code>MIN_VALUE(msl.equipment.resources.picotech.picoscope.ps3000.PicoScope3000 attribute), 346</code>	<code>mode2(msl.equipment.resources.thorlabs.kinesis.structs.KLS_Trig attribute), 585</code>
<code>MIN_VALUE(msl.equipment.resources.picotech.picoscope.ps4000.PicoScope4000 attribute), 349</code>	<code>model(msl.equipment.record_types.ConnectionRecord attribute), 351</code>
<code>MIN_VALUE(msl.equipment.resources.picotech.picoscope.ps4000.PicoScope4000A attribute), 351</code>	<code>model(msl.equipment.record_types.EquipmentRecord attribute), 351</code>
<code>MIN_VALUE(msl.equipment.resources.picotech.picoscope.ps5000.PicoScope5000 attribute), 353</code>	<code>modelNumber(msl.equipment.resources.thorlabs.kinesis.structs.TLI_Ha attribute), 571</code>
<code>MIN_VALUE(msl.equipment.resources.picotech.picoscope.ps6000.PicoScope6000 attribute), 356</code>	<code>modificationState</code>
<code>min_value()<code>(msl.equipment.resources.greisinger.gmh3000.GMH3000ment.resources.thorlabs.kinesis.structs.TLI_Ha method), 149</code></code>	<code>Modular_NanoTrak</code>
<code>MIN_VALUE_16BIT (msl.equipment.resources.picotech.picoscope.ps5000.attribute), 354</code>	<code>Modular_Piezo(msl.equipment.resources.thorlabs.kinesis.motion attribute), 568</code>
<code>MIN_VALUE_8BIT (msl.equipment.resources.picotech.picoscope.ps5000.attribute), 354</code>	<code>Modular_Stepper_Motor</code>
<code>MIN_WIRES(msl.equipment.resources.picotech.pt104.PT104 attribute), 372</code>	<code>(msl.equipment.resources.thorlabs.kinesis.motion_controls.NT_CircleParameters attribute), 568</code>
<code>minDiameter(msl.equipment.resources.thorlabs.kin attribute), 577</code>	<code>msl.equipment, 17</code>
<code>minimum_value()<code>(msl.equipment.resources.picotech.picoscope.pic method), 326</code></code>	<code>msl.equipment.config, 17</code>
<code>minPosition(msl.equipment.resources.thorlabs.kinesis.m attribute), 573</code>	<code>msl.equipment.connection_demo, 21</code>
<code>minVelocity(msl.equipment.resources.thorlabs.kinesis.struc attribute), 572</code>	<code>msl.equipment.connection_message_based, 31</code>
<code>minXdemand(msl.equipment.resources.thorlabs.kinesis.struc attribute), 589</code>	<code>msl.equipment.connection_nidaq, 31</code>
<code>minYdemand(msl.equipment.resources.thorlabs.kinesis.struc attribute), 589</code>	<code>msl.equipment.connection_sdlk, 38</code>
<code>MKSInstrumentsError, 55</code>	<code>msl.equipment.connection_serial, 39</code>
<code>mode(msl.equipment.resources.thorlabs.kinesis.structs.KM attribute), 586</code>	<code>msl.equipment.connection_tcpip_hislip, 41</code>
<code>mode(msl.equipment.resources.thorlabs.kinesis.structs.MOT_A attribute), 572</code>	<code>msl.equipment.connection_tcpip_vxi11, 48</code>
<code>mode(msl.equipment.resources.thorlabs.kinesis.structs.MOT_V attribute), 572</code>	<code>msl.equipment.connection_zeromq, 48</code>
<code>mode(msl.equipment.resources.thorlabs.kinesis.structs.N attribute), 577</code>	<code>msl.equipment.database, 51</code>
<code>mode(msl.equipment.resources.thorlabs.kinesis.structs.N attribute), 578</code>	<code>msl.equipment.service_discovery, 54</code>

msl.equipment.exceptions, 54	msl.equipment.resources.optosigma.shot702, 200
msl.equipment.factory, 56	msl.equipment.resources.optronic_laboratories, 206
msl.equipment.hislip, 57	msl.equipment.resources.optronic_laboratories.old, 206
msl.equipment.resources, 87	msl.equipment.resources.optronic_laboratories.old, 206
msl.equipment.resources.aim_tti, 90	msl.equipment.resources.optronic_laboratories.old, 206
msl.equipment.resources.aim_tti.mx_series, 90	msl.equipment.resources.optronic_laboratories.old, 206
msl.equipment.resources.avantes, 97	msl.equipment.resources.picotech, 224
msl.equipment.resources.avantes.avaspec, msl.equipment.resources.optronic_laboratories.old, 97	msl.equipment.resources.picotech, 224
msl.equipment.resources.bentham, 127	msl.equipment.resources.picotech.errors, 228
msl.equipment.resources.bentham.benhw32, 128	msl.equipment.resources.picotech.picoscope, 228
msl.equipment.resources.bentham.benhw64, 130	msl.equipment.resources.picotech.picoscope.call, 228
msl.equipment.resources.bentham.errors, 131	msl.equipment.resources.picotech.picoscope.channels, 229
msl.equipment.resources.bentham.tokens, 131	msl.equipment.resources.picotech.picoscope.channels, 229
msl.equipment.resources.cmi, 131	msl.equipment.resources.picotech.picoscope.functions, 230
msl.equipment.resources.cmi.sia3, 131	msl.equipment.resources.picotech.picoscope.enums, 232
msl.equipment.resources.dataray, 133	msl.equipment.resources.picotech.picoscope.helpers, 233
msl.equipment.resources.dataray.datarayocx_3223, 133	msl.equipment.resources.picotech.picoscope.helpers, 233
msl.equipment.resources.dataray.datarayocx_6423, 133	msl.equipment.resources.picotech.picoscope.picos, 235
msl.equipment.resources.electron_dynamics, 136	msl.equipment.resources.picotech.picoscope.picos, 235
msl.equipment.resources.electron_dynamics.tc320series, 136	msl.equipment.resources.picotech.picoscope.picos, 235
msl.equipment.resources.energetiq, 141	msl.equipment.resources.picotech.picoscope.ps2000, 331
msl.equipment.resources.energetiq.eq99, 141	msl.equipment.resources.picotech.picoscope.ps2000, 342
msl.equipment.resources.greisinger, 147	msl.equipment.resources.picotech.picoscope.ps3000, 344
msl.equipment.resources.greisinger.gmh3000, 147	msl.equipment.resources.picotech.picoscope.ps3000, 345
msl.equipment.resources.mks_instruments, 151	msl.equipment.resources.picotech.picoscope.ps4000, 347
msl.equipment.resources.mks_instruments.pr4000b, 151	msl.equipment.resources.picotech.picoscope.ps4000, 351
msl.equipment.resources.nkt, 162	msl.equipment.resources.picotech.picoscope.ps5000, 353
msl.equipment.resources.nkt.nktpdll, 162	msl.equipment.resources.picotech.picoscope.ps5000, 354
msl.equipment.resources.omega, 195	msl.equipment.resources.picotech.picoscope.ps6000, 356
msl.equipment.resources.omega.ithx, 195	msl.equipment.resources.picotech.picoscope.structure, 356
msl.equipment.resources.optosigma, 199	msl.equipment.resources.picotech.picoscope.structure, 356

```

    358                               (msl.equipment.resources.princeton_instruments.arc_instr
msl.equipment.resources.picotech.pt104,      method), 375
    370                               mono_grating_calc_gadjust()
msl.equipment.resources.princeton_instrument(msl.equipment.resources.princeton_instruments.arc_instr
    374                               method), 375
msl.equipment.resources.princeton_instrumentmono_grating_calibrate()mono_grating_calibrate()
    374                               (msl.equipment.resources.princeton_instruments.arc_instr
msl.equipment.resources.raicol, 401           method), 375
msl.equipment.resources.raicol.raicolmono_grating_install()mono_grating_install()
    401                               (msl.equipment.resources.princeton_instruments.arc_instr
msl.equipment.resources.thorlabs,             method), 375
    402                               mono_grating_uninstall()
msl.equipment.resources.thorlabs.fwxx2c,       (msl.equipment.resources.princeton_instruments.arc_instr
    593                               method), 376
msl.equipment.resources.thorlabs.kinesisiso_move_steps()iso_move_steps()
    402                               (msl.equipment.resources.princeton_instruments.arc_instr
msl.equipment.resources.thorlabs.kinesis.apinfinditionsfinditions,
    402                               mono_reset() (msl.equipment.resources.princeton_instruments.arc_instr
msl.equipment.resources.thorlabs.kinesis.benchtopstepper_motorstepper_motor,
    402                               mono_restore_factory_settings()
msl.equipment.resources.thorlabs.kinesis.calhomehomemono_start_jog()mono_start_jog()
    431                               (msl.equipment.resources.princeton_instruments.arc_instr
msl.equipment.resources.thorlabs.kinesisiso_scansiso_scans, done()
    431                               (msl.equipment.resources.princeton_instruments.arc_instr
msl.equipment.resources.thorlabs.kinesis.errormodmod, 376
    478                               mono_slit_home()
msl.equipment.resources.thorlabs.kinesis.filterfilterclipperclipperresources.princeton_instruments.arc_instrresources.princeton_instruments.arc_instr
    478                               method), 376
msl.equipment.resources.thorlabs.kinesisiso_integratediso_integrated()stepper_motorsstepper_motors,
    483                               (msl.equipment.resources.princeton_instruments.arc_instr
msl.equipment.resources.thorlabs.kinesis.kcubeatidenshieldatidenshield, 377
    505                               mono_start_jog()
msl.equipment.resources.thorlabs.kinesis.kcubebesidebesidemono_start_jog()mono_start_jog()
    530                               (msl.equipment.resources.princeton_instruments.arc_instr
msl.equipment.resources.thorlabs.kinesisiso_kstetiso_kstetstepper_motorstepper_motor),
    539                               (msl.equipment.resources.princeton_instruments.arc_instr
msl.equipment.resources.thorlabs.kinesis.messagemodmod, 377
    565                               mono_stop_jog()
msl.equipment.resources.thorlabs.kinesis.motioncontrollcontrollresources.princeton_instruments.arc_instrresources.princeton_instruments.arc_instr
    566                               method), 377
msl.equipment.resources.thorlabs.kineMotMotstrucstrucequipment.resources.nkt.nktpdll.DateTimeTypeequipment.resources.nkt.nktpdll.DateTimeType
    570                               attribute), 162
msl.equipment.resources.utils, 88          MOT_Backwards (msl.equipment.resources.thorlabs.kinesis.enums.
msl.equipment.utils, 598                   attribute), 433
msl.equipment.vxi11, 601                  MOT_BrushlessCurrentLoopParameters
monitorOPBandwidth                         (class                                     in
                                         (msl.equipment.resources.thorlabs.kinesis.structs.PPsisIOSEquipmentEquipment.resources.thorlabs.kinesis.structs),
                                         attribute), 581                           575
monitorOPSig (msl.equipment.resources.thorlabs.kineMOTsBrushlessPEMOTsBrushlessPEStringStringoutputParametersoutputParameters
                                         attribute), 581                           (class                                     in
                                         msl.equipment.resources.thorlabs.kinesis.structs),

```

575 MOT_BrushlessMotor (<i>msl.equipment.resources.thorlabs.kinesis.enums.MOT_JogModeTypes</i> attribute), 432 MOT_BrushlessPositionLoopParameters (class in <i>MOT_JogModeUndefined</i> <i>msl.equipment.resources.thorlabs.kinesis structs</i>), (<i>msl.equipment.resources.thorlabs.kinesis.enums.MOT_JogModeTypes</i> attribute), 434 574 MOT_BrushlessTrackSettleParameters (class in <i>MOT_JogParameters</i> <i>msl.equipment.resources.thorlabs.kinesis structs</i>), 572 575 MOT_ButtonModes (class in <i>MOT_JoystickParameters</i> <i>msl.equipment.resources.thorlabs.kinesis structs</i>), 574 434 MOT_ButtonModeUndefined (class in <i>MOT_LimitsSoftwareApproachPolicy</i> <i>msl.equipment.resources.thorlabs.kinesis.enums.MOT_ButtonModeTypes</i> attribute), 436 436 MOT_ButtonParameters (class in <i>MOT_LimitSwitchBreakOnContact</i> <i>msl.equipment.resources.thorlabs.kinesis structs</i>), (<i>msl.equipment.resources.thorlabs.kinesis.enums.MOT_LimitSwitchBreakOnContact</i> attribute), 435 582 MOT_Continuous (class in <i>MOT_LimitSwitchBreakOnContactSwapped</i> <i>msl.equipment.resources.thorlabs.kinesis.enums.MOT_LimitSwitchBreakOnContact</i> attribute), 435 MOT_CurrentLoopPhases (class in <i>MOT_LimitSwitchBreakOnHome</i> <i>msl.equipment.resources.thorlabs.kinesis enums</i>), (<i>msl.equipment.resources.thorlabs.kinesis.enums.MOT_LimitSwitchBreakOnHome</i> attribute), 435 436 MOT_CustomMotor (class in <i>MOT_LimitSwitchBreakOnHomeSwapped</i> <i>msl.equipment.resources.thorlabs.kinesis.enums.MOT_LimitSwitchBreakOnHome</i> attribute), 436 MOT_DC_PIDParameters (class in <i>MOT_LimitSwitchDirectionUndefined</i> <i>msl.equipment.resources.thorlabs.kinesis structs</i>), (<i>msl.equipment.resources.thorlabs.kinesis.enums.MOT_LimitSwitchDirectionUndefined</i> attribute), 433 576 MOT_DCMotor (<i>msl.equipment.resources.thorlabs.kinesis.enums.MOT_Direction</i> attribute), 432 MOT_DirectionSense (class in <i>MOT_LimitSwitchIgnored_Rotational</i> <i>msl.equipment.resources.thorlabs.kinesis enums</i>), (<i>msl.equipment.resources.thorlabs.kinesis.enums.MOT_LimitSwitchIgnored_Rotational</i> attribute), 436 433 MOT_ForwardLimitSwitch (class in <i>MOT_LimitSwitchIgnored_SwitchDirection</i> <i>msl.equipment.resources.thorlabs.kinesis enums</i>), (<i>msl.equipment.resources.thorlabs.kinesis.enums.MOT_LimitSwitchIgnored_SwitchDirection</i> attribute), 434 MOT_Forwards (<i>msl.equipment.resources.thorlabs.kinesis.enums.MOT_TravelDirection</i> attribute), 433 MOT_HomeLimitSwitchDirection (class in <i>MOT_LimitSwitchIgnoreSwitchDirection</i> <i>msl.equipment.resources.thorlabs.kinesis enums</i>), (<i>msl.equipment.resources.thorlabs.kinesis.enums.MOT_LimitSwitchIgnoreSwitchDirection</i> attribute), 435 433 MOT_HomingParameters (class in <i>MOT_LimitSwitchMakeOnContact</i> <i>msl.equipment.resources.thorlabs.kinesis structs</i>), (<i>msl.equipment.resources.thorlabs.kinesis.enums.MOT_LimitSwitchMakeOnContact</i> attribute), 435 572 MOT_Immediate (<i>msl.equipment.resources.thorlabs.kinesis.enums.MOT_StopMode</i> attribute), 434 MOT_StopMode (class in <i>MOT_LimitSwitchMakeOnContactSwapped</i> <i>msl.equipment.resources.thorlabs.kinesis enums</i>), (<i>msl.equipment.resources.thorlabs.kinesis.enums.MOT_LimitSwitchMakeOnContactSwapped</i> attribute), 435	
--	--

MOT_LimitSwitchMakeOnHome	(<i>msl.equipment.resources.thorlabs.kinesis.enums.MOT_LimitSwitchMode</i>)	<i>attribute</i>), 437
	<i>(msl.equipment.resources.thorlabs.kinesis.enums.MOT_LimitSwitchMode)</i>	<i>attribute</i>), 437
MOT_LimitSwitchMakeOnHomeSwapped	MOT_PID_LoopMode	(class in <i>(msl.equipment.resources.thorlabs.kinesis.enums.MOT_LimitSwitchMode)</i> , 438)
MOT_LimitSwitchModes	(class in MOT_PIDClosedLoopMode <i>(msl.equipment.resources.thorlabs.kinesis.enums.MOT_LimitSwitchMode)</i> , 438)	<i>(msl.equipment.resources.thorlabs.kinesis.enums.MOT_P</i>
MOT_LimitSwitchModeUndefined	MOT_PIDLoopEncoderParams	(class in <i>(msl.equipment.resources.thorlabs.kinesis.enums.MOT_LimitSwitchMode)</i> , 581)
MOT_LimitSwitchParameters	(class in MOT_PIDLoopModeDisabled <i>(msl.equipment.resources.thorlabs.kinesis.structs)</i> , 576)	<i>(msl.equipment.resources.thorlabs.kinesis.enums.MOT_P</i>
MOT_LimitSwitchStopImmediate	MOT_PIDOpenLoopMode	<i>(msl.equipment.resources.thorlabs.kinesis.enums.MOT_LimitSwitchSWMode)</i> , 438
	<i>(msl.equipment.resources.thorlabs.kinesis.enums.MOT_LimitSwitchSWMode)</i>	<i>attribute</i>), 438
MOT_LimitSwitchStopImmediate_Rotational	MOT_PMD_Reserved	<i>(msl.equipment.resources.thorlabs.kinesis.enums.MOT_Li</i>
	<i>(msl.equipment.resources.thorlabs.kinesis.enums.MOT_Li</i>	<i>attribute</i>), 435
MOT_LimitSwitchStopProfiled	MOT_PotentiometerStep	(class in <i>(msl.equipment.resources.thorlabs.kinesis.enums.MOT_L</i>
	<i>(msl.equipment.resources.thorlabs.kinesis.structs)</i> , 582	<i>attribute</i>), 436
MOT_LimitSwitchStopProfiled_Rotational	MOT_PotentiometerSteps	(class in <i>(msl.equipment.resources.thorlabs.kinesis.enums.MOT_L</i>
	<i>(msl.equipment.resources.thorlabs.kinesis.structs)</i> , 582	<i>attribute</i>), 436
MOT_LimitSwitchSWModes	(class in MOT_PowerParameters <i>(msl.equipment.resources.thorlabs.kinesis.enums.MOT</i>	<i>(class in <i>(msl.equipment.resources.thorlabs.kinesis.structs)</i>, 576</i>
MOT_LimitSwitchSWModeUndefined	MOT_Preset	<i>(msl.equipment.resources.thorlabs.kinesis.enums.MO</i>
	<i>(msl.equipment.resources.thorlabs.kinesis.enums.MOT</i>	<i>(msl.equipment.resources.thorlabs.kinesis.enums.MO</i>
	<i>(attribute), 436</i>	<i>(attribute), 436</i>
MOT_Linear	MOT_Profiled	<i>(msl.equipment.resources.thorlabs.kinesis.enums.MO</i>
	<i>(msl.equipment.resources.thorlabs.kinesis.enums.MOT</i>	<i>(msl.equipment.resources.thorlabs.kinesis.enums.MO</i>
	<i>(attribute), 432</i>	<i>(attribute), 432</i>
MOT_MotorTypes	(class in MOT_ReverseLimitSwitch <i>(msl.equipment.resources.thorlabs.kinesis.enums.MOT</i>	<i>(attribute), 433</i>
	<i>(attribute), 432</i>	<i>(msl.equipment.resources.thorlabs.kinesis.enums.MOT_H</i>
MOT_MovementDirections	(class in MOT_Rotational <i>(msl.equipment.resources.thorlabs.kinesis.enums.MOT</i>	<i>(attribute), 434</i>
	<i>(attribute), 437</i>	<i>(msl.equipment.resources.thorlabs.kinesis.enums.MOT_T</i>
MOT_MovementModes	(class in MOT_SCurve <i>(msl.equipment.resources.thorlabs.kinesis.enums.MOT</i>	<i>(attribute), 433</i>
	<i>(attribute), 437</i>	<i>(msl.equipment.resources.thorlabs.kinesis.enums.MOT_C</i>
MOT_Normal	MOT_StepDirectionSense	<i>(msl.equipment.resources.thorlabs.kinesis.enums.MOT_D</i>
	<i>(attribute), 433</i>	<i>(attribute), 433</i>
MOT_NotMotor	MOT_MotorTypes	<i>(msl.equipment.resources.thorlabs.kinesis.enums.MOT_M</i>
	<i>(attribute), 432</i>	<i>(attribute), 432</i>
MOT_PhaseA	MOT_StageAxisParameters	(class in <i>(msl.equipment.resources.thorlabs.kinesis.enums.MOT_Q</i>
	<i>(attribute), 437</i>	<i>(msl.equipment.resources.thorlabs.kinesis.structs)</i> , 573
MOT_PhaseAB	MOT_StepPhase	<i>(msl.equipment.resources.thorlabs.kinesis.enums.MOT_C</i>

(*msl.equipment.resources.thorlabs.kinesis.emove.MOT_StopMode*)
attribute), 432
MOT_StopModes (class in *msl.equipment.resources.thorlabs.kinesis.emove.MOT_StopMode*)
434
MOT_StopModeUndefined (*msl.equipment.resources.thorlabs.kinesis.emove.MOT_StopMode*)
attribute), 434
MOT_Trapezoidal (*msl.equipment.resources.thorlabs.kinesis.emove.MOT_StopMode*)
attribute), 435
MOT_TravelDirection (class in *msl.equipment.resources.thorlabs.kinesis.emove.MOT_StopMode*)
433
MOT_TravelDirectionDisabled (*msl.equipment.resources.thorlabs.kinesis.emove.MOT_StopMode*)
attribute), 433
MOT_TravelModes (class in *msl.equipment.resources.thorlabs.kinesis.emove.MOT_StopMode*)
432
MOT_TravelModeUndefined (*msl.equipment.resources.thorlabs.kinesis.emove.MOT_StopMode*)
attribute), 432
MOT_VelocityParameters (class in *msl.equipment.resources.thorlabs.kinesis.emove.MOT_StopMode*)
572
MOT_VelocityProfileModes (class in *move_jog()*)
msl.equipment.resources.thorlabs.kinesis.enums, method), 493
435
MOT_VelocityProfileParameters (class in *move_jog()*)
msl.equipment.resources.thorlabs.kinesis.enums, method), 550
572
MotionControl (class in *move_relative()*)
msl.equipment.resources.thorlabs.kinesis.motion_control(*msl.equipment.resources.optosigma.shot702.SHOT702*)
566
MotionControlCallback (in module *move_relative()*)
msl.equipment.resources.thorlabs.kinesis.callbacks, (*msl.equipment.resources.thorlabs.kinesis.benchtop_stepper*
431
motorSignalBias (*msl.equipment.resources.thorlabs.kinesis.structs.MotorSignalBias*)
attribute), 576
motorSignalLimit (*msl.equipment.resources.thorlabs.kinesis.structs.MotorSignalLimit*)
attribute), 576
motorType (*msl.equipment.resources.thorlabs.kinesis.motor_type*)
attribute), 571
move() (*msl.equipment.resources.optosigma.shot702.SHOT702*)
method), 550
method), 201
move_absolute() (*msl.equipment.resources.optosigma.shot702.SHOT702*)
method), 202
move_absolute() (*msl.equipment.resources.thorlabs.kinesis.benchtop_stepper*)
method), 493
move_at_velocity() (*msl.equipment.resources.thorlabs.kinesis.benchtop_stepper*)
method), 516
move_jog() (*msl.equipment.resources.thorlabs.kinesis.benchtop_stepper*)
method), 415
move_relative() (*msl.equipment.resources.thorlabs.kinesis.benchtop_stepper*)
method), 202
move_relative() (*msl.equipment.resources.thorlabs.kinesis.benchtop_stepper*)
method), 550
move_relative() (*msl.equipment.resources.thorlabs.kinesis.benchtop_stepper*)
method), 493
move_relative() (*msl.equipment.resources.thorlabs.kinesis.benchtop_stepper*)
method), 516
move_relative() (*msl.equipment.resources.thorlabs.kinesis.benchtop_stepper*)
method), 202
move_relative() (*msl.equipment.resources.thorlabs.kinesis.benchtop_stepper*)
method), 416
move_relative() (*msl.equipment.resources.thorlabs.kinesis.benchtop_stepper*)
method), 201
move_relative_distance() (*msl.equipment.resources.thorlabs.kinesis.benchtop_stepper*)
method), 493
move_relative_distance() (*msl.equipment.resources.thorlabs.kinesis.benchtop_stepper*)
method), 516
move_relative_distance() (*msl.equipment.resources.thorlabs.kinesis.benchtop_stepper*)
method), 202

(*msl.equipment.resources.thorlabs.kinesis.instrudl.stepipermetatorstIntsgRatekStepptirMate*)s
method), 493 50

move_relative_distance() msl.equipment
(*msl.equipment.resources.thorlabs.kinesis.kcubeModule.KCubeDCServo*
method), 516 msl.equipment.config
move_relative_distance() module, 17
(*msl.equipment.resources.thorlabs.kinesis.kmsle_equipment.coKubeSuperMotor*
method), 550 module, 19

move_to_position() msl.equipment.connection_demo
(*msl.equipment.resources.thorlabs.kinesis.benchmotor.BenchtopStepperMotor*
method), 416 msl.equipment.connection_gpib
move_to_position() module, 22
(*msl.equipment.resources.thorlabs.kinesis.fmsl_equipmFconnection_message_based*
method), 480 module, 29

move_to_position() msl.equipment.connection_nidaq
(*msl.equipment.resources.thorlabs.kinesis.integrstepper_motors.IntegratedStepperMotors*
method), 494 msl.equipment.connection_prologix
move_to_position() module, 33
(*msl.equipment.resources.thorlabs.kinesis.kmsle_equipmentKCubeConnection_pyvisa*
method), 516 module, 37

move_to_position() msl.equipment.connection_sdk
(*msl.equipment.resources.thorlabs.kinesis.kcubeStepper_Motor.KCubeStepperMotor*
method), 550 msl.equipment.connection_serial
move_to_wavelength() module, 39
(*msl.equipment.resources.optronic_laboratnids.equipment.Omconnection_socket*
method), 218 module, 41

movePercentage msl.equipment.connection_tcpip_hislip
(*msl.equipment.resources.thorlabs.kinesis.structs.MOTEPowerParameters*
attribute), 576 msl.equipment.connection_tcpip_vxi11
MS (*msl.equipment.resources.picotech.picoscope.enums.PS1000ETimeUnits*
attribute), 245 msl.equipment.connection_zeromq
MS (*msl.equipment.resources.picotech.picoscope.enums.PS1000ETimeUnits*
attribute), 234 msl.equipment.constants
MS (*msl.equipment.resources.picotech.picoscope.enums.PS1000ETimeUnits*
attribute), 263 msl.equipment.database
MS (*msl.equipment.resources.picotech.picoscope.enums.PS1000ETimeUnits*
attribute), 253 msl.equipment.dns_service_discovery
MS (*msl.equipment.resources.picotech.picoscope.enums.PS1000ETimeUnits*
attribute), 284 msl.equipment.exceptions
MS (*msl.equipment.resources.picotech.picoscope.enums.PS1000ETimeUnits*
attribute), 273 msl.equipment.factory
MS (*msl.equipment.resources.picotech.picoscope.enums.PS1000ETimeUnits*
attribute), 308 msl.equipment.hislip
MS (*msl.equipment.resources.picotech.picoscope.enums.PS1000ETimeUnits*
attribute), 298 msl.equipment.resources
MS (*msl.equipment.resources.picotech.picoscope.enums.PS1000ETimeUnits*
attribute), 317 msl.equipment.resources.aim_tti
MSG_ACCEPTED (*msl.equipment.vxi11.ReplyStatus*
attribute), 602 module, 90
MSG_DENIED (*msl.equipment.vxi11.ReplyStatus* at-
tribute), 602 msl.equipment.resources.aim_tti.mx_series
msl.equipment.resources.avantes

```
    module, 97
msl.equipment.resources.avantes.avaspec
    module, 97
msl.equipment.resources.bentham
    module, 127
msl.equipment.resources.bentham.benhw32
    module, 128
msl.equipment.resources.bentham.benhw64
    module, 130
msl.equipment.resources.bentham.errors
    module, 131
msl.equipment.resources.bentham.tokens
    module, 131
msl.equipment.resources.cmi
    module, 131
msl.equipment.resources.cmi.sia3
    module, 131
msl.equipment.resources.dataray
    module, 133
msl.equipment.resources.dataray.datarayocx_32
    module, 133
msl.equipment.resources.dataray.datarayocx_64
    module, 133
msl.equipment.resources.electron_dynamics
    module, 136
msl.equipment.resources.electron_dynamics.tandem
    module, 136
msl.equipment.resources.energetiq
    module, 141
msl.equipment.resources.energetiq.eq99
    module, 141
msl.equipment.resources.greisinger
    module, 147
msl.equipment.resources.greisinger.gmh3000
    module, 147
msl.equipment.resources.mks_instruments
    module, 151
msl.equipment.resources.mks_instruments.pr4000
    module, 151
msl.equipment.resources.nkt
    module, 162
msl.equipment.resources.nkt.nktpdll
    module, 162
msl.equipment.resources.omega
    module, 195
msl.equipment.resources.omega.ithx
    module, 195
msl.equipment.resources.optosigma
    module, 199
msl.equipment.resources.optosigma.shot702
    module, 200
msl.equipment.resources.optronic_laboratories
    module, 206
msl.equipment.resources.optronic_laboratories.ol756
    module, 206
msl.equipment.resources.optronic_laboratories.ol756
    module, 224
msl.equipment.resources.optronic_laboratories.ol_cu
    module, 225
msl.equipment.resources.picotech
    module, 228
msl.equipment.resources.picotech.errors
    module, 228
msl.equipment.resources.picotech.picoscope
    module, 229
msl.equipment.resources.picotech.picoscope.callback
    module, 229
msl.equipment.resources.picotech.picoscope.channel
    module, 230
msl.equipment.resources.picotech.picoscope.enums
    module, 232
msl.equipment.resources.picotech.picoscope.function
    module, 232
msl.equipment.resources.picotech.picoscope.helper
    module, 233
msl.equipment.resources.picotech.picoscope.picoscop
    module, 235
msl.equipment.resources.picotech.picoscope.picoscop
    module, 329
msl.equipment.resources.picotech.picoscope.picoscop
    module, 331
msl.equipment.resources.picotech.picoscope.ps2000
    module, 342
msl.equipment.resources.picotech.picoscope.ps2000a
    module, 344
msl.equipment.resources.picotech.picoscope.ps3000
    module, 345
msl.equipment.resources.picotech.picoscope.ps3000a
    module, 347
msl.equipment.resources.picotech.picoscope.ps4000
    module, 349
msl.equipment.resources.picotech.picoscope.ps4000a
    module, 351
msl.equipment.resources.picotech.picoscope.ps5000
    module, 353
msl.equipment.resources.picotech.picoscope.ps5000a
    module, 354
msl.equipment.resources.picotech.picoscope.ps6000
    module, 356
msl.equipment.resources.picotech.picoscope structs
    module, 358
msl.equipment.resources.picotech.pt104
    module, 358
```

```
    module, 370                               method), 129
msl.equipment.resources.princeton_instruments_auto_range()
    module, 374                               (msl.equipment.resources.bentham.benhw32.Bentham32
msl.equipment.resources.princeton_instruments.analog_instrument
    module, 374                               multi_get_no_of_dark_currents()
msl.equipment.resources.raicol
    module, 401                               (msl.equipment.resources.bentham.benhw32.Bentham32
msl.equipment.resources.raicol.raicol_te
    module, 401                               method), 128
multi_get_zero_calibration_info()
msl.equipment.resources.thorlabs
    module, 402                               multi_initialise()
msl.equipment.resources.thorlabs.fwxx2c
    module, 593                               (msl.equipment.resources.bentham.benhw32.Bentham32
msl.equipment.resources.thorlabs.kinesis
    module, 402                               multi_measurement()
msl.equipment.resources.thorlabs.kinesis.api_functions
    module, 402                               multi_park() (msl.equipment.resources.bentham.benhw32.Bentham32
msl.equipment.resources.thorlabs.kinesis.benchtop_stepper_motor
    module, 402                               multi_select_wavelength()
msl.equipment.resources.thorlabs.kinesis.callbacks
    module, 431                               (msl.equipment.resources.bentham.benhw32.Bentham32
msl.equipment.resources.thorlabs.kinesis.menus
    module, 431                               multi_zero_calibration()
msl.equipment.resources.thorlabs.kinesis.errors
    module, 478                               (msl.equipment.resources.thorlabs.kinesis.enums.TST_Stag
msl.equipment.resources.thorlabs.kinesis.filter_afilippe
    module, 478                               MXSeries           (class)           in
msl.equipment.resources.thorlabs.kinesis.integrated_stepper_motors.aim_tti.mx_series)
    module, 483                               90
msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo
    module, 505                               N
msl.equipment.resources.thorlabs.kinesis.land_dock_gpi
    module, 530                               land_dock_gpi
msl.equipment.resources.thorlabs.kinesis.land_dock_gpi.connection_gpib.ConnectionGPIB
    module, 530                               property), 26
msl.equipment.resources.thorlabs.kinesis.land_dock_gpi
    module, 539                               NanoTrak
msl.equipment.resources.thorlabs.kinesis.land_dock_gpi.tepper_motor(in
    module, 539                               module
msl.equipment.resources.thorlabs.kinesis.land_dock_gpi
    module, 566                               msl.equipment.resources.thorlabs.kinesis.messages),
msl.equipment.resources.thorlabs.kinesis.message
    module, 565                               ncl_filter_home()
msl.equipment.resources.thorlabs.kinesis.motion
    module, 566                               (msl.equipment.resources.princeton_instruments.arc_instru
msl.equipment.resources.thorlabs.kinesis.motion
    module, 566                               method), 378
msl.equipment.resources.thorlabs.kinesis.motion
    module, 570                               needs_homing()
msl.equipment.resources.utils
    module, 88                                (msl.equipment.resources.thorlabs.kinesis.benchtop_stepp
msl.equipment.utils
    module, 598                               method), 416
msl.equipment.vxi11
    module, 601                               needs_homing()
MSLConnectionError, 54
MSLTimeoutError, 54
multi_auto_measure()
    (msl.equipment.resources.bentham.benhw32.Bentham32
method), 551
```

NEGATIVE_RUNT (<i>msl.equipment.resources.picotech.picoscope.attributes.PS2000AThresholdDirection</i> , attribute), 249	170
NEGATIVE_RUNT (<i>msl.equipment.resources.picotech.NKTRegisters.PS3000AThresholdDirection</i> in attribute), 267	<i>NKTRegisters.PS3000AThresholdDirection</i> in <i>msl.equipment.resources.nkt.nktpdll</i> ,
NEGATIVE_RUNT (<i>msl.equipment.resources.picotech.picoscope.attributes.PS4000AThresholdDirection</i> , attribute), 290	<i>NKTError</i> , 55
NEGATIVE_RUNT (<i>msl.equipment.resources.picotech.NO_OF_CHANNELS</i> PS4000AThresholdDirection attribute), 278	(<i>msl.equipment.resources.picotech.picoscope.enums.PS4000AThresholdDirection</i> attribute), 278
NEGATIVE_RUNT (<i>msl.equipment.resources.picotech.picoscope.attributes.PS5000AThresholdDirection</i> , attribute), 311	<i>no_of_streaming_values()</i>
NEGATIVE_RUNT (<i>msl.equipment.resources.picotech.picoscope.attributes.PS6000AThresholdDirection</i> , attribute), 321	<i>method</i>), 335
newRange (<i>msl.equipment.resources.thorlabs.kinesis.NO_OF_TRIGGER_PROPERTIES</i> attribute), 586	(<i>msl.equipment.resources.picotech.picoscope.enums.PS4000AThresholdDirection</i> attribute), 586
newRange (<i>msl.equipment.resources.thorlabs.kinesis.structs.NTTRange</i> PS4000AThresholdDirection attribute), 578	<i>NO_PRESS</i> (<i>msl.equipment.resources.picotech.picoscope.enums.PS4000AThresholdDirection</i> attribute), 578
NEWZFS06 (<i>msl.equipment.resources.thorlabs.kinesis.enums.TSTrStage</i>), 236	<i>NONE</i> (<i>msl.equipment.constants.Interface</i> at- attribute), 477
NEWZFS13 (<i>msl.equipment.resources.thorlabs.kinesis.enums.TSTrStage</i>), 250	<i>NONE</i> (<i>msl.equipment.constants.Parity</i> attribute), attribute), 477
NEWZFS25 (<i>msl.equipment.resources.thorlabs.kinesis.enums.TSTrStage</i> , attribute), 477	<i>NONE</i> (<i>msl.equipment.resources.picotech.picoscope.enums.PS2000AThresholdDirection</i> attribute), 477
NEWZST06 (<i>msl.equipment.resources.thorlabs.kinesis.enums.TSttrStage</i>), 251	<i>NONE</i> (<i>msl.equipment.resources.picotech.picoscope.enums.PS2000AThresholdDirection</i> attribute), 477
NEWZST13 (<i>msl.equipment.resources.thorlabs.kinesis.enums.TSttrStage</i>), 250	<i>NONE</i> (<i>msl.equipment.resources.picotech.picoscope.enums.PS2000AThresholdDirection</i> attribute), 477
NEWZST25 (<i>msl.equipment.resources.thorlabs.kinesis.enums.TSttrStage</i>), 247	<i>NONE</i> (<i>msl.equipment.resources.picotech.picoscope.enums.PS2000AThresholdDirection</i> attribute), 477
next_calibration_date() (<i>msl.equipment.record_types.EquipmentRecord</i> , method), 82	<i>NONE</i> (<i>msl.equipment.resources.picotech.picoscope.enums.PS2000AThresholdDirection</i> attribute), 249
NIDAQ (<i>msl.equipment.constants.Backend</i> attribute), 50	<i>NONE</i> (<i>msl.equipment.resources.picotech.picoscope.enums.PS2000AThresholdDirection</i> attribute), 233
NKT (<i>class</i> in <i>msl.equipment.resources.nkt.nktpdll</i>), 168	<i>NONE</i> (<i>msl.equipment.resources.picotech.picoscope.enums.PS2000AThresholdDirection</i> attribute), 239
NKT.DeviceModeTypes (<i>class</i> in <i>msl.equipment.resources.nkt.nktpdll</i>), 168	<i>NONE</i> (<i>msl.equipment.resources.picotech.picoscope.enums.PS3000AThresholdDirection</i> attribute), 268
NKT.DeviceStatusTypes (<i>class</i> in <i>msl.equipment.resources.nkt.nktpdll</i>), 169	<i>NONE</i> (<i>msl.equipment.resources.picotech.picoscope.enums.PS3000AThresholdDirection</i> attribute), 268
NKT.ParamSetUnitTypes (<i>class</i> in <i>msl.equipment.resources.nkt.nktpdll</i>), 169	<i>NONE</i> (<i>msl.equipment.resources.picotech.picoscope.enums.PS3000AThresholdDirection</i> attribute), 267
NKT.PortStatusTypes (<i>class</i> in <i>msl.equipment.resources.nkt.nktpdll</i>), 172	<i>NONE</i> (<i>msl.equipment.resources.picotech.picoscope.enums.PS3000AThresholdDirection</i> attribute), 252
NKT.RegisterDataTypes (<i>class</i> in <i>msl.equipment.resources.nkt.nktpdll</i>), 171	<i>NONE</i> (<i>msl.equipment.resources.picotech.picoscope.enums.PS3000AThresholdDirection</i> attribute), 256
NKT.RegisterPriorityTypes (<i>class</i> in	<i>NONE</i> (<i>msl.equipment.resources.picotech.picoscope.enums.PS4000AThresholdDirection</i> attribute), 292

NONE (msl.equipment.resources.picotech.picoscope.enums.PC_ATTRIBUTE, 291) NOT RESPONDING (msl.equipment.resources.picotech.errors.PS3000Error attribute), 291

NONE (msl.equipment.resources.picotech.picoscope.enums.PS4000A_SigGenTrigSource attribute), 288 NotchFilter1 (msl.equipment.resources.thorlabs.kinesis.enums.PC_ATTRIBUTE, 288)

NONE (msl.equipment.resources.picotech.picoscope.enums.PS4000A_HoldDirection attribute), 290 notchFilter10n

NONE (msl.equipment.resources.picotech.picoscope.enums.PS4000C_GroupType attribute), 274 notchFilter10n (msl.equipment.resources.thorlabs.kinesis.enums.PC_ATTRIBUTE, 274)

NONE (msl.equipment.resources.picotech.picoscope.enums.PS4000C_HoldDirection attribute), 272 notchFilter10n (msl.equipment.resources.thorlabs.kinesis.enums.PC_ATTRIBUTE, 272)

NONE (msl.equipment.resources.picotech.picoscope.enums.PS4000C_PulseWidthType attribute), 279 (msl.equipment.resources.thorlabs.kinesis.structs.PPC_NonAttribute), 580

NONE (msl.equipment.resources.picotech.picoscope.enums.PS4000C_RatedMode attribute), 278 NotchFilterBoth

NONE (msl.equipment.resources.picotech.picoscope.enums.PS4000C_SigGenTrigSources attribute), 276 (msl.equipment.resources.thorlabs.kinesis.enums.PC_NonAttribute), 448

NONE (msl.equipment.resources.picotech.picoscope.enums.PS4000C_TriggerFrequency attribute), 278 (msl.equipment.resources.thorlabs.kinesis.structs.QD_LockAttribute), 448

NONE (msl.equipment.resources.picotech.picoscope.enums.PS5000A_PulseWidthType attribute), 313 notchFilterCenterFrequency

NONE (msl.equipment.resources.picotech.picoscope.enums.PS5000A_RatedMode attribute), 312 (msl.equipment.resources.thorlabs.kinesis.structs.QD_NonAttribute), 589

NONE (msl.equipment.resources.picotech.picoscope.enums.PS5000A_SigGenTrigSource attribute), 309 (msl.equipment.resources.thorlabs.kinesis.structs.QD_LockAttribute), 589

NONE (msl.equipment.resources.picotech.picoscope.enums.PS5000A_HoldDirection attribute), 311 notchFilterEnabled

NONE (msl.equipment.resources.picotech.picoscope.enums.PS5000C_PulseWidthType attribute), 303 (msl.equipment.resources.thorlabs.kinesis.structs.QD_NonAttribute), 589

NONE (msl.equipment.resources.picotech.picoscope.enums.PS5500C_RatedMode attribute), 302 (msl.equipment.resources.thorlabs.kinesis.enums.PPC_NonAttribute), 589

NONE (msl.equipment.resources.picotech.picoscope.enums.PS5500C_SigGenTrigSource attribute), 300 NotchFilterOn (msl.equipment.resources.thorlabs.kinesis.enums.PC_ATTRIBUTE, 300)

NONE (msl.equipment.resources.picotech.picoscope.enums.PS5500C_HoldDirection attribute), 302 notchFilterQ (msl.equipment.resources.thorlabs.kinesis.structs.QD_NonAttribute), 589

NONE (msl.equipment.resources.picotech.picoscope.enums.PS6000A_PulseWidthType attribute), 323 notchFilterQ (msl.equipment.resources.thorlabs.kinesis.structs.QD_NonAttribute), 589

NONE (msl.equipment.resources.picotech.picoscope.enums.PS6000A_RatedMode attribute), 322 notes (msl.equipment.resources.thorlabs.kinesis.structs.TLI_HardAttribute), 589

NONE (msl.equipment.resources.picotech.picoscope.enums.PS6000A_SigGenTrigSource attribute), 320 notUsed (msl.equipment.resources.thorlabs.kinesis.structs.MOT_BAttribute), 589

NONE (msl.equipment.resources.picotech.picoscope.enums.PS6000A_HoldDirection attribute), 321 notUsed (msl.equipment.resources.thorlabs.kinesis.structs.MOT_BAttribute), 589

NONE (msl.equipment.resources.thorlabs.kinesis.enums.ChannelEnableMode attribute), 456 notUsed (msl.equipment.resources.thorlabs.kinesis.structs.MOT_BAttribute), 589

NOT_FOUND (msl.equipment.resources.picotech.errors.PS2000Attribute), 574 notUsed (msl.equipment.resources.thorlabs.kinesis.structs.MOT_BAttribute), 589

NOT_FOUND (msl.equipment.resources.picotech.errors.PS3000Attribute), 575 notUsed (msl.equipment.resources.thorlabs.kinesis.structs.MOT_VAttribute), 589

NOT_RESPONDING (msl.equipment.resources.picotech.errors.PS2000InUse attribute), 228 (msl.equipment.resources.thorlabs.kinesis.structs.NTAttribute), 579

NOT_RESPONDING (msl.equipment.resources.picotech.errors.PS3000InUse attribute), 228 (msl.equipment.resources.thorlabs.kinesis.structs.NTAttribute), 579

notYetInUse (<i>msl.equipment.resources.thorlabs.kinesis.enums.NT_AutoRangeAtSelected</i> attribute), 592	<i>attribute</i>), 443
NR360 (<i>msl.equipment.resources.thorlabs.kinesis.enums.NT_AutoRangeAtParameter</i> attribute), 472	(<i>msl.equipment.resources.thorlabs.kinesis.enums.NT_TIA</i>)
NR360 (<i>msl.equipment.resources.thorlabs.kinesis.enums.TST_Start</i> attribute), 442	<i>NT_AutoRangeAtSelected</i>
NR_ANALOG_OUTPUTS	(<i>msl.equipment.resources.thorlabs.kinesis.enums.NT_TIA</i>)
(<i>msl.equipment.resources.avantes.avaspec.Avantes</i> attribute), 108	<i>NT_BadSignal</i> (<i>msl.equipment.resources.thorlabs.kinesis.enums.NT_TIA</i>)
NR_DAC_POL_COEF	<i>NT_BadSignal</i> (<i>msl.equipment.resources.thorlabs.kinesis.enums.NT_TIA</i>)
(<i>msl.equipment.resources.avantes.avaspec.NT_BNC_10v</i> (<i>msl.equipment.resources.thorlabs.kinesis.enums.NT_TIA</i>) attribute), 106	<i>attribute</i>), 438
NR_DEFECTIVE_PIXELS	<i>NT_BNC_1v</i> (<i>msl.equipment.resources.thorlabs.kinesis.enums.NT_F</i>)
(<i>msl.equipment.resources.avantes.avaspec.Avantes</i> attribute), 439	
(<i>msl.equipment.resources.avantes.avaspec.NT_BNC_2v</i> (<i>msl.equipment.resources.thorlabs.kinesis.enums.NT_F</i>) attribute), 106	<i>attribute</i>), 439
NR_DIGITAL_INPUTS	<i>NT_BNC_5v</i> (<i>msl.equipment.resources.thorlabs.kinesis.enums.NT_F</i>)
(<i>msl.equipment.resources.avantes.avaspec.NT_BNC_5v</i> (<i>msl.equipment.resources.thorlabs.kinesis.enums.NT_F</i>) attribute), 107	<i>attribute</i>), 439
NR_DIGITAL_OUTPUTS	<i>NT_BNCModeLVOut</i>
(<i>msl.equipment.resources.avantes.avaspec.Avantes</i> attribute), 445	(<i>msl.equipment.resources.thorlabs.kinesis.enums.BNT_BN</i> attribute), 445
NR_NONLIN_POL_COEF	<i>NT_BNCModeTrigger</i>
(<i>msl.equipment.resources.avantes.avaspec.Avantes</i> attribute), 106	(<i>msl.equipment.resources.thorlabs.kinesis.enums.BNT_BN</i> attribute), 445
NR_TEMP_POL_COEF	<i>NT_CircleAdjustment</i> (class in <i>msl.equipment.resources.thorlabs.kinesis.enums</i>), 441
NR_WAVELEN_POL_COEF	<i>NT_CircleDiameterLUT</i> (class in <i>msl.equipment.resources.thorlabs.kinesis.structs</i>), 577
NS (<i>msl.equipment.resources.picotech.picoscope.enums.NT_PSD0001TimeUnits</i> attribute), 245	<i>TimeMode</i> (class in <i>msl.equipment.resources.thorlabs.kinesis.enums</i>),
NS (<i>msl.equipment.resources.picotech.picoscope.enums.PS2000TimeUnits</i> attribute), 234	<i>NT_CircleParameters</i> (class in <i>msl.equipment.resources.thorlabs.kinesis.enums</i>),
NS (<i>msl.equipment.resources.picotech.picoscope.enums.PS3000TimeUnits</i> attribute), 262	<i>NT_CircleUnit</i> (class in <i>msl.equipment.resources.thorlabs.kinesis.structs</i>), 577
NS (<i>msl.equipment.resources.picotech.picoscope.enums.NT_PSD0001Loop</i> (<i>msl.equipment.resources.thorlabs.kinesis.enums</i> attribute), 253	<i>attribute</i>), 439
NS (<i>msl.equipment.resources.picotech.picoscope.enums.NT_PSD0001Smoothed</i> attribute), 284	(<i>msl.equipment.resources.thorlabs.kinesis.enums.NT_Con</i>)
NS (<i>msl.equipment.resources.picotech.picoscope.enums.PS4000Tributary</i> attribute), 273	<i>ControlMode</i> (class in <i>msl.equipment.resources.thorlabs.kinesis.enums</i>), 439
NS (<i>msl.equipment.resources.picotech.picoscope.enums.PS5000Tributary</i> attribute), 308	<i>NT_ControlMode</i> (class in <i>msl.equipment.resources.thorlabs.kinesis.enums</i>), 439
NS (<i>msl.equipment.resources.picotech.picoscope.enums.NT_PSD0001Mode</i> attribute), 298	<i>NT_ControlMode</i> (class in <i>msl.equipment.resources.thorlabs.kinesis.enums</i>), 439
NS (<i>msl.equipment.resources.picotech.picoscope.enums.PS6000Tributary</i> attribute), 317	<i>NT_CubeCircleAdjustment</i>
NT_AbsPowerCircleMode	(<i>msl.equipment.resources.thorlabs.kinesis.enums.NT_Circ</i>)
(<i>msl.equipment.resources.thorlabs.kinesis.enums.NT_GurableDiameterMode</i> attribute), 441	<i>attribute</i>), 442
	<i>NT_CurrentLimit_100mA</i>

(*msl.equipment.resources.thorlabs.kinesis.enums.NT_MOTOR_CURRENT_LIMIT* (class in *msl.equipment.resources.thorlabs.kinesis.structs*), 444
NT_CurrentLimit_250mA (*msl.equipment.resources.thorlabs.kinesis.enums.NT_MOTOR_CURRENT_LIMIT* (class in *msl.equipment.resources.thorlabs.kinesis.enums.NT_MOTOR_CURRENT_LIMIT* attribute), 438
NT_CurrentLimit_500mA (*msl.equipment.resources.thorlabs.kinesis.enums.NT_MOTOR_CURRENT_LIMIT* (class in *msl.equipment.resources.thorlabs.kinesis.enums.NT_CIRCLE_MODE* attribute), 441
NT_Db (*msl.equipment.resources.thorlabs.kinesis.enums.NT_MOTOR_CURRENT_LIMIT* (class in *msl.equipment.resources.thorlabs.kinesis.enums.NT_CIRCLE_MODE* attribute), 443
NT_Even (*msl.equipment.resources.thorlabs.kinesis.enums.NT_MOTOR_CURRENT_LIMIT* (class in *msl.equipment.resources.thorlabs.kinesis.enums.NT_CIRCLE_MODE* attribute), 440
NT_FeedbackSignalAC (*msl.equipment.resources.thorlabs.kinesis.enums.NT_MOTOR_CURRENT_LIMIT* (class in *msl.equipment.resources.thorlabs.kinesis.enums.NT_LOW_PASS_FILTER_PARAMETERS* attribute), 445
NT_FeedbackSignalDC (*msl.equipment.resources.thorlabs.kinesis.enums.NT_MOTOR_CURRENT_LIMIT* (class in *msl.equipment.resources.thorlabs.kinesis.enums.NT_LOW_PASS_FILTER_PARAMETERS* attribute), 445
NT_FeedbackSource (class in *msl.equipment.resources.thorlabs.kinesis.enums.NT_MOTOR_CURRENT_SOURCE* attribute), 439
NT_FeedbackSourceUndefined (*msl.equipment.resources.thorlabs.kinesis.enums.NT_MOTOR_CURRENT_SOURCE* attribute), 461
NT_FeedbackSourceUndefined (*msl.equipment.resources.thorlabs.kinesis.enums.NT_MOTOR_CURRENT_SOURCE* attribute), 439
NT_FullRange (*msl.equipment.resources.thorlabs.kinesis.enums.NT_MOTOR_CURRENT_SOURCE* attribute), 463
NT_FullRange (*msl.equipment.resources.thorlabs.kinesis.enums.NT_MOTOR_CURRENT_SOURCE* attribute), 444
NT_GainParameters (class in *msl.equipment.resources.thorlabs.kinesis.structs.NT_MOTOR_CURRENT_SOURCE* attribute), 439
NT_GainParameters (class in *msl.equipment.resources.thorlabs.kinesis.structs.NT_MOTOR_CURRENT_SOURCE* attribute), 442
NT_GoodSignal (*msl.equipment.resources.thorlabs.kinesis.enums.NT_MOTOR_CURRENT_SOURCE* attribute), 438
NT_HorizontalTracking (*msl.equipment.resources.thorlabs.kinesis.enums.NT_MOTOR_CURRENT_SOURCE* attribute), 441
NT_HubOrSMA (*msl.equipment.resources.thorlabs.kinesis.enums.NT_MOTOR_CURRENT_SOURCE* attribute), 443
NT_HVComponent (class in *msl.equipment.resources.thorlabs.kinesis.structs.NT_MOTOR_CURRENT_SOURCE* attribute), 442
NT_InRange (*msl.equipment.resources.thorlabs.kinesis.enums.NT_MOTOR_CURRENT_SOURCE* attribute), 441
NT_I01_5v (*msl.equipment.resources.thorlabs.kinesis.enums.NT_MOTOR_CURRENT_SOURCE* attribute), 461
NT_I01_Units (class in *msl.equipment.resources.thorlabs.kinesis.enums.NT_MOTOR_CURRENT_SOURCE* attribute), 438
NT_Odd (*msl.equipment.resources.thorlabs.kinesis.enums.NT_MOTOR_CURRENT_SOURCE* attribute), 463
NT_LinearCircleAdjustment (*msl.equipment.resources.thorlabs.kinesis.enums.NT_CIRCLE_MODE* attribute), 441
NT_LowPass_100Hz (*msl.equipment.resources.thorlabs.kinesis.enums.NT_CIRCLE_MODE* attribute), 441
NT_LowPass_10Hz (*msl.equipment.resources.thorlabs.kinesis.enums.NT_CIRCLE_MODE* attribute), 441
NT_LowPass_1Hz (*msl.equipment.resources.thorlabs.kinesis.enums.NT_CIRCLE_MODE* attribute), 441
NT_LowPass_30Hz (*msl.equipment.resources.thorlabs.kinesis.enums.NT_CIRCLE_MODE* attribute), 442
NT_LowPassFilterParameters (class in *msl.equipment.resources.thorlabs.kinesis.structs.NT_CIRCLE_MODE* attribute), 438
NT_LUTCircleMode (*msl.equipment.resources.thorlabs.kinesis.enums.NT_CIRCLE_MODE* attribute), 442
NT_ManualRangeAtParameter (*msl.equipment.resources.thorlabs.kinesis.enums.NT_CIRCLE_MODE* attribute), 442
NT_ModeUnderOrOver (class in *msl.equipment.resources.thorlabs.kinesis.enums.NT_CIRCLE_MODE* attribute), 442
NT_ModeUndefined (*msl.equipment.resources.thorlabs.kinesis.enums.NT_CIRCLE_MODE* attribute), 442
NT_Odd (*msl.equipment.resources.thorlabs.kinesis.enums.NT_CIRCLE_MODE* attribute), 442

attribute), 440

NT_OddAndEven (*msl.equipment.resources.thorlabs.kinesis.enums.NT_TIAOddOrEven*),
attribute), 440

NT_OddOrEven (*class* *in* **NT_TIARange10_100uA**
msl.equipment.resources.thorlabs.kinesis.enums), (*msl.equipment.resources.thorlabs.kinesis.enums.NT_TIAOddOrEven*),
attribute), 440

NT_OpenLoop (*msl.equipment.resources.thorlabs.kinesis.enums.NT_TIARange1_ClosedLoopMode*
attribute), 439

NT_OpenLoopSmoothed (*msl.equipment.resources.thorlabs.kinesis.enums.NT_TIARange12_Mode*
attribute), 439

NT_OutputFilter_100Hz (*msl.equipment.resources.thorlabs.kinesis.enums.NT_TIARange13_BandPassFilter*
attribute), 444

NT_OutputFilter_10Hz (*msl.equipment.resources.thorlabs.kinesis.enums.NT_TIARange14_BandPassFilter*
attribute), 444

NT_OutputFilter_5kHz (*msl.equipment.resources.thorlabs.kinesis.enums.NT_TIARange15_BandPassFilter*
attribute), 445

NT_OutputFilter_None (*msl.equipment.resources.thorlabs.kinesis.enums.NT_TIARange16_BandPassFilter*
attribute), 445

NT_OutputVoltageRoute (*class* *in* **NT_TIARange3_30nA**
msl.equipment.resources.thorlabs.kinesis.enums.NT_TIAUnderOrOver,
attribute), 443

NT_OverRange (*msl.equipment.resources.thorlabs.kinesis.enums.NT_TIAUnderOrOver*,
attribute), 441

NT_ParameterCircleMode (*msl.equipment.resources.thorlabs.kinesis.enums.NT_TIAParameterMode*,
attribute), 441

NT_Piezo (*msl.equipment.resources.thorlabs.kinesis.enums.NT_TIAEquipment*),
attribute), 438

NT_PowerInputUnits (*class* *in* **NT_TIARange6_1uA**
msl.equipment.resources.thorlabs.kinesis.enums), (*msl.equipment.resources.thorlabs.kinesis.enums.NT_TIAParameterMode*),
attribute), 443

NT_SignalState (*class* *in* **NT_TIARange7_3uA**
msl.equipment.resources.thorlabs.kinesis.enums), (*msl.equipment.resources.thorlabs.kinesis.enums.NT_TIAParameterMode*),
attribute), 438

NT_SMA_Units (*class* *in* **NT_TIARange8_10uA**
msl.equipment.resources.thorlabs.kinesis.enums), (*msl.equipment.resources.thorlabs.kinesis.enums.NT_TIAParameterMode*),
attribute), 444

NT_SMAOnly (*msl.equipment.resources.thorlabs.kinesis.enums.NT_TIARange9_100nA*,
attribute), 443

NT_SquareCircleAdjustment (*msl.equipment.resources.thorlabs.kinesis.enums.NT_TIARange10_Adjustment* (*class* *in*
msl.equipment.resources.thorlabs.kinesis.enums.NT_TIAParameterMode),
attribute), 442

NT_TIA (*msl.equipment.resources.thorlabs.kinesis.enums.KNAFeedbackSource*,
attribute), 461

NT_TIA (*msl.equipment.resources.thorlabs.kinesis.enums.NT_TIAFeedbackSource*),
attribute), 439

NT_TIARangeParameters (class in attribute), 571
msl.equipment.resources.thorlabs.kinesis.structs.numCycles (msl.equipment.resources.thorlabs.kinesis structs.PZ_LUT attribute), 579
578

NT_TIAReading (class in numCycles (msl.equipment.resources.thorlabs.kinesis structs.SC_Cycle attribute), 591
msl.equipment.resources.thorlabs.kinesis structs), attribute), 578
578 Numerator (msl.equipment.resources.nkt.nktpdll.ParameterSetType attribute), 63

NT_Tracking (msl.equipment.resources.thorlabs.kinesis.enums.NT_TrackingMode attribute), 438
attribute), 438 numOutTriggerRepeat

NT_UnderOrOver (class in (msl.equipment.resources.thorlabs.kinesis structs.PZ_LUT attribute), 579
msl.equipment.resources.thorlabs.kinesis enums), attribute), 440
440

NT_UnderRange (msl.equipment.resources.thorlabs.kinesis enums.NT_UnderOrOver attribute), 441
attribute), 441 ODD (msl.equipment.constants.Parity attribute), 50

NT_UserDefined OEM_DATA_LEN (msl.equipment.resources.avantes.avaspec.Avantes (msl.equipment.resources.thorlabs.kinesis enums.NT_UserDefined attribute), 106
attribute), 463 OemDataType (class in msl.equipment.resources.avantes.avaspec),

NT_UserDefined (msl.equipment.resources.thorlabs.kinesis enums.NT_ISMA_Units attribute), 444 OFF (msl.equipment.resources.picotech.picoscope enums.PS2000AAttribute), 244

NT_VerticalTracking OFF (msl.equipment.resources.thorlabs.kinesis enums.NT_Equipment resources.picotech.picoscope enums.PS2000AAttribute), 246
attribute), 439 attribute), 246

NT_Voltage OFF (msl.equipment.resources.thorlabs.kine (msl.equipment.resources.picotech.picoscope enums.PS2000EAttribute), 236
attribute), 463 attribute), 236

NT_Voltage OFF (msl.equipment.resources.thorlabs.kine (msl.equipment.resources.picotech.picoscope enums.PS3000AAttribute), 262
attribute), 444 attribute), 262

NT_VoltageRange OFF (msl.equipment.resources.picotech.picoscope enums.PS3000AAttribute), 264
msl.equipment.resources.thorlabs.kinesis enums), attribute), 264
442 OFF (msl.equipment.resources.picotech.picoscope enums.PS3000EAttribute), 255
attribute), 255

NT_VoltageRange_10v OFF (msl.equipment.resources.picotech.picoscope enums.PS4000AAttribute), 286
msl.equipment.resources.thorlabs.kinesis enums.NT_VoltageRange attribute), 443 attribute), 286

NT_VoltageRange_5v OFF (msl.equipment.resources.picotech.picoscope enums.PS4000AAttribute), 284
msl.equipment.resources.thorlabs.kinesis enums.NT_VoltageRange attribute), 443 attribute), 284

OFF (msl.equipment.resources.picotech.picoscope enums.PS4000AAttribute), 280
NT_VoltageRangeUndefined OFF (msl.equipment.resources.picotech.picoscope enums.PS4000EAttribute), 273
msl.equipment.resources.thorlabs.kinesis enums.NT_VoltageRange attribute), 443 attribute), 273

NT_Watts OFF (msl.equipment.resources.picotech.picoscope enums.PS5000AAttribute), 307
msl.equipment.resources.thorlabs.kinesis enums.NT_PowerInPulses attribute), 443 attribute), 307

NTC1_ID OFF (msl.equipment.resources.picotech.picoscope enums.PS5000AAttribute), 304
msl.equipment.resources.avantes.avaspec (msl.equipment.resources.picotech.picoscope enums.PS5000EAttribute), 107 attribute), 107

NTC2_ID OFF (msl.equipment.resources.picotech.picoscope enums.PS5000EAttribute), 298
msl.equipment.resources.avantes.avaspec (msl.equipment.resources.picotech.picoscope enums.PS5000EAttribute), 108 attribute), 108

NULL (msl.equipment.vxi11.OperationFlag at- OFF (msl.equipment.resources.picotech.picoscope enums.PS6000EAttribute), 317
tribute), 602 attribute), 317

num_locks (msl.equipment.hislip.AsyncLockInfoResponse (msl.equipment.resources.picotech.picoscope enums.PS6000EAttribute), 319
property), 64 attribute), 319

num_samples (msl.equipment.resources.picotech.picoscope (msl.equipment.picotech.picoscope enums.Pt104.Pt104DataTypeAttribute), 371
property), 232 attribute), 371

numChannels (msl.equipment.resources.thorlabs.kin OFF (msl.equipment.services.mspifpInHardwareInformation fwxx2c.SensorMode attribute), 63

attribute), 594
0ff(*msl.equipment.resources.thorlabs.kinesis.enums.0FF*) *DisplayOutputIntensity*.resources.thorlabs.kinesis.kcube_dc_servo.
attribute), 450
off()(*msl.equipment.resources.raicol.raicol_tec.RaicolTEC*) *msl.equipment.resources.thorlabs.kinesis.kcube_solenoid.*
method), 401
offset(*msl.equipment.resources.nkt.nktpdll.Parameter*) *openDynamixel**msl.equipment.resources.thorlabs.kinesis.kcube_stepper_r*
attribute), 163
offset_correction()
(msl.equipment.resources.greisinger.gmh3000.GMH3000Mod), 378
method), 149
offsetDistance
(msl.equipment.resources.thorlabs.kinesis.structs.MOTOffsetHoldingParameters
attribute), 572
OK(*msl.equipment.resources.picotech.errors.PS2000Error*) *msl.equipment.resources.princeton_instruments.arc_instr*
attribute), 228
OK(*msl.equipment.resources.picotech.errors.PS3000Error*) *Open_mono()* *msl.equipment.resources.princeton_instruments.arc*
attribute), 228
OL756 *(class* *in open_mono_port()*
*msl.equipment.resources.optronic_laboratories.ol756**(msl.equipment.resources.princeton_instruments.arc_instr*
206
OL756 *(class* *in open_mono_serial()*
*msl.equipment.resources.optronic_laboratories.ol756**(msl.equipment.resources.princeton_instruments.arc_instr*
224
OLCurrentSource *(class* *in open_ports()* (*msl.equipment.resources.nkt.nktpdll.NKT*
msl.equipment.resources.optronic_laboratories.ol_current_source_method), 178
225
OmegaError, 55
ON(*msl.equipment.resources.thorlabs.fwxx2c.SensorMode*) *method), 378*
attribute), 594
on()(*msl.equipment.resources.raicol.raicol_tec.RaicolTEC*) *msl.equipment.resources.princeton_instruments.arc_instr*
method), 401
ONE (*msl.equipment.constants.StopBits* *attribute), 51*
open_unit() (*msl.equipment.resources.picotech.picoscope.picoscope*
method), 330
ONE_POINT_FIVE
(msl.equipment.constants.StopBits *attribute), 51*
open_unit() (*msl.equipment.resources.picotech.picoscope.picoscope*
method), 336
online() (*msl.equipment.connection_gpib.ConnectionGPIB*) (*msl.equipment.resources.picotech.picoscope.picoscope_2*
method), 330
OP_200Hz (*msl.equipment.resources.thorlabs.kinesis.opensource.OP200Hz*) *openUnitBandwidth*
attribute), 449
OP_Unfiltered (*msl.equipment.resources.thorlabs.kinesis.emulation.OPUnfiltered*) *C310OutputBandwidth*
attribute), 449
open() (*msl.equipment.resources.picotech.pt104.PT104*) *open_unit_async_ex()*
method), 373
open() (*msl.equipment.resources.thorlabs.fwxx2c.FilterWhirlwind2c*)
method), 596
open() (*msl.equipment.resources.thorlabs.kinesis.benchtop_stapler*) *openUnitProgress()*
method), 416
open() (*msl.equipment.resources.thorlabs.kinesis.filter_flipper*) *openUnitProgress()*
method), 478
open() (*msl.equipment.resources.thorlabs.kinesis.integrated_stepper_motors*) *IntegratedStepperMotors*
method), 330

(*msl.equipment.resources.picotech.picoscope.picoscopetrioipiPicoscopeApi*
method), 336 OUTSIDE (*msl.equipment.resources.picotech.picoscope.enums.PS3000AThresholdDirection*)
open_via_ip() (*msl.equipment.resources.picotech.pt104.PT104Attribute*), 256
method), 373 OUTSIDE (*msl.equipment.resources.picotech.picoscope.enums.PS4000AThresholdDirection*)
openLoopOption attribute), 289
(msl.equipment.resources.thorlabs.kinesis.s OUTSIDE (*msl.equipment.resources.picotech.picoscope.enums.PS4000AThresholdDirection*)
attribute), 590 attribute), 277
openTime (*msl.equipment.resources.thorlabs.kinesis* OUTSIDE (*msl.equipment.resources.picotech.picoscope.enums.PS5000AThresholdDirection*)
attribute), 591 attribute), 311
OperationFlag (*class in msl.equipment.vxi11*), OUTSIDE (*msl.equipment.resources.picotech.picoscope.enums.PS5000AThresholdDirection*)
601 attribute), 302
Optical (*msl.equipment.resources.thorlabs.kinesis.e* OUTSIDE (*msl.equipment.resources.picotech.picoscope.enums.PS6000AThresholdDirection*)
attribute), 449 attribute), 321
OPTICAL_SWITCH outTriggerDuration
(msl.equipment.resources.picotech.picoscope.enums.P OUTSIDE (*msl.equipment.resources.thorlabs.kinesis.structs.PZ_LUT*)
attribute), 272 attribute), 580
OptoSigmaError, 55 outTriggerStart
OptronicLaboratoriesError, 55 (*msl.equipment.resources.thorlabs.kinesis.structs.PZ_LUT*)
OR (*msl.equipment.resources.picotech.picoscope.enums.PS2000AThresholdOper*
attribute), 241 overlapped (*msl.equipment.hislip.InitializeResponse*
OS_NOT_SUPPORTED property), 61
(msl.equipment.resources.picotech.errors.P OUT_OF_RANGE (*msl.equipment.resources.picotech.pt104.PMSLPS2000APulseWidthType* method),
attribute), 228 60
OS_NOT_SUPPORTED (*msl.equipment.resources.picotech.errors.PS3000Error* P
attribute), 229
OUT_OF_RANGE (*msl.equipment.resources.picotech.pt104.PMSLPS2000APulseWidthType* method),
attribute), 251 60
OUT_OF_RANGE (*msl.equipment.resources.picotech.pt104.PMSLPS2000APulseWidthType* attribute), 578
attribute), 239
OUT_OF_RANGE (*msl.equipment.resources.picotech.pt104.PMSLPS2000APulseWidthType* attribute), 578
attribute), 268
OUT_OF_RANGE (*msl.equipment.resources.picotech.pt104.PMSLPS2000APulseWidthType* attribute), 578
attribute), 257
OUT_OF_RANGE (*msl.equipment.resources.picotech.pt104.PMSLPS2000APulseWidthType* attribute), 578
attribute), 292
OUT_OF_RANGE (*msl.equipment.resources.picotech.pt104.PMSLPS2000APulseWidthType* attribute), 578
attribute), 279
OUT_OF_RANGE (*msl.equipment.resources.picotech.pt104.PMSLPS2000APulseWidthType* attribute), 578
attribute), 313
OUT_OF_RANGE (*msl.equipment.resources.picotech.picoscope.attributePS5000PulseWidthType* ParameterSetType (class in
attribute), 303
OUT_OF_RANGE (*msl.equipment.resources.picotech.picoscope.mslapiPS6000APulseWidthType* attribute), 163
attribute), 323
OUTPUT (*msl.equipment.resources.thorlabs.fwxx2c.TriggeredUnitTypes* ParameterSetUnitTypes (class in
attribute), 594 msl.equipment.resources.nkt.nktpdll),
OUTSIDE (*msl.equipment.resources.picotech.picoscope.enums.PS2000AThresholdDirection* PS
attribute), 249 Parity (class in msl.equipment.constants), 50
OUTSIDE (*msl.equipment.resources.picotech.picoscope.parity.PS2000AThresholdDirection* ConnectionSerial
attribute), 238 property), 40
OUTSIDE (*msl.equipment.resources.picotech.picoscope.parity.PS3000AThresholdDirection* bentham.benhw32.Bentham32)

method), 129
park() (*msl.equipment.resources.bentham.benhw64.Bentham*) *method), 416*
method), 131
parse_address()
(msl.equipment.connection.Connection
static method), 21
parse_address()
(msl.equipment.connection_gpib.ConnectionGPIB
static method), 26
parse_address()
(msl.equipment.connection_prologix.ConnectionPrologix
static method), 36
parse_address()
(msl.equipment.connection_sdk.ConnectionSDK
static method), 39
parse_address()
(msl.equipment.connection_serial.ConnectionSerial
static method), 40
parse_address()
(msl.equipment.connection_socket.ConnectionSocket
static method), 42
parse_address()
(msl.equipment.connection_tcpip_hislip.ConnectionTCP/IP_HISLIP
static method), 43
parse_address()
(msl.equipment.connection_tcpip_vxi11.ConnectionTCP/IP_VXI11
static method), 46
parse_address()
(msl.equipment.connection_zeromq.ConnectionZeroMQ)
static method), 49
parse_lxi_webserver() (*in module*
msl.equipment.utils), 601
parse_pico_scope_api_header() (*in module*
msl.equipment.resources.picotech.picoscope.helper), 344
323
partNumber (*msl.equipment.resources.thorlabs.kinesis.structs.MOTQuiescenceParameter*) *attribute), 573*
329
pass_control()
(msl.equipment.connection_gpib.ConnectionGPIB
method), 26
PATH (*msl.equipment.config.Config* attribute), 18
path (*msl.equipment.config.Config* property), 19
path (*msl.equipment.connection_sdk.ConnectionSDK*
property), 38
path (*msl.equipment.database.Database* prop-
erty), 52
PENDING (*msl.equipment.resources.picotech.picoscope.PicoScope4000OpenProgress* &
class attribute), 236
PENDING (*msl.equipment.resources.picotech.picoscope.enums.PS13000OpenProgress*
attribute), 255
persist_settings()
(msl.equipment.resources.thorlabs.kinesis.benchtop_stepper_
method), 479
persist_settings()
(msl.equipment.resources.thorlabs.kinesis.filter_flipper.Fil-
method), 494
persist_settings()
(msl.equipment.resources.thorlabs.kinesis.integrated_step-
method), 517
persist_settings()
(msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo-
method), 534
persist_settings()
(msl.equipment.resources.thorlabs.kinesis.kcube_solenoid-
method), 551
phase (*msl.equipment.resources.thorlabs.kinesis structs.MOT_Bru-*
attribute), 575
PICOPE2000_OLD
(msl.equipment.resources.picotech.errors.PS2000Error
attribute), 228
PICOPE2000
(msl.equipment.resources.picotech.errors.PS3000Error
attribute), 229
PicoScope1000
(class in
msl.equipment.resources.picotech.picoscope.picoscope),
325
PicoScope2000
(class in
msl.equipment.resources.picotech.picoscope.ps2000),
342
PicoScope2k3k
(class in
msl.equipment.resources.picotech.picoscope.ps2000a),
344
PicoScope3000
(class in
msl.equipment.resources.picotech.picoscope.ps3000),
345
PicoScope3000A
(class in
msl.equipment.resources.picotech.picoscope.ps3000a),
347
PicoScope4000
(class in
msl.equipment.resources.picotech.picoscope.ps4000),
349
PicoScope4000OpenProgress
(class in
msl.equipment.resources.picotech.picoscope.ps4000a),
351
PicoScope5000
(class in
msl.equipment.resources.picotech.picoscope.ps5000),
353

	353		
PicoScope5000A	(class	in	(<i>msl.equipment.resources.nkt.nktpdll.NKT method</i>), 179
	<i>msl.equipment.resources.picotech.picoscope5000A::point_port_get()</i>		
	354		(<i>msl.equipment.resources.nkt.nktpdll.NKT attribute</i>), 585
PicoScope6000	(class	in	(<i>method</i>), 179
	<i>msl.equipment.resources.picotech.picoscope6000A::y1</i> (<i>msl.equipment.resources.thorlabs.kinesis structs.KLD attribute</i>), 585		
	356		
PicoScopeApi	(class	in	<i>polarity1</i> (<i>msl.equipment.resources.thorlabs.kinesis structs.KLS mslice</i>), 585
	<i>msl.equipment.resources.picotech.picoscope::polar1::polarity1</i>), 585		
	331		<i>polarity2</i> (<i>msl.equipment.resources.thorlabs.kinesis structs.KLD attribute</i>), 585
PicoScopeChannel	(class	in	(<i>attribute</i>), 585
	<i>msl.equipment.resources.picotech.picoscope::channel::polarity2</i> (<i>msl.equipment.resources.thorlabs.kinesis structs.KLS attribute</i>), 585		
	230		
PicoScopeInfoApi	(class	in	<i>poll_scan()</i> (<i>msl.equipment.resources.avantes.avaspec.Avantes mslice</i>), 122
	<i>msl.equipment.resources.picotech.picoscope.enums</i>), <i>method</i>), 122		
	232		<i>polling_duration()</i>
PicoTechError	, 56		(<i>msl.equipment.resources.thorlabs.kinesis.benchtop_stepper attribute</i>), 570
PID	(<i>msl.equipment.resources.thorlabs.kinesis structs.TLI_DemodInfb</i>), 417		<i>polling_duration()</i>
	<i>attribute</i>), 570		
PIDConstsSD	(<i>msl.equipment.resources.thorlabs.kinesis structs.RPLCqPIDConsts</i>), 481		<i>method</i>), 481
	<i>attribute</i>), 580		
PIDConstsDFC	(<i>msl.equipment.resources.thorlabs.kinesis structs.PollingDurationPIDConsts</i>), 580		(<i>msl.equipment.resources.thorlabs.kinesis.integrated_steppe attribute</i>), 580
	<i>attribute</i>), 580		
PIDConstsI	(<i>msl.equipment.resources.thorlabs.kinesis structs.RPLCqPIDConsts</i>), 580		<i>polling_duration()</i>
	<i>attribute</i>), 580		
PIDConstsP	(<i>msl.equipment.resources.thorlabs.kinesis structs.RPLCqPIDConsts</i>), 517		<i>method</i>), 517
	<i>attribute</i>), 580		
PIDDerivFilterOn			<i>polling_duration()</i>
	<i>msl.equipment.resources.thorlabs.kinesis structs.PPGnPIDConsts</i>), 534		<i>method</i>), 534
	<i>attribute</i>), 580		
PIDOutputLimit			<i>polling_duration()</i>
	<i>msl.equipment.resources.thorlabs.kinesis structs.MQTrsPIDLipodInredenParathorlabs.kinesis.kcube stepper attribute</i>), 581		<i>method</i>), 551
	<i>attribute</i>), 581		
PIDTolerance	(<i>msl.equipment.resources.thorlabs.kinesis structs.MQTrsPIDLipodInredenParathorlabs.kinesis.kcube ConnectionSocket attribute</i>), 581		<i>property</i>), 42
	<i>attribute</i>), 581		
ping_unit()	(<i>msl.equipment.resources.picotech.picoscope::ipclikequip::tcpip_hislip.Connection_tcip_hislip.ConnectionTCPIPH attribute</i>), 326		<i>property</i>), 43
	<i>method</i>), 326		
PLS_X25MM	(<i>msl.equipment.resources.thorlabs.kinesis structs.KSQuStage</i>), 46		<i>connection_tcip_vxi11.ConnectionTCPIPVX attribute</i>), 472
	<i>attribute</i>), 472		
PLS_X25MM	(<i>msl.equipment.resources.thorlabs.kinesis structs.KSQuStage</i>), 49		<i>connection_zeromq.ConnectionZeroMQ attribute</i>), 477
	<i>attribute</i>), 477		
PLS_X25MM_HiRes			<i>port</i> (<i>msl.equipment.resources.avantes.avaspec.Avantes BroadcastAnswerT attribute</i>), 99
	<i>msl.equipment.resources.thorlabs.kinesis enums.KSTitStage</i>), 110		
	<i>attribute</i>), 472		<i>port</i> (<i>msl.equipment.resources.avantes.avaspec.BroadcastAnswerT attribute</i>), 99
PLS_X25MM_HiRes			
	<i>msl.equipment.resources.thorlabs.kinesis enums.KSTitStage</i>), 110		
	<i>attribute</i>), 477		
point_to_point_port_add()			<i>PORT0</i> (<i>msl.equipment.resources.picotech.picoscope.enums.PS2000 attribute</i>), 241
	<i>msl.equipment.resources.nkt.nktpdll.NKT method</i>), 179		
			<i>attribute</i>), 259
point_to_point_port_del()			<i>PORT1</i> (<i>msl.equipment.resources.picotech.picoscope.enums.PS2000 attribute</i>), 241

PORT1 (<i>msl.equipment.resources.picotech.picoscope.enums.PS3000ADigitalPort</i> attribute), 259	<i>nkt.nktpdll.NKT.PortStatusType</i> attribute), 172
PORT2 (<i>msl.equipment.resources.picotech.picoscope.PortStatusType</i> attribute), 241	<i>nkt.nktpdll.NKT.PortStatusType</i> (<i>msl.equipment.resources.nkt.nktpdll.PortStatusTypes</i>)
PORT2 (<i>msl.equipment.resources.picotech.picoscope.enums.PS2000DigitalPort</i> attribute), 259	<i>PortStatusCallback</i> (in module <i>msl.equipment.resources.nkt.nktpdll.NKT.PortStatusTypes</i>)
PORT3 (<i>msl.equipment.resources.picotech.picoscope.enums.PS2000DigitalPort</i> attribute), 241	162
PORT3 (<i>msl.equipment.resources.picotech.picoscope.PortStatusCallback</i> attribute), 259	<i>msl.equipment.resources.nkt.nktpdll.NKT</i> (<i>msl.equipment.resources.nkt.nktpdll.NKT.PortStatusType</i>)
PortClosed (<i>msl.equipment.resources.nkt.nktpdll.NKT.PortStatusType</i> attribute), 172	<i>PortStatusTypes</i> (class in <i>msl.equipment.resources.nkt.nktpdll.PortStatusTypes</i> & <i>msl.equipment.resources.nkt.nktpdll</i>), 168
PortClosed (<i>msl.equipment.resources.nkt.nktpdll.PortStatusType</i> attribute), 168	167
PortClosing (<i>msl.equipment.resources.nkt.nktpdll.NKT_PortStatusUnknown</i> attribute), 172	(<i>msl.equipment.resources.nkt.nktpdll.NKT.PortStatusType</i>)
PortClosing (<i>msl.equipment.resources.nkt.nktpdll.PortStatusType</i> attribute), 168	<i>PortStatusUnknown</i>
PortOpened (<i>msl.equipment.resources.nkt.nktpdll.NKT.PortStatusType</i> attribute), 172	<i>msl.equipment.resources.nkt.nktpdll.PortStatusTypes</i> attribute), 167
PortOpened (<i>msl.equipment.resources.nkt.nktpdll.PortStatusType</i> attribute), 167	(<i>msl.equipment.resources.thorlabs.kinesis.enums.P</i> attribute), 449
PortOpenFail (<i>msl.equipment.resources.nkt.nktpdll.NKT_PortStatusType</i> attribute), 172	<i>equipment.resources.thorlabs.kinesis.structs.N</i> attribute), 590
PortOpenFail (<i>msl.equipment.resources.nkt.nktpdll.PositionType</i> attribute), 168	<i>PositionType</i> (<i>msl.equipment.resources.thorlabs.kinesis.enums.TIM_E</i> attribute), 475
PortOpening (<i>msl.equipment.resources.nkt.nktpdll.NKT_PortStatusType</i> attribute), 172	<i>msl.equipment.resources.thorlabs.kinesis.enums.FF_F</i> attribute), 450
PortOpening (<i>msl.equipment.resources.nkt.nktpdll.PositionType</i> attribute), 167	<i>msl.equipment.resources.thorlabs.kinesis.enums.FF_F</i> attribute), 450
PortReady (<i>msl.equipment.resources.nkt.nktpdll.NKT_PortStatusType</i> attribute), 172	<i>msl.equipment.resources.thorlabs.kinesis.structs.MOT_B</i> attribute), 574
PortReady (<i>msl.equipment.resources.nkt.nktpdll.PortStatusType</i> attribute), 168	<i>POSITIVE_RUNT</i> (<i>msl.equipment.resources.picotech.picoscope.enums</i> attribute), 249
PortScanDeviceFound	
(<i>msl.equipment.resources.nkt.nktpdll.NKT_POSITIVE_R</i> attribute), 172	<i>POSITIVE_RUNT</i> (<i>msl.equipment.resources.picotech.picoscope.enums</i> attribute), 267
PortScanDeviceFound	
(<i>msl.equipment.resources.nkt.nktpdll.PortStatusType</i> attribute), 168	<i>POSITIVE_RUNT</i> (<i>msl.equipment.resources.picotech.picoscope.enums</i> attribute), 290
PortScanEnded (<i>msl.equipment.resources.nkt.nktpdll.NKT_PortStatusType</i> attribute), 172	<i>POSITIVE_RUNT</i> (<i>msl.equipment.resources.picotech.picoscope.enums</i> attribute), 328
PortScanEnded (<i>msl.equipment.resources.nkt.nktpdll.PortStatusType</i> attribute), 168	<i>POSITIVE_RUNT</i> (<i>msl.equipment.resources.picotech.picoscope.enums</i> attribute), 311
PortScanProgress	
(<i>msl.equipment.resources.nkt.nktpdll.NKT_PortStatusType</i> attribute), 172	<i>POSITIVE_RUNT</i> (<i>msl.equipment.resources.picotech.picoscope.enums</i> attribute), 449
PortScanProgress	
(<i>msl.equipment.resources.nkt.nktpdll.PortStatusType</i> attribute), 168	<i>postCycleDelay</i> (<i>msl.equipment.resources.nkt.nktpdll.PortStatusType</i> (<i>msl.equipment.resources.thorlabs.kinesis.structs.PZ_LUT</i> attribute), 580
PortScanStarted	
	<i>potentiometerStepParameters</i>

(<i>msl.equipment.resources.thorlabs.kinesis.structs.PRBS</i> . MAX_EQUIPMENT_TIMESLOTS <i>picotech.picoscope.enums.PS6000EAttribute</i>),	582	<i>attribute</i>),	319
power_off_time()		PRBS_MAX_FREQUENCY	
(<i>msl.equipment.resources.greisinger.gmh3000.GMH3000.equipment.resources.picotech.picoscope.ps2000a.PicotechMethod</i>),	149	<i>attribute</i>),	345
PPC_DerivFilterState	(class in PRBS_MAX_FREQUENCY)	<i>msl.equipment.resources.thorlabs.kinesis.enums</i>),	(<i>msl.equipment.resources.picotech.picoscope.ps3000a.PicotechAttribute</i>),
	447		348
PPC_DisplayIntensity	(class in PRBS_MAX_FREQUENCY)	<i>msl.equipment.resources.thorlabs.kinesis.enums</i>),	(<i>msl.equipment.resources.picotech.picoscope.ps6000.PicotechAttribute</i>),
	449		357
PPC_IoControlMode	(class in PRBS_MIN_FREQUENCY)	<i>msl.equipment.resources.thorlabs.kinesis.enums</i>),	(<i>msl.equipment.resources.picotech.picoscope.ps2000a.PicotechAttribute</i>),
	448		345
PPC_IoFeedbackSourceDefinition	(class in PRBS_MIN_FREQUENCY)	<i>msl.equipment.resources.thorlabs.kinesis.enums</i>),	(<i>msl.equipment.resources.picotech.picoscope.ps3000a.PicotechAttribute</i>),
	449		348
PPC_IoOutputBandwidth	(class in pre_trigger(msl.equipment.resources.picotech.picoscope.picoscopes))	<i>msl.equipment.resources.thorlabs.kinesis.enums</i>),	<i>property</i>),
	449		326
			preCycleDelay(msl.equipment.resources.thorlabs.kinesis.structs.KM)
PPC_IoOutputMode	(class in attribute),	580	
	<i>msl.equipment.resources.thorlabs.kinesis.enums</i>		prepare_measure()
	448		(<i>msl.equipment.resources.avantes.avaspec.AvantesMethod</i>),
PPC_IoSettings	(class in method),	123	
	<i>msl.equipment.resources.thorlabs.kinesis.structs.PresetPos1</i>		(<i>msl.equipment.resources.thorlabs.kinesis.structs.KMAttribute</i>),
	580		583
PPC_NotchFilterChannel	(class in PresetPos1(msl.equipment.resources.thorlabs.kinesis.structs.KP2))	<i>msl.equipment.resources.thorlabs.kinesis.enums</i>),	<i>attribute</i>),
	447		588
			PresetPos2(msl.equipment.resources.thorlabs.kinesis.structs.KM)
PPC_NotchFilterState	(class in attribute),	583	
	<i>msl.equipment.resources.thorlabs.kinesis.enums</i>		PresetPos2(msl.equipment.resources.thorlabs.kinesis.structs.KP2)
	447		<i>attribute</i>),
PPC_NotchParams	(class in PRESSURE_SENSOR_50BAR)	<i>msl.equipment.resources.thorlabs.kinesis.structs</i>),	(<i>msl.equipment.resources.picotech.picoscope.enums.PS4000AAttribute</i>),
	580		272
PPC_PIDConsts	(class in PRESSURE_SENSOR_5BAR)	<i>msl.equipment.resources.thorlabs.kinesis.structs</i>),	(<i>msl.equipment.resources.picotech.picoscope.enums.PS4000AAttribute</i>),
	580		272
PR4000B	(class in primary_address)	<i>msl.equipment.resources.mks_instruments.pr4000b</i>),	(<i>msl.equipment.connection_gpib.ConnectionGPIBProperty</i>),
	151		27
PRBS (<i>msl.equipment.resources.picotech.picoscope.enums.PS3000AEExtraOperations</i>)	(class in attribute),	246	
			<i>msl.equipment.resources.princeton_instruments.arc_instrument</i>
PRBS (<i>msl.equipment.resources.picotech.picoscope.enums.PS3000AEExtraOperations</i>)	(class in attribute),	264	
			PrincetonInstrumentsError , 56
PRBS (<i>msl.equipment.resources.picotech.picoscope.enums.PS4000AOperationTypes</i>)	(in module print_PSA0A_DefSignatures())		
			<i>msl.equipment.resources.picotech.picoscope.helper</i>),
PRBS (<i>msl.equipment.resources.picotech.picoscope.enums.PS4000AOperationTypes</i>)	(in module print_common_functions())		
			<i>msl.equipment.resources.picotech.picoscope.helper</i>),
PRBS (<i>msl.equipment.resources.picotech.picoscope.enums.PS5000AEquipOpers</i>)	(in module print_EquipOpers())		
			<i>msl.equipment.resources.picotech.picoscope.helper</i>),
			324

`print_define_statements()` (in module `attribute`), 245
`msl.equipment.resources.picotech.picoscope.PS`(~~msl.equipment.resources.picotech.picoscope.enums.PS2000Tim~~
`attribute), 234`
`PROBES`(~~msl.equipment.resources.picotech.picoscope.PS~~(~~msl.equipment.resources.picotech.picoscope.enums.PS3000ATi~~
`attribute), 273`
`attribute), 262`
`PROC_UNAVAIL`(`msl.equipment.vxi11.AcceptStatus` PS (~~msl.equipment.resources.picotech.picoscope.enums.PS3000Tim~~
`attribute), 602`
`attribute), 253`
`ProcessControlType` (class in `PS`(~~msl.equipment.resources.picotech.picoscope.enums.PS4000ATi~~
`msl.equipment.resources.avantes.avaspec),`
`attribute), 284`
`104`
`PS`(~~msl.equipment.resources.picotech.picoscope.enums.PS4000Tim~~
`attribute), 273`
`PROG_MISMATCH`(`msl.equipment.vxi11.AcceptStatus` PS(~~msl.equipment.resources.picotech.picoscope.enums.PS5000ATi~~
`attribute), 602`
`attribute), 273`
`PROG_UNAVAIL`(`msl.equipment.vxi11.AcceptStatus` PS(~~msl.equipment.resources.picotech.picoscope.enums.PS5000Tim~~
`attribute), 602`
`attribute), 308`
`PROLOGIX`(`msl.equipment.constants.Interface` at-
`tribute), 50`
`tribute), 298`
`prologue` (`msl.equipment.hislip.Message` at-
`tribute), 59`
`tribute), 317`
`PS2000A_ThresholdMode` (class in
`properties`(`msl.equipment.record_types.ConnectionRecord` ~~msl.equipment.resources.picotech.picoscope.enums~~,
`attribute), 86`
`attribute), 248`
`proportionalGain` `ps2000aBlockReady` (in module
`(msl.equipment.resources.thorlabs.kinesis.structs.MOts_Requisites_CrossTalk_Coupling_Picoscope.callbacks),`
`attribute), 575`
`229`
`proportionalGain` `PS2000AChannel` (class in
`(msl.equipment.resources.thorlabs.kinesis.structs.MOts_Requisites_Position_Error_Picoscope.enums),`
`attribute), 574`
`240`
`proportionalGain` `PS2000AChannelBufferIndex` (class in
`(msl.equipment.resources.thorlabs.kinesis.structs.MOts_Requisites_Position_Error_Picoscope.enums),`
`attribute), 577`
`239`
`proportionalGain` `PS2000AChannelInfo` (class in
`(msl.equipment.resources.thorlabs.kinesis.structs.QDn_EquipParameter_Picoscope.enums),`
`attribute), 581`
`244`
`proportionalGain` `PS2000ACoupling` (class in
`(msl.equipment.resources.thorlabs.kinesis.structs.QDn_EquipParameter_Picoscope.enums),`
`attribute), 589`
`244`
`proportionalGain` `ps2000aDataReady` (in module
`(msl.equipment.resources.thorlabs.kinesis.structs.QDn_EquipParameter_Picoscope.callbacks),`
`attribute), 589`
`230`
`proportionalGain` `PS2000ADigitalChannel` (class in
`(msl.equipment.resources.thorlabs.kinesis.structs.TC_Ns_Freq_Punkto_Punkt_Gangstyp_Picoscope.structs),`
`attribute), 593`
`241`
`proportionalTerm` `PS2000ADigitalChannelDirections` (class in
`(msl.equipment.resources.thorlabs.kinesis.structs.KN_As_Freq_Punkto_Punkt_Gangstyp_Picoscope.structs),`
`attribute), 587`
`360`
`proportionalTerm` `PS2000ADigitalDirection` (class in
`(msl.equipment.resources.thorlabs.kinesis.structs.PZn_Freq_Punkto_Punkt_Constant_Picoscope.enums),`
`attribute), 579`
`249`
`protocol_version` `PS2000AEtsMode` (class in
`(msl.equipment.hislip.InitializeResponse` ~~msl.equipment.resources.picotech.picoscope.enums~~,
`property), 62`
`244`
`PS`(~~msl.equipment.resources.picotech.picoscope.enums.PS2000AEtsOperations~~
`PS2000AEtsOperations` (class in

msl.equipment.resources.picotech.picoscope.enums),msl.equipment.resources.picotech.picoscope.enums),
246 245

PS2000AHoldOffType (class in **PS2000ButtonState** (class in
msl.equipment.resources.picotech.picoscope.enums),msl.equipment.resources.picotech.picoscope.enums),
251 236

PS2000AIndexMode (class in **PS2000Channel** (class in
msl.equipment.resources.picotech.picoscope.enums),msl.equipment.resources.picotech.picoscope.enums),
247 233

PS2000APulseWidthType (class in **PS2000DigitalPort** (class in
msl.equipment.resources.picotech.picoscope.enums),msl.equipment.resources.picotech.picoscope.enums),
250 241

PS2000APwqConditions (class in **PS2000Error** (class in
msl.equipment.resources.picotech.picoscope structs),msl.equipment.resources.picotech.errors),
359 228

PS2000ARange (class in **PS2000EtsMode** (class in
msl.equipment.resources.picotech.picoscope.enums),msl.equipment.resources.picotech.picoscope.enums),
243 236

PS2000ARatioMode (class in **PS2000Info** (class in
msl.equipment.resources.picotech.picoscope.enums),msl.equipment.resources.picotech.picoscope.enums),
250 234

PS2000ASigGenTrigSource (class in **PS2000OpenProgress** (class in
msl.equipment.resources.picotech.picoscope.enums),msl.equipment.resources.picotech.picoscope.enums),
247 235

PS2000ASigGenTrigType (class in **PS2000PulseWidthType** (class in
msl.equipment.resources.picotech.picoscope.enums),msl.equipment.resources.picotech.picoscope.enums),
246 239

ps2000aStreamingReady (in module **PS2000PwqConditions** (class in
msl.equipment.resources.picotech.picoscope.callbacks),msl.equipment.resources.picotech.picoscope.structs),
229 358

PS2000ASweepType (class in **PS2000Range** (class in
msl.equipment.resources.picotech.picoscope.enums),msl.equipment.resources.picotech.picoscope.enums),
245 233

PS2000AThresholdDirection (class in **PS2000SweepType** (class in
msl.equipment.resources.picotech.picoscope.enums),msl.equipment.resources.picotech.picoscope.enums),
248 237

PS2000ATimeUnits (class in **PS2000ThresholdDirection** (class in
msl.equipment.resources.picotech.picoscope.enums),msl.equipment.resources.picotech.picoscope.enums),
245 238

PS2000ATriggerChannelProperties (class in **PS2000ThresholdMode** (class in
msl.equipment.resources.picotech.picoscope.structs),msl.equipment.resources.picotech.picoscope.enums),
360 238

PS2000ATriggerConditions (class in **PS2000TimeUnits** (class in
msl.equipment.resources.picotech.picoscope.structs),msl.equipment.resources.picotech.picoscope.enums),
359 234

PS2000ATriggerOperand (class in **PS2000TriggerChannelProperties** (class in
msl.equipment.resources.picotech.picoscope.enums),msl.equipment.resources.picotech.picoscope.structs),
241 358

PS2000ATriggerState (class in **PS2000TriggerConditions** (class in
msl.equipment.resources.picotech.picoscope.enums),msl.equipment.resources.picotech.picoscope.structs),
250 358

PS2000AWaveType (class in **PS2000TriggerDirection** (class in

<i>msl.equipment.resources.picotech.picoscope.enums)</i> , 235	<i>(msl.equipment.resources.picotech.picoscope.ps2000a.Pic-</i>
PS2000TriggerState <i>(class in PS2207_MAX_ETS_CYCLES</i>	<i>msl.equipment.resources.picotech.picoscope.enums),(msl.equipment.resources.picotech.picoscope.ps2000a.Pic-</i>
<i>attribute), 344</i>	<i>attribute), 344</i>
PS2000WaveType <i>(class in PS2207_MAX_INTERLEAVE</i>	<i>msl.equipment.resources.picotech.picoscope.enums),(msl.equipment.resources.picotech.picoscope.ps2000a.Pic-</i>
<i>attribute), 237</i>	<i>attribute), 344</i>
PS2104_MAX_ETS_CYCLES	PS2208_MAX_ETS_CYCLES
<i>(msl.equipment.resources.picotech.picoscope.ps2000(Pic-</i>	<i>attribute), 344</i>
<i>attribute), 342</i>	<i>attribute), 344</i>
PS2104_MAX_ETS_INTERLEAVE	PS2208_MAX_INTERLEAVE
<i>(msl.equipment.resources.picotech.picoscope.ps2000(Pic-</i>	<i>attribute), 344</i>
<i>attribute), 342</i>	<i>attribute), 344</i>
PS2104_MAX_TIMEBASE	PS3000_CALLBACK_FUNC <i>(in module</i>
<i>(msl.equipment.resources.picotech.picoscope.ps2000(Pic-</i>	<i>msl.equipment.resources.picotech.picoscope.callbacks),</i>
<i>attribute), 342</i>	<i>230</i>
PS2105_MAX_ETS_CYCLES	PS3000A_ThresholdMode <i>(class in</i>
<i>(msl.equipment.resources.picotech.picoscope.ps2000(Pic-</i>	<i>msl.equipment.resources.picotech.picoscope.enums),</i>
<i>attribute), 342</i>	<i>265</i>
PS2105_MAX_ETS_INTERLEAVE	PS3000ABandwidthLimiter <i>(class in</i>
<i>(msl.equipment.resources.picotech.picoscope.ps2000(Pic-</i>	<i>msl.equipment.resources.picotech.picoscope.enums),</i>
<i>attribute), 342</i>	<i>257</i>
PS2105_MAX_TIMEBASE	ps3000aBlockReady <i>(in module</i>
<i>(msl.equipment.resources.picotech.picoscope.ps2000(Pic-</i>	<i>msl.equipment.resources.picotech.picoscope.callbacks),</i>
<i>attribute), 342</i>	<i>229</i>
PS2200_MAX_TIMEBASE	PS3000AChannel <i>(class in</i>
<i>(msl.equipment.resources.picotech.picoscope.ps2000(Pic-</i>	<i>msl.equipment.resources.picotech.picoscope.enums),</i>
<i>attribute), 342</i>	<i>258</i>
PS2203_MAX_ETS_CYCLES	PS3000AChannelBufferIndex <i>(class in</i>
<i>(msl.equipment.resources.picotech.picoscope.ps2000(Pic-</i>	<i>msl.equipment.resources.picotech.picoscope.enums),</i>
<i>attribute), 342</i>	<i>257</i>
PS2203_MAX_ETS_INTERLEAVE	PS3000AChannelInfo <i>(class in</i>
<i>(msl.equipment.resources.picotech.picoscope.ps2000(Pic-</i>	<i>msl.equipment.resources.picotech.picoscope.enums),</i>
<i>attribute), 343</i>	<i>261</i>
PS2204_MAX_ETS_CYCLES	PS3000ACoupling <i>(class in</i>
<i>(msl.equipment.resources.picotech.picoscope.ps2000(Pic-</i>	<i>msl.equipment.resources.picotech.picoscope.enums),</i>
<i>attribute), 343</i>	<i>261</i>
PS2204_MAX_ETS_INTERLEAVE	ps3000aDataReady <i>(in module</i>
<i>(msl.equipment.resources.picotech.picoscope.ps2000(Pic-</i>	<i>msl.equipment.resources.picotech.picoscope.callbacks),</i>
<i>attribute), 343</i>	<i>230</i>
PS2205_MAX_ETS_CYCLES	PS3000ADigitalChannel <i>(class in</i>
<i>(msl.equipment.resources.picotech.picoscope.ps2000(Pic-</i>	<i>msl.equipment.resources.picotech.picoscope.enums),</i>
<i>attribute), 343</i>	<i>259</i>
PS2205_MAX_ETS_INTERLEAVE	PS3000ADigitalChannelDirections <i>(class in</i>
<i>(msl.equipment.resources.picotech.picoscope.ps2000(Pic-</i>	<i>msl.equipment.resources.picotech.picoscope.structs),</i>
<i>attribute), 343</i>	<i>363</i>
PS2206_MAX_ETS_CYCLES	PS3000ADigitalDirection <i>(class in</i>
<i>(msl.equipment.resources.picotech.picoscope.ps2000(Pic-</i>	<i>msl.equipment.resources.picotech.picoscope.enums),</i>
<i>attribute), 344</i>	<i>267</i>
PS2206_MAX_INTERLEAVE	PS3000ADigitalPort <i>(class in</i>

	<i>msl.equipment.resources.picotech.picoscope.enums), msl.equipment.resources.picotech.picoscope structs),</i>	
	259	361
PS3000AEtsMode	(class in PS3000ATriggerConditionsV2 (class in <i>msl.equipment.resources.picotech.picoscope.enums), msl.equipment.resources.picotech.picoscope structs),</i>	
	262	362
PS3000AExtraOperations	(class in PS3000ATriggerInfo (class in <i>msl.equipment.resources.picotech.picoscope.enums), msl.equipment.resources.picotech.picoscope structs),</i>	
	264	364
PS3000AHoldOffType	(class in PS3000ATriggerState (class in <i>msl.equipment.resources.picotech.picoscope.enums), msl.equipment.resources.picotech.picoscope enums),</i>	
	268	267
PS3000AIndexMode	(class in PS3000AWaveType (class in <i>msl.equipment.resources.picotech.picoscope.enums), msl.equipment.resources.picotech.picoscope enums),</i>	
	265	263
PS3000APulseWidthType	(class in PS3000Channel (class in <i>msl.equipment.resources.picotech.picoscope.enums), msl.equipment.resources.picotech.picoscope enums),</i>	
	268	251
PS3000APwqConditions	(class in PS3000Error (class in <i>msl.equipment.resources.picotech.picoscope structs), msl.equipment.resources.picotech.errors),</i>	
	362	228
PS3000APwqConditionsV2	(class in PS3000EtsMode (class in <i>msl.equipment.resources.picotech.picoscope structs), msl.equipment.resources.picotech.picoscope enums),</i>	
	363	255
PS3000ARange	(class in PS3000Info (class in <i>msl.equipment.resources.picotech.picoscope.enums), msl.equipment.resources.picotech.picoscope enums),</i>	
	260	253
PS3000ARatioMode	(class in PS3000OpenProgress (class in <i>msl.equipment.resources.picotech.picoscope.enums), msl.equipment.resources.picotech.picoscope enums),</i>	
	268	254
PS3000ASigGenTrigSource	(class in PS3000PulseWidthType (class in <i>msl.equipment.resources.picotech.picoscope.enums), msl.equipment.resources.picotech.picoscope enums),</i>	
	264	257
PS3000ASigGenTrigType	(class in PS3000PwqConditions (class in <i>msl.equipment.resources.picotech.picoscope.enums), msl.equipment.resources.picotech.picoscope structs),</i>	
	264	361
ps3000aStreamingReady	(in module PS3000Range (class in <i>msl.equipment.resources.picotech.picoscope.callbacks), msl.equipment.resources.picotech.picoscope enums),</i>	
	229	252
PS3000ASweepType	(class in PS3000ThresholdDirection (class in <i>msl.equipment.resources.picotech.picoscope.enums), msl.equipment.resources.picotech.picoscope enums),</i>	
	263	255
PS3000AThresholdDirection	(class in PS3000ThresholdMode (class in <i>msl.equipment.resources.picotech.picoscope.enums), msl.equipment.resources.picotech.picoscope enums),</i>	
	266	256
PS3000ATimeUnits	(class in PS3000TimeUnits (class in <i>msl.equipment.resources.picotech.picoscope.enums), msl.equipment.resources.picotech.picoscope enums),</i>	
	262	253
PS3000ATriggerChannelProperties	(class in PS3000TriggerChannelProperties (class in <i>msl.equipment.resources.picotech.picoscope structs), msl.equipment.resources.picotech.picoscope structs),</i>	
	363	360
PS3000ATriggerConditions	(class in PS3000TriggerConditions (class in	

	<i>msl.equipment.resources.picotech.picoscope structs),(msl.equipment.resources.picotech.picoscope.ps3000.Picoattribute), 346</i>
PS3000TriggerDirection	(class in PS3206_MAX_ETS_INTERLEAVE <i>msl.equipment.resources.picotech.picoscope enums),(msl.equipment.resources.picotech.picoscope.ps3000.Picoattribute), 346</i>
PS3000TriggerState	(class in PS3206_MAX_TIMEBASE <i>msl.equipment.resources.picotech.picoscope enums),(msl.equipment.resources.picotech.picoscope.ps3000.Picoattribute), 345</i>
PS3000WaveTypes	(class in PS3206A_MAX_ETS_CYCLES <i>msl.equipment.resources.picotech.picoscope enums),(msl.equipment.resources.picotech.picoscope.ps3000a.Picoattribute), 347</i>
PS300_MAX_ETS_SAMPLES	PS3206A_MAX_INTERLEAVE <i>(msl.equipment.resources.picotech.picoscope ps3000(PicodEquipment3000resources.picotech.picoscope.ps3000a.Picoattribute), 347</i>
PS3204_MAX_ETS_CYCLES	PS3206B_MAX_SIG_GEN_BUFFER_SIZE <i>(msl.equipment.resources.picotech.picoscope ps3000(PicodEquipment3000resources.picotech.picoscope.ps3000a.Picoattribute), 348</i>
PS3204_MAX_ETS_INTERLEAVE	PS3206MSO_MAX_INTERLEAVE <i>(msl.equipment.resources.picotech.picoscope ps3000(PicodEquipment3000resources.picotech.picoscope.ps3000a.Picoattribute), 347</i>
PS3204_MAX_TIMEBASE	PS3207A_MAX_ETS_CYCLES <i>(msl.equipment.resources.picotech.picoscope ps3000(PicodEquipment3000resources.picotech.picoscope.ps3000a.Picoattribute), 347</i>
PS3204A_MAX_ETS_CYCLES	PS3207A_MAX_INTERLEAVE <i>(msl.equipment.resources.picotech.picoscope ps3000(anBlockScope3000sources.picotech.picoscope.ps3000a.Picoattribute), 347</i>
PS3204A_MAX_INTERLEAVE	PS3207B_MAX_SIG_GEN_BUFFER_SIZE <i>(msl.equipment.resources.picotech.picoscope ps3000(anBlockScope3000sources.picotech.picoscope.ps3000a.Picoattribute), 347</i>
PS3204MSO_MAX_INTERLEAVE	PS3223_MAX_TIMEBASE <i>(msl.equipment.resources.picotech.picoscope ps3000(anBlockScope3000sources.picotech.picoscope.ps3000.Picoattribute), 346</i>
PS3205_MAX_ETS_CYCLES	PS3224_MAX_TIMEBASE <i>(msl.equipment.resources.picotech.picoscope ps3000(PicodEquipment3000resources.picotech.picoscope.ps3000.Picoattribute), 346</i>
PS3205_MAX_ETS_INTERLEAVE	PS3225_MAX_TIMEBASE <i>(msl.equipment.resources.picotech.picoscope ps3000(PicodEquipment3000resources.picotech.picoscope.ps3000.Picoattribute), 346</i>
PS3205_MAX_TIMEBASE	PS3226_MAX_TIMEBASE <i>(msl.equipment.resources.picotech.picoscope ps3000(PicodEquipment3000resources.picotech.picoscope.ps3000.Picoattribute), 346</i>
PS3205A_MAX_ETS_CYCLES	PS3423_MAX_TIMEBASE <i>(msl.equipment.resources.picotech.picoscope ps3000(anBlockScope3000sources.picotech.picoscope.ps3000.Picoattribute), 346</i>
PS3205A_MAX_INTERLEAVE	PS3424_MAX_TIMEBASE <i>(msl.equipment.resources.picotech.picoscope ps3000(anBlockScope3000sources.picotech.picoscope.ps3000.Picoattribute), 346</i>
PS3205MSO_MAX_INTERLEAVE	PS3425_MAX_TIMEBASE <i>(msl.equipment.resources.picotech.picoscope ps3000(anBlockScope3000sources.picotech.picoscope.ps3000.Picoattribute), 346</i>
PS3206_MAX_ETS_CYCLES	PS3426_MAX_TIMEBASE

(<i>msl.equipment.resources.picotech.picoscope.ps3000AEquipmentResources.picotech.picoscope.enums</i>), attribute), 346	288
PS4000ABandwidthLimiter (class in PS4000AMetaFormat (class in <i>msl.equipment.resources.picotech.picoscope.enums</i>), <i>msl.equipment.resources.picotech.picoscope.enums</i>), 280	287
ps4000aBlockReady (in module PS4000AMetaOperation (class in <i>msl.equipment.resources.picotech.picoscope.callbacks</i>), <i>msl.equipment.resources.picotech.picoscope.enums</i>), 229	286
PS4000AChannel (class in PS4000AMetaType (class in <i>msl.equipment.resources.picotech.picoscope.enums</i>), <i>msl.equipment.resources.picotech.picoscope.enums</i>), 281	286
PS4000AChannelBufferIndex (class in PS4000APicoStringValue (class in <i>msl.equipment.resources.picotech.picoscope.enums</i>), <i>msl.equipment.resources.picotech.picoscope.enums</i>), 282	292
PS4000AChannelInfo (class in PS4000APulseWidthType (class in <i>msl.equipment.resources.picotech.picoscope.enums</i>), <i>msl.equipment.resources.picotech.picoscope.enums</i>), 292	292
PS4000AChannelLed (class in PS4000ARange (class in <i>msl.equipment.resources.picotech.picoscope.enums</i>), <i>msl.equipment.resources.picotech.picoscope.enums</i>), 286	282
PS4000AChannelLedSetting (class in PS4000ARatioMode (class in <i>msl.equipment.resources.picotech.picoscope structs</i>), <i>msl.equipment.resources.picotech.picoscope.enums</i>), 365	291
PS4000ACondition (class in PS4000AResistanceRange (class in <i>msl.equipment.resources.picotech.picoscope structs</i>), <i>msl.equipment.resources.picotech.picoscope.enums</i>), 366	283
PS4000AConditionsInfo (class in PS4000ASensorState (class in <i>msl.equipment.resources.picotech.picoscope.enums</i>), <i>msl.equipment.resources.picotech.picoscope.enums</i>), 291	290
PS4000AConnectDetect (class in PS4000ASigGenTrigSource (class in <i>msl.equipment.resources.picotech.picoscope structs</i>), <i>msl.equipment.resources.picotech.picoscope.enums</i>), 366	288
PS4000ACoupling (class in PS4000ASigGenTrigType (class in <i>msl.equipment.resources.picotech.picoscope.enums</i>), <i>msl.equipment.resources.picotech.picoscope.enums</i>), 280	287
ps4000aDataReady (in module ps4000aStreamingReady (in module <i>msl.equipment.resources.picotech.picoscope.callbacks</i>), <i>msl.equipment.resources.picotech.picoscope.callbacks</i>), 230	229
PS4000ADirection (class in PS4000ASweepType (class in <i>msl.equipment.resources.picotech.picoscope structs</i>), <i>msl.equipment.resources.picotech.picoscope.enums</i>), 365	285
PS4000AEtsMode (class in PS4000AThresholdDirection (class in <i>msl.equipment.resources.picotech.picoscope.enums</i>), <i>msl.equipment.resources.picotech.picoscope.enums</i>), 284	289
PS4000AExtraOperations (class in PS4000AThresholdMode (class in <i>msl.equipment.resources.picotech.picoscope.enums</i>), <i>msl.equipment.resources.picotech.picoscope.enums</i>), 280	288
PS4000AFrequencyCounterRange (class in PS4000ATimeUnits (class in <i>msl.equipment.resources.picotech.picoscope.enums</i>), <i>msl.equipment.resources.picotech.picoscope.enums</i>), 290	284
PS4000AIndexMode (class in PS4000ATriggerChannelProperties (class in	

<i>msl.equipment.resources.picotech.picoscope structs</i>), <i>msl.equipment.resources.picotech.picoscope enums</i>), 366	278
PS4000ATriggerState (class in PS4000SigGenTrigSource (class in <i>msl.equipment.resources.picotech.picoscope enums</i>), <i>msl.equipment.resources.picotech.picoscope enums</i>), 290	276
PS4000AWaveType (class in PS4000SigGenTrigType (class in <i>msl.equipment.resources.picotech.picoscope enums</i>), <i>msl.equipment.resources.picotech.picoscope enums</i>), 285	275
ps4000BlockReady (in module ps4000StreamingReady (in module <i>msl.equipment.resources.picotech.picoscope callbacks</i>), <i>msl.equipment.resources.picotech.picoscope callbacks</i>), 229	229
PS4000Channel (class in PS4000SweepType (class in <i>msl.equipment.resources.picotech.picoscope enums</i>), <i>msl.equipment.resources.picotech.picoscope enums</i>), 269	274
PS4000ChannelBufferIndex (class in PS4000ThresholdDirection (class in <i>msl.equipment.resources.picotech.picoscope enums</i>), <i>msl.equipment.resources.picotech.picoscope enums</i>), 269	277
PS4000ChannelInfo (class in PS4000ThresholdMode (class in <i>msl.equipment.resources.picotech.picoscope enums</i>), <i>msl.equipment.resources.picotech.picoscope enums</i>), 272	276
ps4000DataReady (in module PS4000TimeUnits (class in <i>msl.equipment.resources.picotech.picoscope callbacks</i>), <i>msl.equipment.resources.picotech.picoscope enums</i>), 230	273
PS4000EtsMode (class in PS4000TriggerChannelProperties (class in <i>msl.equipment.resources.picotech.picoscope enums</i>), <i>msl.equipment.resources.picotech.picoscope structs</i>), 273	365
PS4000FrequencyCounterRange (class in PS4000TriggerConditions (class in <i>msl.equipment.resources.picotech.picoscope enums</i>), <i>msl.equipment.resources.picotech.picoscope structs</i>), 279	364
PS4000IndexMode (class in PS4000TriggerState (class in <i>msl.equipment.resources.picotech.picoscope enums</i>), <i>msl.equipment.resources.picotech.picoscope enums</i>), 276	278
PS4000OperationTypes (class in PS4000WaveType (class in <i>msl.equipment.resources.picotech.picoscope enums</i>), <i>msl.equipment.resources.picotech.picoscope enums</i>), 274	274
PS4000Probe (class in PS4262_MAX_VALUE <i>msl.equipment.resources.picotech.picoscope enums</i>),(<i>msl.equipment.resources.picotech.picoscope ps4000.Pico attribute</i>), 272	349
PS4000Ps4000HoldOffType (class in PS4262_MAX_WAVEFORM_BUFFER_SIZE <i>msl.equipment.resources.picotech.picoscope enums</i>),(<i>msl.equipment.resources.picotech.picoscope ps4000.Pico attribute</i>), 279	350
PS4000PulseWidthType (class in PS4262_MIN_DWELL_COUNT <i>msl.equipment.resources.picotech.picoscope enums</i>),(<i>msl.equipment.resources.picotech.picoscope ps4000.Pico attribute</i>), 278	350
PS4000PwqConditions (class in PS4262_MIN_VALUE <i>msl.equipment.resources.picotech.picoscope structs</i>),(<i>msl.equipment.resources.picotech.picoscope ps4000.Pico attribute</i>), 364	349
PS4000Range (class in PS4XXX_MAX_ETS_CYCLES <i>msl.equipment.resources.picotech.picoscope enums</i>),(<i>msl.equipment.resources.picotech.picoscope ps4000.Pico attribute</i>), 270	349
PS4000RatioMode (class in PS4XXX_MAX_INTERLEAVE	

(<i>msl.equipment.resources.picotech.picoscope.ps4000a.StreamingReady</i> , <i>msl.equipment.picoscope.enums</i>), attribute), 349	309
PS5000ABandwidthLimiter (class in <i>ps5000a.StreamingReady</i> (in module <i>msl.equipment.resources.picotech.picoscope.enums</i>), <i>msl.equipment.resources.picotech.picoscope.callbacks</i>), 304	230
ps5000aBlockReady (in module <i>PS5000ASweepType</i> (class in <i>msl.equipment.resources.picotech.picoscope.callbacks</i> , <i>msl.equipment.resources.picotech.picoscope.enums</i>), 229	308
PS5000AChannel (class in <i>PS5000AThresholdDirection</i> (class in <i>msl.equipment.resources.picotech.picoscope.enums</i>), <i>msl.equipment.resources.picotech.picoscope.enums</i>), 305	310
PS5000AChannelBufferIndex (class in <i>PS5000AThresholdMode</i> (class in <i>msl.equipment.resources.picotech.picoscope.enums</i>), <i>msl.equipment.resources.picotech.picoscope.enums</i>), 306	310
PS5000AChannelInfo (class in <i>PS5000ATimeUnits</i> (class in <i>msl.equipment.resources.picotech.picoscope.enums</i>), <i>msl.equipment.resources.picotech.picoscope.enums</i>), 313	307
PS5000ACoupling (class in <i>PS5000ATriggerChannelProperties</i> (class in <i>msl.equipment.resources.picotech.picoscope.enums</i>), <i>msl.equipment.resources.picotech.picoscope structs</i>), 305	369
ps5000aDataReady (in module <i>PS5000ATriggerConditions</i> (class in <i>msl.equipment.resources.picotech.picoscope.callbacks</i> , <i>msl.equipment.resources.picotech.picoscope structs</i>), 230	368
PS5000ADeviceResolution (class in <i>PS5000ATriggerInfo</i> (class in <i>msl.equipment.resources.picotech.picoscope.enums</i>), <i>msl.equipment.resources.picotech.picoscope structs</i>), 303	367
PS5000AEtsMode (class in <i>PS5000ATriggerState</i> (class in <i>msl.equipment.resources.picotech.picoscope.enums</i>), <i>msl.equipment.resources.picotech.picoscope enums</i>), 307	311
PS5000AExtraOperations (class in <i>PS5000ATriggerWithinPreTrigger</i> (class in <i>msl.equipment.resources.picotech.picoscope.enums</i>), <i>msl.equipment.resources.picotech.picoscope enums</i>), 304	312
PS5000AIndexMode (class in <i>PS5000AWaveType</i> (class in <i>msl.equipment.resources.picotech.picoscope.enums</i>), <i>msl.equipment.resources.picotech.picoscope enums</i>), 310	308
PS5000APulseWidthType (class in <i>ps5000a.BlockReady</i> (in module <i>msl.equipment.resources.picotech.picoscope.enums</i>), <i>msl.equipment.resources.picotech.picoscope callbacks</i>), 313	229
PS5000APwqConditions (class in <i>PS5000Channel</i> (class in <i>msl.equipment.resources.picotech.picoscope.structs</i>), <i>msl.equipment.resources.picotech.picoscope enums</i>), 368	296
PS5000ARange (class in <i>PS5000ChannelBufferIndex</i> (class in <i>msl.equipment.resources.picotech.picoscope.enums</i>), <i>msl.equipment.resources.picotech.picoscope enums</i>), 306	296
PS5000ARatioMode (class in <i>PS5000ChannelInfo</i> (class in <i>msl.equipment.resources.picotech.picoscope.enums</i>), <i>msl.equipment.resources.picotech.picoscope enums</i>), 312	303
PS5000ASigGenTrigSource (class in <i>ps5000aDataReady</i> (in module <i>msl.equipment.resources.picotech.picoscope.enums</i>), <i>msl.equipment.resources.picotech.picoscope callbacks</i>), 309	230
PS5000ASigGenTrigType (class in <i>PS5000EtsMode</i> (class in in	

<i>msl.equipment.resources.picotech.picoscope.enums),(msl.equipment.resources.picotech.picoscope.ps5000a.Pic</i>		
298	<i>attribute), 355</i>	
PS5000IndexMode	(class	<i>in PS5242A_MAX_ETS_INTERLEAVE</i>
	<i>msl.equipment.resources.picotech.picoscope.enums),(msl.equipment.resources.picotech.picoscope.ps5000a.Pic</i>	
300	<i>attribute), 355</i>	
PS5000PulseWidthType	(class	<i>in PS5243A_MAX_ETS_CYCLES</i>
	<i>msl.equipment.resources.picotech.picoscope.enums),(msl.equipment.resources.picotech.picoscope.ps5000a.Pic</i>	
303	<i>attribute), 355</i>	
PS5000PwqConditions	(class	<i>in PS5243A_MAX_ETS_INTERLEAVE</i>
	<i>msl.equipment.resources.picotech.picoscope structs),(msl.equipment.resources.picotech.picoscope.ps5000a.Pic</i>	
367	<i>attribute), 355</i>	
PS5000Range	(class	<i>in PS5244A_MAX_ETS_CYCLES</i>
	<i>msl.equipment.resources.picotech.picoscope.enums),(msl.equipment.resources.picotech.picoscope.ps5000a.Pic</i>	
297	<i>attribute), 355</i>	
PS5000RatioMode	(class	<i>in PS5244A_MAX_ETS_INTERLEAVE</i>
	<i>msl.equipment.resources.picotech.picoscope.enums),(msl.equipment.resources.picotech.picoscope.ps5000a.Pic</i>	
302	<i>attribute), 355</i>	
PS5000SigGenTrigSource	(class	<i>in PS5X42A_MAX_SIG_GEN_BUFFER_SIZE</i>
	<i>msl.equipment.resources.picotech.picoscope.enums),(msl.equipment.resources.picotech.picoscope.ps5000a.Pic</i>	
300	<i>attribute), 354</i>	
PS5000SigGenTrigType	(class	<i>in PS5X43A_MAX_SIG_GEN_BUFFER_SIZE</i>
	<i>msl.equipment.resources.picotech.picoscope.enums),(msl.equipment.resources.picotech.picoscope.ps5000a.Pic</i>	
300	<i>attribute), 354</i>	
ps5000StreamingReady	(in module	<i>PS5X44A_MAX_SIG_GEN_BUFFER_SIZE</i>
	<i>msl.equipment.resources.picotech.picoscope.callbacks),(msl.equipment.resources.picotech.picoscope.ps5000a.Pic</i>	
230	<i>attribute), 355</i>	
PS5000SweepType	(class	<i>in PS6000BandwidthLimiter</i> (class
	<i>msl.equipment.resources.picotech.picoscope.enums),msl.equipment.resources.picotech.picoscope.enums),</i>	<i>in</i>
299	<i>314</i>	
PS5000ThresholdDirection	(class	<i>in ps6000BlockReady</i> (in module
	<i>msl.equipment.resources.picotech.picoscope.enums),msl.equipment.resources.picotech.picoscope.callbacks),</i>	
301	<i>229</i>	
PS5000ThresholdMode	(class	<i>in PS6000Channel</i> (class
	<i>msl.equipment.resources.picotech.picoscope.enums),msl.equipment.resources.picotech.picoscope.enums),</i>	<i>in</i>
301	<i>314</i>	
PS5000TimeUnits	(class	<i>in PS6000ChannelBufferIndex</i> (class
	<i>msl.equipment.resources.picotech.picoscope.enums),msl.equipment.resources.picotech.picoscope.enums),</i>	<i>in</i>
298	<i>315</i>	
PS5000TriggerChannelProperties	(class	<i>in PS6000Coupling</i> (class
	<i>msl.equipment.resources.picotech.picoscope structs),msl.equipment.resources.picotech.picoscope.enums),</i>	<i>in</i>
367	<i>316</i>	
PS5000TriggerConditions	(class	<i>in ps6000DataReady</i> (in module
	<i>msl.equipment.resources.picotech.picoscope structs),msl.equipment.resources.picotech.picoscope.callbacks),</i>	
366	<i>230</i>	
PS5000TriggerState	(class	<i>in PS6000EtsMode</i> (class
	<i>msl.equipment.resources.picotech.picoscope.enums),msl.equipment.resources.picotech.picoscope.enums),</i>	<i>in</i>
302	<i>317</i>	
PS5000WaveType	(class	<i>in PS6000ExternalFrequency</i> (class
	<i>msl.equipment.resources.picotech.picoscope.enums),msl.equipment.resources.picotech.picoscope.enums),</i>	<i>in</i>
299	<i>313</i>	
PS5242A_MAX_ETS_CYCLES		PS6000ExtraOperations (class
		<i>in</i>

*msl.equipment.resources.picotech.picoscope.enums), (msl.equipment.resources.picotech.picoscope.ps6000.Pico
318
attribute), 356*

PS6000IndexMode (class in *PT100 (msl.equipment.resources.picotech.pt104.Pt104DataType
msl.equipment.resources.picotech.picoscope.enums), attribute), 371
320
*PT1000 (msl.equipment.resources.picotech.pt104.Pt104DataType**

PS6000PulseWidthType (class in *attribute), 371
msl.equipment.resources.picotech.picoscope.PT104s), (class in
322
*msl.equipment.resources.picotech.pt104),**

PS6000PwqConditions (class in *371
msl.equipment.resources.picotech.picoscope.PT104DataTypes (class in
370
*msl.equipment.resources.picotech.pt104),**

PS6000Range (class in *370
msl.equipment.resources.picotech.picoscope.PULSESWIDTH_ARRAY_OF_BLOCK_CONDITIONS
316
*(msl.equipment.resources.picotech.picoscope.enums.PS40**

PS6000RatioMode (class in *attribute), 294
msl.equipment.resources.picotech.picoscope.PULSESWIDTH_CONDITIONS
322
*(msl.equipment.resources.picotech.picoscope.enums.PS40**

PS6000SigGenTrigSource (class in *attribute), 294
msl.equipment.resources.picotech.picoscope.PULSESWIDTH_CONDITIONS_SOURCE
319
*(msl.equipment.resources.picotech.picoscope.enums.PS40**

PS6000SigGenTrigType (class in *attribute), 294
msl.equipment.resources.picotech.picoscope.PULSESWIDTH_CONDITIONS_STATE
319
*(msl.equipment.resources.picotech.picoscope.enums.PS40**

ps6000StreamingReady (in module *attribute), 294
msl.equipment.resources.picotech.picoscope.PULSESWIDTH_NO_OF_BLOCK_CONDITIONS
230
*(msl.equipment.resources.picotech.picoscope.enums.PS40**

PS6000SweepType (class in *attribute), 294
msl.equipment.resources.picotech.picoscope.PULSESWIDTH_NO_OF_CONDITIONS
317
*(msl.equipment.resources.picotech.picoscope.enums.PS40**

PS6000ThresholdDirection (class in *attribute), 294
msl.equipment.resources.picotech.picoscope.PULSESWIDTH_PROPERTIES
320
*(msl.equipment.resources.picotech.picoscope.enums.PS40**

PS6000ThresholdMode (class in *attribute), 294
msl.equipment.resources.picotech.picoscope.PULSESWIDTH_PROPERTIES_DIRECTION
320
*(msl.equipment.resources.picotech.picoscope.enums.PS40**

PS6000TimeUnits (class in *attribute), 294
msl.equipment.resources.picotech.picoscope.PULSESWIDTH_PROPERTIES_LOWER
317
*(msl.equipment.resources.picotech.picoscope.enums.PS40**

PS6000TriggerChannelProperties (class in *attribute), 294
msl.equipment.resources.picotech.picoscope.PULSESWIDTH_PROPERTIES_TYPE
370
*(msl.equipment.resources.picotech.picoscope.enums.PS40**

PS6000TriggerConditions (class in *attribute), 294
msl.equipment.resources.picotech.picoscope.PULSESWIDTH_PROPERTIES_UPPER
369
*(msl.equipment.resources.picotech.picoscope.enums.PS40**

PS6000TriggerState (class in *attribute), 294
msl.equipment.resources.picotech.picoscope.PULSESWIDTH_SOURCE
321
*(msl.equipment.resources.picotech.picoscope.enums.PS40**

PS6000WaveType (class in *attribute), 281
msl.equipment.resources.picotech.picoscope.PulseWidthQualifier
318
*(msl.equipment.resources.picotech.picoscope.structs.PS20**

PS640X_C_D_MAX_SIG_GEN_BUFFER_SIZE (attribute), 359

`pulseWidthQualifier` `PZ_FeedbackLoopConstants` (class in `(msl.equipment.resources.picotech.picoscope structs.PS2000ATriggerConditions`, attribute), 358 579

`pulseWidthQualifier` `PZ_Fixed` (`msl.equipment.resources.thorlabs.kinesis.enums.PZ_Output` (`msl.equipment.resources.picotech.picoscope structs.PS3000ATriggerConditions` attribute), 362 446

`pulseWidthQualifier` `PZ_InputSourceFlags` (class in `msl.equipment.resources.thorlabs.kinesis.enums`), (`msl.equipment.resources.picotech.picoscope structs.PS3000ATriggerConditionsV2` attribute), 362 446

`pulseWidthQualifier` `PZ_InputTrigEnable` (`msl.equipment.resources.thorlabs.kinesis.enums.PZ_Output` (`msl.equipment.resources.picotech.picoscope structs.PS3000ATriggerConditions` attribute), 361 446

`pulseWidthQualifier` `PZ_InputTrigSenseHigh` (`msl.equipment.resources.thorlabs.kinesis.enums.PZ_Output` (`msl.equipment.resources.picotech.picoscope structs.PS4000ATriggerConditions` attribute), 364 447

`pulseWidthQualifier` `PZ_LUTWaveParameters` (class in `msl.equipment.resources.thorlabs.kinesis.structs`), (`msl.equipment.resources.picotech.picoscope structs.PS5000ATriggerConditions` attribute), 368 446

`pulseWidthQualifier` `PZ_OpenLoop` (`msl.equipment.resources.thorlabs.kinesis.enums.KN_Flags` attribute), 465

`pulseWidthQualifier` `PZ_OpenLoopSmooth` (`msl.equipment.resources.picotech.picoscope structs.PS1000ATriggerConditions` attribute), 369 466

`PyVISA` (`msl.equipment.constants.Backend` attribute), 50 `PZ_OpenLoopSmooth` (`msl.equipment.resources.thorlabs.kinesis.enums.PZ_Con`

`PyVISA_LIBRARY` (`msl.equipment.config.Config` attribute), 18 `PZ_OutputGated` (`msl.equipment.resources.thorlabs.kinesis.enums.PZ_Co`

`PZ_All` (`msl.equipment.resources.thorlabs.kinesis.enums.PZ_InputSourceFlags` attribute), 446 `PZ_OutputGated` (`msl.equipment.resources.thorlabs.kinesis.enums.PZ_Out`

`PZ_CloseLoop` (`msl.equipment.resources.thorlabs.kinesis.enums.PZ_OutputLTMModeFeedbackModeTypes` attribute), 446 `PZ_OutputLTMModeFeedbackModeTypes` (`msl.equipment.resources.thorlabs.kinesis.enums`), 446

`PZ_CloseLoop` (`msl.equipment.resources.thorlabs.kinesis.enums.PZ_ControlModeTypes` attribute), 446 `PZ_ControlModeTypes` 446

`PZ_CloseLoopSmooth` `PZ_OutputTrigEnable` (`msl.equipment.resources.thorlabs.kinesis.enums.PZ_Out`

`PZ_CloseLoopSmooth` (`msl.equipment.resources.thorlabs.kinesis.enums.KNAttFlagModeTypes` attribute), 446 `PZ_AttFlagModeTypes` 446

`PZ_CloseLoopSmooth` `PZ_OutputTrigRepeat` (`msl.equipment.resources.thorlabs.kinesis.enums.PZ_Out`

`PZ_CloseLoopSmooth` (`msl.equipment.resources.thorlabs.kinesis.enums.PZ_ControlModeTypes` attribute), 446 `PZ_ControlModeTypes` 446

`PZ_Continuous` (`msl.equipment.resources.thorlabs.kinesis.enums.PZ_OutputLTMMode` attribute), 446 `PZ_OutputLTMMode` (`msl.equipment.resources.thorlabs.kinesis.enums.PZ_Out`

`PZ_ControlModeTypes` (class in `PZ_Potentiometer` (`msl.equipment.resources.thorlabs.kinesis.enums`), 445 `PZ_Input` (`msl.equipment.resources.thorlabs.kinesis.enums.PZ_Inpu`

`PZ_ControlModeUndefined` `PZ_SoftwareOnly` (`msl.equipment.resources.thorlabs.kinesis.enums.KNAttFlagModeTypes` attribute), 445 `PZ_AttFlagModeTypes` 445

`PZ_ExternalSignal` `PZ_Undefined` (`msl.equipment.resources.thorlabs.kinesis.enums.PZ_Input` (`msl.equipment.resources.thorlabs.kinesis.enums.PZ_AttFlagModeTypes` attribute), 446 `PZ_AttFlagModeTypes` 446

Q	QD_OpenLoopHoldValues (class in <i>msl.equipment.resources.thorlabs.kinesis.enums</i>),
QD_AutoOpenClosedLoop	(<i>msl.equipment.resources.thorlabs.kinesis.enums.QD_OperatingMode</i> attribute), 468
QD_ClosedLoop	(<i>msl.equipment.resources.thorlabs.kinesis.enums.QD_OperatingMode</i> attribute), 468
QD_Disabled	(<i>msl.equipment.resources.thorlabs.kinesis.enums.QD_FilterEnable</i> class in <i>msl.equipment.resources.thorlabs.kinesis structs</i>), 469
QD_Enabled	(<i>msl.equipment.resources.thorlabs.kinesis.enums.QD_FilterEnable</i> attribute), 469
QD_FilterEnable	(class in <i>msl.equipment.resources.thorlabs.kinesis structs</i>), 469
QD_HoldOnLastValue	(<i>msl.equipment.resources.thorlabs.kinesis.enums.QD_OpenLoopHoldValues</i> attribute), 469
QD_HoldOnZero	(<i>msl.equipment.resources.thorlabs.kinesis.enums.QD_OpenLoopHoldValues</i> attribute), 469
QD_HubAndSMA	(<i>msl.equipment.resources.thorlabs.kinesis.enums.QD_LowVoltageRoute</i> attribute), 469
QD_KPA_DigitalIO	(class in <i>msl.equipment.resources.thorlabs.kinesis structs</i>), 591
QD_KPA_TrigIOConfig	(class in <i>QD_Disabled</i> (<i>msl.equipment.resources.thorlabs.kinesis.enums.QD_KPA</i> attribute), 470)
QD_KPA_TrigModes	(class in <i>QD_TrigIn_GPI</i> (<i>msl.equipment.resources.thorlabs.kinesis.enums.QD_KPA</i> attribute), 470)
QD_KPA_TrigPolarities	(class in <i>QD_TrigIn_GPI</i> (<i>msl.equipment.resources.thorlabs.kinesis.enums.QD_KPA</i> attribute), 470)
QD_LoopParameters	(class in <i>Quad</i> (<i>msl.equipment.resources.thorlabs.kinesis structs</i>), 588)
QD_LowPassFilterParameters	(class in <i>Quad</i> (<i>msl.equipment.resources.picotech.picoscope.enums.PS2000A</i> attribute), 248)
QD_LowVoltageRoute	(class in <i>QUAD</i> (<i>msl.equipment.resources.picotech.picoscope.enums.PS3000A</i> attribute), 265)
QD_ModeUndefined	(<i>msl.equipment.resources.thorlabs.kinesis.enums.QD_OperatingMode</i> attribute), 468
QD_Monitor	(<i>msl.equipment.resources.thorlabs.kinesis.enums.QD_OperatingMode</i> attribute), 310
QD_NotchFilterParameters	(class in <i>QUAD</i> (<i>msl.equipment.resources.picotech.picoscope.enums.PS5000A</i> attribute), 301)
QD_OpenLoop	(<i>msl.equipment.resources.thorlabs.kinesis.enums.QD_OperatingMode</i> attribute), 468
	query() (<i>msl.equipment.connection_message_based.ConnectionM</i>

method), 31	R_10MV (msl.equipment.resources.picotech.picoscope.enums.PS6000ARange)
query() (msl.equipment.connection_prologix.ConnectionPrologixAttribute), 316	R_10V (msl.equipment.resources.picotech.picoscope.enums.PS2000ARange)
method), 35	R_10V (msl.equipment.resources.picotech.picoscope.enums.PS2000ARange)
query_auto (msl.equipment.connection_prologix.ConnectionPrologix), 243	R_10V (msl.equipment.resources.picotech.picoscope.enums.PS3000ARange)
property), 36	R_10V (msl.equipment.resources.picotech.picoscope.enums.PS3000ARange)
Quickest (msl.equipment.resources.thorlabs.kinesis.enums.MOTFileMovementDirections attribute), 437	R_10V (msl.equipment.resources.picotech.picoscope.enums.PS3000ARange)
R	R_10V (msl.equipment.resources.picotech.picoscope.enums.PS3000ARange)
R_100MV (msl.equipment.resources.picotech.picoscope.enums.PS2000ARange attribute), 243	R_10V (msl.equipment.resources.picotech.picoscope.enums.PS4000ARange)
R_100MV (msl.equipment.resources.picotech.picoscope.enums.PS2000ARange attribute), 234	R_10V (msl.equipment.resources.picotech.picoscope.enums.PS4000ARange)
R_100MV (msl.equipment.resources.picotech.picoscope.enums.PS3000ARange attribute), 261	R_10V (msl.equipment.resources.picotech.picoscope.enums.PS5000ARange)
R_100MV (msl.equipment.resources.picotech.picoscope.enums.PS3000ARange attribute), 252	R_10V (msl.equipment.resources.picotech.picoscope.enums.PS5000ARange)
R_100MV (msl.equipment.resources.picotech.picoscope.enums.PS4000ARange attribute), 283	R_10V (msl.equipment.resources.picotech.picoscope.enums.PS6000ARange)
R_100MV (msl.equipment.resources.picotech.picoscope.enums.PS4000ARange attribute), 270	R_1100K (msl.equipment.resources.picotech.picoscope.enums.PS4000ARange)
R_100MV (msl.equipment.resources.picotech.picoscope.enums.PS5000ARange attribute), 307	R_1V (msl.equipment.resources.picotech.picoscope.enums.PS2000ARange)
R_100MV (msl.equipment.resources.picotech.picoscope.enums.PS5000ARange attribute), 297	R_1V (msl.equipment.resources.picotech.picoscope.enums.PS2000ARange)
R_100MV (msl.equipment.resources.picotech.picoscope.enums.PS6000ARange attribute), 316	R_1V (msl.equipment.resources.picotech.picoscope.enums.PS3000ARange)
R_100V (msl.equipment.resources.picotech.picoscope.enums.PS3000ARange attribute), 252	R_1V (msl.equipment.resources.picotech.picoscope.enums.PS3000ARange)
R_100V (msl.equipment.resources.picotech.picoscope.enums.PS4000ARange attribute), 283	R_1V (msl.equipment.resources.picotech.picoscope.enums.PS4000ARange)
R_100V (msl.equipment.resources.picotech.picoscope.enums.PS4000ARange attribute), 271	R_1V (msl.equipment.resources.picotech.picoscope.enums.PS4000ARange)
R_10M (msl.equipment.resources.picotech.picoscope.enums.PS1000AResidenceRange attribute), 284	R_1V (msl.equipment.resources.picotech.picoscope.enums.PS5000ARange)
R_10MV (msl.equipment.resources.picotech.picoscope.enums.PS2000ARange attribute), 243	R_1V (msl.equipment.resources.picotech.picoscope.enums.PS5000ARange)
R_10MV (msl.equipment.resources.picotech.picoscope.enums.PS2000ARange attribute), 234	R_1V (msl.equipment.resources.picotech.picoscope.enums.PS6000ARange)
R_10MV (msl.equipment.resources.picotech.picoscope.enums.PS3000ARange attribute), 261	R_1V (msl.equipment.resources.picotech.picoscope.enums.PS6000ARange)
R_10MV (msl.equipment.resources.picotech.picoscope.enums.PS3000ARange attribute), 252	R_200MV (msl.equipment.resources.picotech.picoscope.enums.PS2000ARange)
R_10MV (msl.equipment.resources.picotech.picoscope.enums.PS4000ARange attribute), 283	R_200MV (msl.equipment.resources.picotech.picoscope.enums.PS3000ARange)
R_10MV (msl.equipment.resources.picotech.picoscope.enums.PS4000ARange attribute), 270	R_200MV (msl.equipment.resources.picotech.picoscope.enums.PS3000ARange)
R_10MV (msl.equipment.resources.picotech.picoscope.enums.PS5000ARange attribute), 306	R_200MV (msl.equipment.resources.picotech.picoscope.enums.PS4000ARange)
R_10MV (msl.equipment.resources.picotech.picoscope.enums.PS5000ARange attribute), 297	R_200MV (msl.equipment.resources.picotech.picoscope.enums.PS4000ARange)

attribute), 297

R_50MV (*msl.equipment.resources.picotech.picoscope.enums.PS6000ARange*)³⁸⁴
attribute), 316

R_50V (*msl.equipment.resources.picotech.picoscope.enums.PS2000ARange*)³⁰⁷
attribute), 243

R_50V (*msl.equipment.resources.picotech.picoscope.enums.PS2000ARange*)²⁹⁸
attribute), 234

R_50V (*msl.equipment.resources.picotech.picoscope.enums.PS2000ARange*)³¹⁶
attribute), 261

RackDevice (in module)

R_50V (*msl.equipment.resources.picotech.picoscope.enums.PS2000ARange*)³⁰⁷
attribute), 252

RaicolTEC (class in)

R_50V (*msl.equipment.resources.picotech.picoscope.RaicolTECRaisePS5000ARange*)⁵⁶
attribute), 283

R_50V (*msl.equipment.resources.picotech.picoscope.enums.PS4000ARange*)⁴⁰¹
attribute), 271

R_50V (*msl.equipment.resources.picotech.picoscope.raisePS5000ARange*)
(*msl.equipment.connection.Connection*)

R_50V (*msl.equipment.resources.picotech.picoscope.enums.PS5000ARange*)³⁰
attribute), 298

raise_timeout()

R_50V (*msl.equipment.resources.picotech.picoscope.enums.PS6000ARange*)³⁰
attribute), 316

method), 30

R_5V (*msl.equipment.resources.picotech.picoscope.enums.PS4000ARange*)²⁴⁶
attribute), 243

R_5V (*msl.equipment.resources.picotech.picoscope.enums.PS4000ARange*)²⁶³
attribute), 234

R_5V (*msl.equipment.resources.picotech.picoscope.enums.PS4000ARange*)²⁸⁵
attribute), 261

R_5V (*msl.equipment.resources.picotech.picoscope.enums.PS4000ARange*)²⁷⁵
attribute), 252

R_5V (*msl.equipment.resources.picotech.picoscope.enums.PS4000ARange*)³⁰⁹
attribute), 283

R_5V (*msl.equipment.resources.picotech.picoscope.enums.PS4000ARange*)²⁹⁹
attribute), 271

R_5V (*msl.equipment.resources.picotech.picoscope.enums.PS4000ARange*)³¹⁸
attribute), 307

R_5V (*msl.equipment.resources.picotech.picoscope.enums.PS4000ARange*)³¹⁸
attribute), 298

RAMP_MAX_FREQUENCY (*msl.equipment.resources.picotech.picoscope.ps2000a.Pic*)

R_5V (*msl.equipment.resources.picotech.picoscope.enums.PS6000ARange*)³⁴⁵
attribute), 316

RAMP_MAX_FREQUENCY

R_ADCV (*msl.equipment.resources.picotech.picoscope.enums.PS4000ARanges*)³⁴⁸
attribute), 284

R_MAX (*msl.equipment.resources.picotech.picoscope.RAMP_MAX_FREQUENCY*)
attribute), 243

R_MAX (*msl.equipment.resources.picotech.picoscope.enums.PS2000ARange*)³⁵²
attribute), 234

RAMP_MAX_FREQUENCY

R_MAX (*msl.equipment.resources.picotech.picoscope.enums.PS3000ARange*)³⁵⁴
attribute), 261

R_MAX (*msl.equipment.resources.picotech.picoscope.RAMP_MAX_FREQUENCY*)
attribute), 252

RAMP_MAX_FREQUENCY (*msl.equipment.resources.picotech.picoscope.ps5000a.Pic*)

R_MAX (*msl.equipment.resources.picotech.picoscope.enums.PS4000ARange*)³⁵⁵
attribute), 283

RAMP_MAX_FREQUENCY

(*msl.equipment.resources.picotech.picoscope.ps6000uPathType*)
attribute), 357
read_reply() (*msl.equipment.vxi11.VXIClient*
RAMP_UP (*msl.equipment.resources.picotech.picoscope.enums.PS4000AWaveType*
attribute), 246
read_stb() (*msl.equipment.connection_tcpip_hislip.ConnectionT*
RAMP_UP (*msl.equipment.resources.picotech.picoscope.enums.PS4000AWaveType*
attribute), 263
read_stb() (*msl.equipment.connection_tcpip_vxi11.ConnectionT*
RAMP_UP (*msl.equipment.resources.picotech.picoscope.enums.PS4000AWaveType*
attribute), 285
read_termination
RAMP_UP (*msl.equipment.resources.picotech.picoscope.enums.PS4000WaveType*)
connection_gpib.ConnectionGPIB
attribute), 275
property), 27
read_stb() (*msl.equipment.connection_message_based.ConnectionM*
attribute), 309
RAMP_UP (*msl.equipment.resources.picotech.picoscope.enums.PS5000WaveType*
attribute), 299
read_termination
RAMP_UP (*msl.equipment.resources.picotech.picoscope.enums.PS4000WaveType*)
connection_prologix.ConnectionPrologix
attribute), 318
property), 34
RAMPDOWN (*msl.equipment.resources.picotech.picoscope.enums.PS2000Wav*
resources.aim_tti.mx_series.MXSeries
attribute), 237
method), 94
RAMPUP (*msl.equipment.resources.picotech.picoscope.enums.PS2000Wav*
resources.aim_tti.mx_series.MXSeries
attribute), 237
method), 94
RANGES (*msl.equipment.resources.picotech.picoscope.enums.PS2000A*
GhsClient.hislip.SyncClient
attribute), 244
method), 75
RANGES (*msl.equipment.resources.picotech.picoscope.enums.PS3000A*
ChassisClient.connection_socket.ConnectionSocket
attribute), 262
method), 42
RANGES (*msl.equipment.resources.picotech.picoscope.enums.PS4000A*
ChassisClient.connection_tcpip_hislip.Connection
attribute), 292
method), 44
RANGES (*msl.equipment.resources.picotech.picoscope.enums.PS4000A*
ChassisClient.connection_tcpip_vxi11.Connection
attribute), 273
method), 46
RANGES (*msl.equipment.resources.picotech.picoscope.enums.PS5000A*
ChassisClient.connection_zeromq.ConnectionZero
attribute), 313
method), 49
RANGES (*msl.equipment.resources.picotech.picoscope.enums.RecordP*
S5000ChassisClient.record_types), 87
attribute), 303
RecordDict (class in
raw (*msl.equipment.resources.picotech.picoscope.channel.PicoScopeChannel*)
record_types), 86
property), 232
records() (*msl.equipment.database.Database*
READ (*msl.equipment.resources.picotech.picoscope.enums.PS4000AWaveType*)
attribute), 287
Operation
read() (*msl.equipment.connection_message_based.ConnectionMessageBased*
method), 30
RegBusy (*msl.equipment.resources.nkt.nktpdll.NKT.RegisterStatus*
method), 172
RegBusy (*msl.equipment.resources.nkt.nktpdll.RegisterStatusTypes*
attribute), 167
read() (*msl.equipment.connection_prologix.ConnectionPrologix*)
attribute), 35
RegBusy (*msl.equipment.resources.nkt.nktpdll.RegisterStatusTypes*
attribute), 167
read() (*msl.equipment.hislip.HiSLIPClient*)
method), 74
RegComError (*msl.equipment.resources.nkt.nktpdll.NKT.RegisterS*
attribute), 172
read() (*msl.equipment.resources.bentham.benhw32.Bentham*)
attribute), 172
RegComError (*msl.equipment.resources.nkt.nktpdll.RegisterStatus*
attribute), 167
read() (*msl.equipment.vxi11.RPCClient*)
method), 604
RegCRCErr (*msl.equipment.resources.nkt.nktpdll.NKT.RegisterStat*
attribute), 167
read_authentication_exchange()
(*msl.equipment.hislip.SyncClient*
method), 76
RegCRCErr (*msl.equipment.resources.nkt.nktpdll.RegisterStatusType*
attribute), 167
read_ol756_flash_settings()
(*msl.equipment.resources.optronic_laboratories.ol756*)
attribute), 172
RegData_Ascii (*msl.equipment.resources.nkt.nktpdll.NKT.Register*
attribute), 167
attribute), 172
OL756

RegData_Ascii (*msl.equipment.resources.nkt.nktpdll.RegisterDataTypes*)
attribute), 166
RegData_B16 (*msl.equipment.resources.nkt.nktpdll.NKT.RegisterDataTypes*)
attribute), 171
RegData_B16 (*msl.equipment.resources.nkt.nktpdll.RegisterDataTypes*)
attribute), 167
RegData_B32 (*msl.equipment.resources.nkt.nktpdll.NKT.RegisterDataTypes*)
attribute), 171
RegData_B32 (*msl.equipment.resources.nkt.nktpdll.RegisterDataTypes*), 171
RegData_B64 (*msl.equipment.resources.nkt.nktpdll.NKT.RegisterDataTypes*)
attribute), 171
RegData_B64 (*msl.equipment.resources.nkt.nktpdll.RegisterDataTypes*), 171
RegData_B8 (*msl.equipment.resources.nkt.nktpdll.NKT.RegisterDataTypes*)
attribute), 171
RegData_B8 (*msl.equipment.resources.nkt.nktpdll.RegisterDataTypes*), 171
RegData_DateTime
(*msl.equipment.resources.nkt.nktpdll.NKT.RegisterDataTypes*)
attribute), 172
RegData_DateTime
(*msl.equipment.resources.nkt.nktpdll.RegisterDataTypes*)
attribute), 167
RegData_F32 (*msl.equipment.resources.nkt.nktpdll.NKT.RegisterDataTypes*)
attribute), 171
RegData_F32 (*msl.equipment.resources.nkt.nktpdll.RegisterDataTypes*), 166
RegData_F64 (*msl.equipment.resources.nkt.nktpdll.NKT.RegisterDataTypes*)
attribute), 171
RegData_F64 (*msl.equipment.resources.nkt.nktpdll.RegisterDataTypes*), 166
RegData_H16 (*msl.equipment.resources.nkt.nktpdll.NKT.RegisterDataTypes*)
attribute), 171
RegData_H16 (*msl.equipment.resources.nkt.nktpdll.RegisterDataTypes*), 166
RegData_H32 (*msl.equipment.resources.nkt.nktpdll.NKT.RegisterDataTypes*)
attribute), 171
RegData_H32 (*msl.equipment.resources.nkt.nktpdll.RegisterDataTypes*), 166
RegData_H64 (*msl.equipment.resources.nkt.nktpdll.NKT.RegisterDataTypes*)
attribute), 171
RegData_H64 (*msl.equipment.resources.nkt.nktpdll.RegisterDataTypes*)
attribute), 167
RegData_H8 (*msl.equipment.resources.nkt.nktpdll.NKT.RegisterDataTypes*)
attribute), 171
RegData_H8 (*msl.equipment.resources.nkt.nktpdll.RegisterDataTypes*)
attribute), 167
RegData_Mixed (*msl.equipment.resources.nkt.nktpdll.NKT.RegisterDataTypes*)
attribute), 171
RegData_Mixed (*msl.equipment.resources.nkt.nktpdll.RegisterDataTypes*)
attribute), 166
RegData_Parmset
RegData_Parmset
(*msl.equipment.resources.nkt.nktpdll.NKT.RegisterDataTypes*)
attribute), 171
RegData_S16 (*msl.equipment.resources.nkt.nktpdll.NKT.RegisterDataTypes*)
attribute), 167
RegData_S16 (*msl.equipment.resources.nkt.nktpdll.RegisterDataTypes*)
attribute), 171
RegData_S32 (*msl.equipment.resources.nkt.nktpdll.NKT.RegisterDataTypes*)
attribute), 166
RegData_S32 (*msl.equipment.resources.nkt.nktpdll.RegisterDataTypes*)
attribute), 171
RegData_S64 (*msl.equipment.resources.nkt.nktpdll.NKT.RegisterDataTypes*)
attribute), 166
RegData_S64 (*msl.equipment.resources.nkt.nktpdll.RegisterDataTypes*)
attribute), 171
RegData_S8 (*msl.equipment.resources.nkt.nktpdll.RegisterDataTypes*)
attribute), 166
RegData_U16 (*msl.equipment.resources.nkt.nktpdll.NKT.RegisterDataTypes*)
attribute), 166
RegData_U16 (*msl.equipment.resources.nkt.nktpdll.RegisterDataTypes*)
attribute), 171
RegData_U32 (*msl.equipment.resources.nkt.nktpdll.NKT.RegisterDataTypes*)
attribute), 171
RegData_U32 (*msl.equipment.resources.nkt.nktpdll.RegisterDataTypes*)
attribute), 166
RegData_U64 (*msl.equipment.resources.nkt.nktpdll.NKT.RegisterDataTypes*)
attribute), 171
RegData_U64 (*msl.equipment.resources.nkt.nktpdll.RegisterDataTypes*)
attribute), 166
RegData_U8 (*msl.equipment.resources.nkt.nktpdll.NKT.RegisterDataTypes*)
attribute), 171
RegData_U8 (*msl.equipment.resources.nkt.nktpdll.RegisterDataTypes*)
attribute), 167
RegData_Unknown
RegData_Unknown
register()
(in
module
register()
(*msl.equipment.resources.avantes.avaspec.Avantes*), 87
register()
(*msl.equipment.resources.avantes.avaspec.Avantes*)
register_create()
register_create()

method), 180
register_exists()
 (*msl.equipment.resources.nkt.nktpdll.NKT*
 method), 180
register_get_all()
 (*msl.equipment.resources.nkt.nktpdll.NKT*
 method), 180
register_message_callback()
 (*msl.equipment.resources.thorlabs.kinesis.benchtop_StepEquipoenBreschunqeSteppenMotibNKT*
 method), 417
register_message_callback()
 (*msl.equipment.resources.thorlabs.kinesis.filter_flipperFilterflipper:resources.nkt.nktpdll.NKT*
 method), 482
register_message_callback()
 (*msl.equipment.resources.thorlabs.kinesis.integrated_StepEquipoenMotibNKT*
 method), 494
register_message_callback()
 (*msl.equipment.resources.thorlabs.kinesis.kcube_dc_gearboxKCubeGearsources.nkt.nktpdll.NKT*
 method), 517
register_message_callback()
 (*msl.equipment.resources.thorlabs.kinesis.kcube_soleKCubeSoleresources.nkt.nktpdll.NKT*
 method), 534
register_message_callback()
 (*msl.equipment.resources.thorlabs.kinesis.kcube_stepStepEquipoenKCubesStappesMotoktpdll.NKT*
 method), 551
register_read()
 (*msl.equipment.resources.nkt.nktpdll.NKT*
 method), 179
register_read_ascii()
 (*msl.equipment.resources.nkt.nktpdll.NKT*
 method), 180
register_read_f32()
 (*msl.equipment.resources.nkt.nktpdll.NKT*
 method), 181
register_read_f64()
 (*msl.equipment.resources.nkt.nktpdll.NKT*
 method), 181
register_read_s16()
 (*msl.equipment.resources.nkt.nktpdll.NKT*
 method), 181
register_read_s32()
 (*msl.equipment.resources.nkt.nktpdll.NKT*
 method), 182
register_read_s64()
 (*msl.equipment.resources.nkt.nktpdll.NKT*
 method), 182
register_read_s8()
 (*msl.equipment.resources.nkt.nktpdll.NKT*
 method), 182
register_read_u16()
 (*msl.equipment.resources.nkt.nktpdll.NKT*
 method), 183
register_read_u32()
 (*msl.equipment.resources.nkt.nktpdll.NKT*
 method), 183
register_read_u64()
 (*msl.equipment.resources.nkt.nktpdll.NKT*
 method), 183
register_read_u8()
 (*msl.equipment.resources.nkt.nktpdll.NKT*
 method), 184
register_remove()
register_remove_all()
register_write()
 (*msl.equipment.resources.nkt.nktpdll.NKT*
 method), 185
register_write_ascii()
register_write_f32()
register_write_f64()
register_write_read()
register_write_read_ascii()
register_write_read_f32()
register_write_read_f64()
register_write_read_s16()
register_write_read_s32()
register_write_read_s64()
register_write_read_s8()

method), 188	167
register_write_read_u16() (msl.equipment.resources.nkt.nktpdll.NKT method), 189	RegNacked(msl.equipment.resources.nkt.nktpdll.NKT.RegisterStatusType attribute), 172
register_write_read_u32() (msl.equipment.resources.nkt.nktpdll.NKT method), 189	RegNacked(msl.equipment.resources.nkt.nktpdll.RegisterStatusType attribute), 167
register_write_read_u64() (msl.equipment.resources.nkt.nktpdll.NKT method), 190	RegPriority_High (msl.equipment.resources.nkt.nktpdll.NKT.RegisterPriorityType attribute), 171
register_write_read_u8() (msl.equipment.resources.nkt.nktpdll.NKT method), 190	RegPriority_High (msl.equipment.resources.nkt.nktpdll.RegisterPriorityType attribute), 166
register_write_s16() (msl.equipment.resources.nkt.nktpdll.NKT method), 190	RegPriority_Low (msl.equipment.resources.nkt.nktpdll.RegisterPriorityType attribute), 171
register_write_s32() (msl.equipment.resources.nkt.nktpdll.NKT method), 191	RegSuccess(msl.equipment.resources.nkt.nktpdll.NKT.RegisterStatusType attribute), 172
register_write_s64() (msl.equipment.resources.nkt.nktpdll.NKT method), 191	RegSuccess(msl.equipment.resources.nkt.nktpdll.RegisterStatusType attribute), 167
register_write_s8() (msl.equipment.resources.nkt.nktpdll.NKT method), 191	RegTimeout(msl.equipment.resources.nkt.nktpdll.NKT.RegisterStatusType attribute), 167
register_write_u16() (msl.equipment.resources.nkt.nktpdll.NKT method), 192	RejectStatus (class in msl.equipment.vxi11), 602
register_write_u32() (msl.equipment.resources.nkt.nktpdll.NKT method), 192	relativeReading (msl.equipment.resources.thorlabs.kinesis structs.KNA_TI attribute), 586
register_write_u64() (msl.equipment.resources.nkt.nktpdll.NKT method), 192	relativeReading (msl.equipment.resources.thorlabs.kinesis structs.NT_TIA attribute), 578
register_write_u8() (msl.equipment.resources.nkt.nktpdll.NKT method), 193	release_stream_buffer() (msl.equipment.resources.picotech.picoscope.ps3000.Pico method), 346
RegisterDataTypes (class in msl.equipment.resources.nkt.nktpdll), 166	remote() (msl.equipment.connection_tcpip_vxi11.ConnectionTCP method), 47
RegisterPriorityTypes (class in msl.equipment.resources.nkt.nktpdll), 166	remote_enable() (msl.equipment.connection_gpib.ConnectionGPIB method), 27
RegisterStatusCallback (in module msl.equipment.resources.nkt.nktpdll), 162	remote_local_control() (msl.equipment.connection_tcpip_hislip.ConnectionTCP method), 45
RegisterStatusCallback (msl.equipment.resources.nkt.nktpdll.NKT attribute), 168	RemoteHostIp(msl.equipment.resources.avantes.avaspec.Avantes attribute), 110
RegisterStatusTypes (class in msl.equipment.resources.nkt.nktpdll),	RemoteHostIp(msl.equipment.resources.avantes.avaspec.Broadca attribute), 99
	repack() (msl.equipment.hislip.Message static method), 60
	REPLY (msl.equipment.vxi11.MessageType at-

tribute), 602
ReplyStatus (class in `msl.equipment.vxi11`), 602
report_date (`msl.equipment.record_types.CalibrationRecord`.*method*), 417
 attribute), 83
report_error()
 (msl.equipment.resources.bentham.benhw32.Bentham32).*method*), 517
 method), 129
report_number (`msl.equipment.record_types.CalibrationRecord`.*method*), 551
 attribute), 83
request_backlash()
 (msl.equipment.resources.thorlabs.kinesis.benchtop_kinematics.kcube_dc_servo).*method*), 417
request_backlash()
 (msl.equipment.resources.thorlabs.kinesis.integrated_stacks_equipment.BenchtopStepperMotors).*method*), 552
request_backlash()
 (msl.equipment.resources.thorlabs.kinesis.kcube_dc_gyro).*method*), 418
request_backlash()
 (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo).*method*), 518
request_bow_index()
 (msl.equipment.resources.thorlabs.kinesis.benchtop_kinematics.kcube_dc_servo).*method*), 417
request_bow_index()
 (msl.equipment.resources.thorlabs.kinesis.integrated_stacks_equipment.BenchtopStepperMotors).*method*), 552
request_bow_index()
 (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo).*method*), 518
request_button_params()
 (msl.equipment.resources.thorlabs.kinesis.integrated_stacks_equipment.BenchtopStepperMotors).*method*), 418
request_cycle_params()
 (msl.equipment.resources.thorlabs.kinesis.kcube_solenoid).*method*), 480
request_dcpid_params()
 (msl.equipment.resources.thorlabs.kinesis.benchtop_kinematics.kcube_dc_servo).*method*), 418
request_digital_outputs()
 (msl.equipment.resources.thorlabs.kinesis.benchtop_kinematics.kcube_dc_servo).*method*), 418
request_digital_outputs()
 (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo).*method*), 518
request_digital_outputs()
 (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo).*method*), 552
request_digital_outputs()
 (msl.equipment.resources.thorlabs.kinesis.integrated_stacks_equipment.BenchtopStepperMotors).*method*), 418
request_encoder_counter()
 (msl.equipment.resources.thorlabs.kinesis.benchtop_stepper_motors).*method*), 417
request_encoder_counter()
 (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo).*method*), 518
request_encoder_counter()
 (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo).*method*), 551
request_front_panel_locked()
 (msl.equipment.resources.thorlabs.kinesis.integrated_stacks_equipment.BenchtopStepperMotors).*method*), 495
request_homing_params()
 (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo).*method*), 418
request_homing_params()
 (msl.equipment.resources.thorlabs.kinesis.integrated_stacks_equipment.BenchtopStepperMotors).*method*), 495
request_homing_params()
 (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo).*method*), 518
request_hub_bay()
 (msl.equipment.resources.thorlabs.kinesis.kcube_solenoid).*method*), 480
request_input_voltage()
 (msl.equipment.resources.thorlabs.kinesis.integrated_stacks_equipment.BenchtopStepperMotors).*method*), 418
request_io_settings()
 (msl.equipment.resources.thorlabs.kinesis.filter_flipper.FilterFlipper).*method*), 480
request_jog_params()
 (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo).*method*), 418
request_jog_params()
 (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo).*method*), 518
request_jog_params()
 (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo).*method*), 552
request_joystick_params()
 (msl.equipment.resources.thorlabs.kinesis.integrated_stacks_equipment.BenchtopStepperMotors).*method*), 418

```

request_key() (msl.equipment.resources.mks_instruments.pmt0D0pulPRA000Bsources.thorlabs.kinesis.kcube_stepper_
    method), 157
    method), 552
request_led_switches() request_operating_mode()
    (msl.equipment.resources.thorlabs.kinesis.kcube_dc_<del>KCubeSolenoid</del>G8sources.thorlabs.kinesis.kcube_solenoid_
    method), 518
    method), 535
request_led_switches() request_operating_state()
    (msl.equipment.resources.thorlabs.kinesis.kcube_solenoid<del>KCubeSolenoid</del>G8sources.thorlabs.kinesis.kcube_solenoid_
    method), 535
    method), 535
request_limit_switch_params() request_pid_loop_encoder_params()
    (msl.equipment.resources.thorlabs.kinesis.benchtop_<del>step</del>equipotentialBenchtopStepperMotors.kinesis.benchtop_stepper_
    method), 418
    method), 419
request_limit_switch_params() request_pid_loop_encoder_params()
    (msl.equipment.resources.thorlabs.kinesis.integrated_<del>step</del>pqniotats:inagatedSothp:Motoris.kcube_stepper_
    method), 495
    method), 552
request_limit_switch_params() request_pos_trigger_params()
    (msl.equipment.resources.thorlabs.kinesis.kcube_dc_<del>KCubeSolenoid</del>G8sources.thorlabs.kinesis.kcube_dc_servo_
    method), 518
    method), 518
request_limit_switch_params() request_pos_trigger_params()
    (msl.equipment.resources.thorlabs.kinesis.kcube_steppos_equipotentialKCubesStoppersMotabs.kinesis.kcube_stepper_
    method), 552
    method), 552
request_mmi_params() request_position()
    (msl.equipment.resources.thorlabs.kinesis.kcube_dc_<del>KCubeSolenoid</del>G8sources.thorlabs.kinesis.benchtop_stepper_
    method), 518
    method), 419
request_mmi_params() request_position()
    (msl.equipment.resources.thorlabs.kinesis.kcube_solenoidKCubesSolenoidrces.thorlabs.kinesis.integrated_step_
    method), 535
    method), 495
request_mmi_params() request_position()
    (msl.equipment.resources.thorlabs.kinesis.kcube_steppos_equipotentialKCubesStoppersMotabs.kinesis.kcube_dc_servo_
    method), 552
    method), 518
request_move_absolute_position() request_position()
    (msl.equipment.resources.thorlabs.kinesis.benchtop_<del>step</del>equipotentialBenchtopStepperMotines.kinesis.kcube_stepper_
    method), 418
    method), 552
request_move_absolute_position() request_potentiometer_params()
    (msl.equipment.resources.thorlabs.kinesis.integrated_<del>step</del>pqniotats:inagatedSothp:Motoris.integrated_step_
    method), 495
    method), 495
request_move_absolute_position() request_power_params()
    (msl.equipment.resources.thorlabs.kinesis.kcube_dc_<del>KCubeSolenoid</del>G8sources.thorlabs.kinesis.benchtop_stepper_
    method), 518
    method), 419
request_move_absolute_position() request_power_params()
    (msl.equipment.resources.thorlabs.kinesis.kcube_steppos_equipotentialKCubesStoppersMotabs.kinesis.integrated_step_
    method), 552
    method), 495
request_move_relative_distance() request_power_params()
    (msl.equipment.resources.thorlabs.kinesis.benchtop_<del>step</del>equipotentialBenchtopStepperMotines.kinesis.kcube_stepper_
    method), 418
    method), 553
request_move_relative_distance() request_rack_digital_outputs()
    (msl.equipment.resources.thorlabs.kinesis.integrated_<del>step</del>pqniotats:inagatedSothp:Motoris.benchtop_stepp_
    method), 495
    method), 419
request_move_relative_distance() request_rack_status_bits()
    (msl.equipment.resources.thorlabs.kinesis.kcube_dc_<del>KCubeSolenoid</del>G8sources.thorlabs.kinesis.benchtop_stepp_
    method), 518
    method), 419
request_move_relative_distance() request_settings()

```

(*msl.equipment.resources.thorlabs.kinesis.benchtop_stepper*.~~step~~*EquipmentBenchtopStep*.~~method~~*kinesis.benchtop_step*
method), 419
request_settings() (*msl.equipment.resources.thorlabs.kinesis.filter_flipper*.~~flipper~~*EquipmentFilter*.~~flipper~~*resources.thorlabs.kinesis.integrated_step*
method), 482
request_settings() (*msl.equipment.resources.thorlabs.kinesis.integrated_strips*.~~strips~~*EquipmentIntegratedStep*.~~strips~~*Motor*.*benchtop_step*
method), 495
request_settings() (*msl.equipment.resources.thorlabs.kinesis.kcube_dc*.~~dc~~*EquipmentKCubeDC*.~~dc~~*sources.thorlabs.kinesis.integrated_step*
method), 519
request_settings() (*msl.equipment.resources.thorlabs.kinesis.kcube_solenoid*.~~solenoid~~*EquipmentKCubeSolenoid*.*resources.thorlabs.kinesis.kcube_dc_servo*
method), 535
request_settings() (*msl.equipment.resources.thorlabs.kinesis.kcube_step*.~~step~~*EquipmentKCubeStep*.~~step~~*Motor*.*thorlabs.kinesis.kcube_stepper*
method), 553
request_status() (*msl.equipment.resources.thorlabs.kinesis.filter_flipper*.~~flipper~~*EquipmentFilter*.~~flipper~~*status*
method), 481
request_status() (*msl.equipment.resources.thorlabs.kinesis.integrated_step*.~~step~~*EquipmentIntegratedStep*.~~step~~*status*
method), 496
request_status() (*msl.equipment.resources.thorlabs.kinesis.kcube_dc*.~~dc~~*EquipmentKCubeDC*.~~dc~~*status*
method), 519
request_status() (*msl.equipment.resources.thorlabs.kinesis.kcube_solenoid*.~~solenoid~~*EquipmentKCubeSolenoid*.~~solenoid~~
method), 535
request_status_bits() (*msl.equipment.resources.thorlabs.kinesis.broadcast*.~~broadcast~~*EquipmentBroadcast*.~~broadcast~~*StepperMotorMaxAvantSpec*.*Avantes.Broadcas*
method), 420
request_status_bits() (*msl.equipment.resources.thorlabs.kinesis.integrated_step*.~~step~~*EquipmentIntegratedStep*.~~step~~*status*
method), 496
request_status_bits() (*msl.equipment.resources.thorlabs.kinesis.kcube_dc*.~~dc~~*EquipmentKCubeDC*.~~dc~~*solenoid*
method), 519
request_status_bits() (*msl.equipment.resources.thorlabs.kinesis.kcube_solenoid*.~~solenoid~~*EquipmentKCubeSolenoid*.~~solenoid~~
method), 535
request_status_bits() (*msl.equipment.resources.thorlabs.kinesis.kcube_dc*.~~dc~~*EquipmentKCubeDC*.~~dc~~*status*
method), 553
request_trigger_config_params() (*msl.equipment.resources.thorlabs.kinesis.kcube_dc*.~~dc~~*EquipmentKCubeDC*.~~dc~~*servo*
method), 519
request_trigger_config_params() (*msl.equipment.resources.thorlabs.kinesis.kcube_dc*.~~dc~~*EquipmentKCubeDC*.~~dc~~*servo*
method), 536
request_trigger_config_params() (*msl.equipment.resources.thorlabs.kinesis.kcube_dc*.~~dc~~*EquipmentKCubeDC*.~~dc~~*servo*
method), 553
request_trigger_switches() (*msl.equipment.resources.thorlabs.kinesis.kcube_dc*.~~dc~~*EquipmentKCubeDC*.~~dc~~*switches*
method), 592

reserved(msl.equipment.resources.thorlabs.kinesis.structs.KSGLMMParentresources.optronic_laboratories.ol756ocx
attribute), 592
method), 219

reserved(msl.equipment.resources.thorlabs.kinesis.resets_KevideTriggerConfig
attribute), 593
(msl.equipment.resources.avantes.avaspec.Avantes

reserved0(msl.equipment.resources.picotech.picoscope.structure_PSD000ATriggerInfo
attribute), 364
reset_rotation_modes()
reserved0(msl.equipment.resources.picotech.picoscope.structure_P8q000ATriggInfo
attribute), 367
method), 420

reserved1(msl.equipment.resources.picotech.picoscope.structure_PSD1000ATriggInfo
attribute), 364
(msl.equipment.resources.thorlabs.kinesis.integrated_step
reset_rotation_modes())

reserved1(msl.equipment.resources.picotech.picoscope.structure_PSD1000ATriggerInfo
attribute), 368
reset_rotation_modes()

reserved1(msl.equipment.resources.thorlabs.kinesis.structure_F110QStepper
attribute), 582
method), 519

reserved1(msl.equipment.resources.thorlabs.kinesis.resets_K1DITrigHoles)
attribute), 585
(msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo
reset_stage_to_defaults())

reserved1(msl.equipment.resources.thorlabs.kinesis.structure_K1DTrigIOParams
attribute), 585
reset_stage_to_defaults()

reserved1(msl.equipment.resources.thorlabs.kinesis.structure_M0TqgqAnInstrument
attribute), 573
method), 496

reserved1(msl.equipment.resources.thorlabs.kinesis.resets_PDC_IOS_defaults
attribute), 581
(msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo
reset_stage_to_defaults())

reserved2(msl.equipment.resources.thorlabs.kinesis.structure_FETH0Secting
attribute), 582
reset_status()

reserved2(msl.equipment.resources.thorlabs.kinesis.structure_K1DDefining
attribute), 585
method), 157

reserved2(msl.equipment.resources.thorlabs.kinesis.resets_K1STrigIOEquipment
attribute), 585
method), 94

reserved2(msl.equipment.resources.thorlabs.kinesis.structure_RESISTANCE_100KageAxisParameters
attribute), 573
(msl.equipment.resources.picotech.picoscope.enums.PS4000b.
RESISTANCE_100R)

reserved3(msl.equipment.resources.thorlabs.kinesis.structure_M0TBUStageAxisParameters
attribute), 573
RESISTANCE_100R

reserved4(msl.equipment.resources.thorlabs.kinesis.structure_M0TqgqAnInstrument
attribute), 573
picotech.picoscope.enums.PS4000b.
attribute), 271

reserved5(msl.equipment.resources.thorlabs.kinesis.structure_RESISTANCE_10KageAxisParameters
attribute), 573
(msl.equipment.resources.picotech.picoscope.enums.PS4000b.
RESISTANCE_10K)

reserved6(msl.equipment.resources.thorlabs.kinesis.structure_M0TBUStageAxisParameters
attribute), 573
RESISTANCE_1K(msl.equipment.resources.picotech.picoscope.enums.PS4000b.
RESISTANCE_10K)

reserved7(msl.equipment.resources.thorlabs.kinesis.structure_M0TBUStageAxisParameters
attribute), 573
RESISTANCE_1M(msl.equipment.resources.picotech.picoscope.enums.PS4000b.
RESISTANCE_10M)

reserved8(msl.equipment.resources.thorlabs.kinesis.structure_M0TBUStageAxisParameters
attribute), 573
RESISTANCE_25K

reset()(msl.equipment.resources.aim_tti.mx_series.MXSeriemsl.equipment.resources.picotech.picoscope.enums.PS4000b.
method), 94
attribute), 272

reset()(msl.equipment.resources.energetiq.eq99.ERESISTANCE_50K
method), 142
(msl.equipment.resources.picotech.picoscope.enums.PS4000b.
RESISTANCE_50K)

reset()(msl.equipment.resources.omega.ithx.iTHX
method), 197
attribute), 272

reset()(msl.equipment.resources.optronic_laboratories.ol_attributes)
method), 226
RESISTANCE_TO_10K
(msl.equipment.resources.picotech.pt104.Pt104DataType
RESISTANCE_TO_10K)

reset_averaging()

attribute), 371
RESISTANCE_TO_375R RISING (msl.equipment.resources.picotech.picoscope.enums.PS3000Attribute), 264
(msl.equipment.resources.picotech.pt104.Pt104DataType), 266
attribute), 371 RISING (msl.equipment.resources.picotech.picoscope.enums.PS3000Attribute), 264
RESISTANCES (msl.equipment.resources.picotech.picoscope.enums.PS4000AChannelInfo attribute), 292 RISING (msl.equipment.resources.picotech.picoscope.enums.PS3000Attribute), 264
RESISTANCES (msl.equipment.resources.picotech.picoscope.enums.PS4000AChannelInfo attribute), 273 RISING (msl.equipment.resources.picotech.picoscope.enums.PS4000Attribute), 264
resolution() (msl.equipment.resources.greisinger.gmh3000.GMH3000Q87 method), 149 RISING (msl.equipment.resources.picotech.picoscope.enums.PS4000Attribute), 264
resource (msl.equipment.connection_pyvisa.ConnectionPyVISAAttribute), 289
property), 37 RISING (msl.equipment.resources.picotech.picoscope.enums.PS4000Attribute), 264
resource_class() RISING (msl.equipment.resources.picotech.picoscope.enums.PS4000Attribute), 275
(msl.equipment.connection_pyvisa.ConnectionPyVISAAttribute), 289
static method), 38 RISING (msl.equipment.resources.picotech.picoscope.enums.PS4000Attribute), 277
resource_manager() RISING (msl.equipment.resources.picotech.picoscope.enums.PS5000Attribute), 309
(msl.equipment.connection_pyvisa.ConnectionPyVISAAttribute), 309
static method), 37 RISING (msl.equipment.resources.picotech.picoscope.enums.PS5000Attribute), 309
ResourceClassNotFound, 54 RISING (msl.equipment.resources.picotech.picoscope.enums.PS5000Attribute), 311
restPercentage RISING (msl.equipment.resources.picotech.picoscope.enums.PS5000Attribute), 311
(msl.equipment.resources.thorlabs.kinesis.structs.MQFtrFlow), 200
attribute), 576 RISING (msl.equipment.resources.picotech.picoscope.enums.PS5000Attribute), 319
resume_move_messages() RISING (msl.equipment.resources.thorlabs.kinesis.bspEquipmentResult), 319
(msl.equipment.resources.thorlabs.kinesis.bspEquipmentResult), 319
method), 420 RISING (msl.equipment.resources.picotech.picoscope.enums.PS6000Attribute), 319
resume_move_messages() RISING (msl.equipment.resources.picotech.picoscope.enums.PS6000Attribute), 319
(msl.equipment.resources.thorlabs.kinesis.kcube_dc_attributes), 200
method), 519 RISING_LOWER (msl.equipment.resources.picotech.picoscope.enums.PS6000Attribute), 319
resume_move_messages() RISING (msl.equipment.resources.thorlabs.kinesis.kcube_dc_attributes), 200
(msl.equipment.resources.thorlabs.kinesis.kcube_dc_attributes), 200
method), 553 RISING_LOWER (msl.equipment.resources.picotech.picoscope.enums.PS6000Attribute), 319
Reverse (msl.equipment.resources.thorlabs.kinesis.eris), 289
attribute), 453 RISING_LOWER (msl.equipment.resources.picotech.picoscope.enums.PS6000Attribute), 289
Reverse (msl.equipment.resources.thorlabs.kinesis.eris), 277
attribute), 438 RISING_LOWER (msl.equipment.resources.picotech.picoscope.enums.PS6000Attribute), 277
Reverse (msl.equipment.resources.thorlabs.kinesis.eris), 311
attribute), 475 RISING_LOWER (msl.equipment.resources.picotech.picoscope.enums.PS6000Attribute), 311
rightButtonPosition RISING_LOWER (msl.equipment.resources.picotech.picoscope.enums.PS6000Attribute), 311
(msl.equipment.resources.thorlabs.kinesis.structs.MQftrButton), 200
attribute), 582 RISING_OR_FALLING
RISING (msl.equipment.resources.picotech.picoscope.enums.PS2000DigitalDirections), 249
attribute), 249 RISING_OR_FALLING (msl.equipment.resources.picotech.picoscope.enums.PS2000DigitalDirections), 249
RISING (msl.equipment.resources.picotech.picoscope.enums.PS2000GenTrigType) RISING_OR_FALLING (msl.equipment.resources.picotech.picoscope.enums.PS2000GenTrigType), 249
attribute), 247 RISING_OR_FALLING (msl.equipment.resources.picotech.picoscope.enums.PS2000GenTrigType), 249
RISING (msl.equipment.resources.picotech.picoscope.enums.PS2000Hold), 249
attribute), 249 RISING_OR_FALLING (msl.equipment.resources.picotech.picoscope.enums.PS2000Hold), 249
RISING (msl.equipment.resources.picotech.picoscope.enums.PS2000TriggerDirections), 238
attribute), 235 RISING_OR_FALLING (msl.equipment.resources.picotech.picoscope.enums.PS2000TriggerDirections), 238
RISING (msl.equipment.resources.picotech.picoscope.enums.PS3000SigGenDirection) RISING_OR_FALLING (msl.equipment.resources.picotech.picoscope.enums.PS3000SigGenDirection), 267
attribute), 267 RISING_OR_FALLING (msl.equipment.resources.picotech.picoscope.enums.PS3000SigGenDirection), 267
RISING (msl.equipment.resources.picotech.picoscope.enums.PS3000SigGenTrigType)

RISING_OR_FALLING
`(msl.equipment.resources.picotech.picoscope.enums.PS3000AThresholdDirection attribute), 266`
 method), 330

RISING_OR_FALLING
`(msl.equipment.resources.picotech.picoscope.enums.PS3000ThresholdDirection attribute), 256`

RISING_OR_FALLING
`(msl.equipment.resources.picotech.picoscope.enums.PS4000AThresholdDirection attribute), 289`
`(msl.equipment.resources.picotech.picoscope.enums.PS4000ThresholdDirection attribute), 245`

RISING_OR_FALLING
`(msl.equipment.resources.picotech.picoscope.enums.PS4000ThresholdDirection attribute), 277`
`(msl.equipment.resources.picotech.picoscope.enums.PS3000ATime attribute), 263`

RISING_OR_FALLING
`(msl.equipment.resources.picotech.picoscope.enums.PS5000AThresholdDirection attribute), 311`
`(msl.equipment.resources.picotech.picoscope.enums.PS5000ATime attribute), 253`

RISING_OR_FALLING
`(msl.equipment.resources.picotech.picoscope.enums.PS5000ThresholdDirection attribute), 302`
`(msl.equipment.resources.picotech.picoscope.enums.PS4000ATime attribute), 274`

RISING_OR_FALLING
`(msl.equipment.resources.picotech.picoscope.enums.PS6000ThresholdDirection attribute), 321`
`(msl.equipment.resources.picotech.picoscope.enums.PS5000ATime attribute), 308`

rmt (msl.equipment.hislip.SyncClient property), S (msl.equipment.resources.picotech.picoscope.enums.PS5000Time attribute), 299

76

root (msl.equipment.config.Config property), 19 S (msl.equipment.resources.picotech.picoscope.enums.PS6000Time attribute), 299

ROOT_NAME_LEN (msl.equipment.resources.avantes.avaspec.AvAttribute), 317

attribute), 106

SAMPLE_PROPERTIES
`(msl.equipment.resources.picotech.picoscope.enums.PS4000ATime attribute), 294`

RotationalUnlimited
`(msl.equipment.resources.thorlabs.kinesis.enums.MOT_MovementModes attribute), 437`

SAMPLE_PROPERTIES_NO_OF_CAPTURES
`(msl.equipment.resources.picotech.picoscope.enums.PS4000ATime attribute), 294`

RotationalWrapping
`(msl.equipment.resources.thorlabs.kinesis.enums.MOT_MovementModes attribute), 437`

SAMPLE_PROPERTIES_OVERLAPPED
`(msl.equipment.resources.picotech.picoscope.enums.PS4000ATime attribute), 295`

RPC_MISMATCH (msl.equipment.vxi11.RejectStatus attribute), 602

SAMPLE_PROPERTIES_OVERLAPPED_DOWN_SAMPLE_RATIO
`(msl.equipment.resources.picotech.picoscope.enums.PS4000ATime attribute), 295`

RS232 (msl.equipment.resources.avantes.avaspec.Avantes.InterfaceType attribute), 108

SAMPLE_PROPERTIES_OVERLAPPED_DOWN_SAMPLE_RATIO_MODE
`(msl.equipment.resources.picotech.picoscope.enums.PS4000ATime attribute), 295`

RS232 (msl.equipment.resources.avantes.avaspec.InterfaceType attribute), 98

SAMPLE_PROPERTIES_OVERLAPPED_MESSAGE_BASED
`(msl.equipment.resources.picotech.picoscope.enums.PS4000ATime attribute), 295`

rstrip (msl.equipment.connection_message_based.ConnectionMessageBased property), 30

SAMPLE_PROPERTIES_OVERLAPPED_REQUESTED_NO_OF_SAMPLES
`(msl.equipment.resources.picotech.picoscope.enums.PS4000ATime attribute), 295`

rstrip (msl.equipment.connection_prologix.ConnectionPrologix property), 34

SAMPLE_PROPERTIES_OVERLAPPED_SEGMENT_INDEX_FROM
`(msl.equipment.resources.picotech.picoscope.enums.PS4000ATime attribute), 295`

run_block() (msl.equipment.resources.picotech.picoscope.picoscopeBlock attribute), 327

SAMPLE_PROPERTIES_OVERLAPPED_SEGMENT_INDEX_TO
`(msl.equipment.resources.picotech.picoscope.enums.PS4000ATime attribute), 295`

run_streaming()
`(msl.equipment.resources.picotech.picoscope.picoscopeBlock attribute), 327`

SAMPLE_PROPERTIES_POST_TRIGGER_SAMPLES
`(msl.equipment.resources.picotech.picoscope.enums.PS4000ATime attribute), 295`

run_streaming_ex()
`(msl.equipment.resources.picotech.picoscope.picoscopeBlock attribute), 351`

S

attribute), 295
SAMPLE_PROPERTIES_PRE_TRIGGER_SAMPLES SC_SolenoidClosed
(*msl.equipment.resources.picotech.picoscope.enums.PS4000APinStringValues.thorlabs.kinesis.enums.SC_Sole*
attribute), 295
attribute), 470
SAMPLE_PROPERTIES_RESOLUTION SC_SolenoidOpen
(*msl.equipment.resources.picotech.picoscope.enums.PS4000APinStringValues.thorlabs.kinesis.enums.SC_Sole*
attribute), 295
attribute), 471
SAMPLE_PROPERTIES_TIMEBASE SC_SolenoidStates (class in
(*msl.equipment.resources.picotech.picoscope.enums.PS4000APinStringValues.thorlabs.kinesis.enums*),
attribute), 295
471
samplesPerRevolution SC_Triggered (*msl.equipment.resources.thorlabs.kinesis.enums.S*
(*msl.equipment.resources.thorlabs.kinesis.structs.NTAttributParameters*
attribute), 577
attribute), 470
SC_Unknown (*msl.equipment.resources.thorlabs.kinesis.enums.SC_*
attribute), 471
SAT_PEAK_INVERSION attribute), 471
(*msl.equipment.resources.avantes.avaspec.Scales(msl.equipment.connection_nidaq.ConnectionNIDAQ*
attribute), 106
property), 32
save() (*msl.equipment.resources.aim_tti.mx_series.MXSeries*).
correction()
method), 95
(*msl.equipment.resources.greisinger.gmh3000.GMH3000*
method), 149
save() (*msl.equipment.resources.thorlabs.fwxx2c.FilterWheelXX2c*), 596
SCOPE_TRIG (*msl.equipment.resources.picotech.picoscope.enums.P*
method), 596
save_all() (*msl.equipment.resources.aim_tti.mx_series.MXSeries*), 247
method), 95
SCOPE_TRIG (*msl.equipment.resources.picotech.picoscope.enums.P*
attribute), 265
save_calibration_file()
(*msl.equipment.resources.optronic_laboratorys.OPT1000*).
SCOPE_TRIG (*msl.equipment.resources.picotech.picoscope.enums.P*
method), 219
attribute), 288
save_measurement_data()
(*msl.equipment.resources.optronic_laboratories.ol7560cibud2*).
SCOPE_TRIG (*msl.equipment.resources.picotech.picoscope.enums.P*
method), 219
SCOPE_TRIG (*msl.equipment.resources.picotech.picoscope.enums.H*
method), 129
save_setup() (*msl.equipment.resources.bentham.benhw32.Bentham*), 2310
method), 129
SCOPE_TRIG (*msl.equipment.resources.picotech.picoscope.enums.H*
attribute), 300
save_streaming_data()
(*msl.equipment.resources.picotech.picoscope.ScopeStreaming*).
SCOPE_TRIG (*msl.equipment.resources.picotech.picoscope.enums.H*
method), 346
attribute), 320
SC_Active (*msl.equipment.resources.thorlabs.kinesis.Sdk(msl.equipment.SdkOpeneratingStation_sdk.ConnectionSDK*
attribute), 471
property), 39
SC_Auto (*msl.equipment.resources.thorlabs.kinesis.Sdk(msl.equipment.SdkOpeneratingModes.Interface* attribute),
attribute), 470
50
SC_CycleParameters (class in *search_for_inst()*
msl.equipment.resources.thorlabs.kinesis.structs), (msl.equipment.resources.princeton_instruments.arc_instr
591
static method), 379
SC_Inactive (*msl.equipment.resources.thorlabs.kinesis.Sdk(msl.equipment.SdkOpeneratingStates.nktpdll.DateTimeType*
attribute), 471
attribute), 162
SC_Manual (*msl.equipment.resources.thorlabs.kinesis.Sdk(msl.equipment.SdkOpeneratingModes*
attribute), 470
(*msl.equipment.connection_gpib.ConnectionGPIB*
secondary_address), 470
attribute), 27
SC_OperatingModes (class in *SECURE_CONNECTION_SUPPORTED*
msl.equipment.resources.thorlabs.kinesis.enums), (msl.equipment.hislip.AsyncInitializeResponse
470
attribute), 68
SC_OperatingStates (class in *SECURE_CONNECTION_SUPPORTED*
msl.equipment.resources.thorlabs.kinesis.enums), (msl.equipment.hislip.AsyncInitializeResponse
470
attribute), 68
SC_Single (*msl.equipment.resources.thorlabs.kinesis.enums.ScOpeneratingModes*

<code>segmentIndex(msl.equipment.resources.picotech.picoscope.smktxqR\$3000AtriggeredByAvantes.avaspec.Avantes.SensType attribute), 364</code>	<code>attribute), 109</code>
<code>segmentIndex(msl.equipment.resources.picotech.picoscope.SENS_HAMS11071_2048X64triggerInfo attribute), 368</code>	<code>(msl.equipment.resources.avantes.avaspec.SensType attribute), 98</code>
<code>select_lamp()(msl.equipment.resources.optronic_laboratory.OLCurrentSource method), 226</code>	<code>SENS_HAMS11155_2048</code>
<code>select_wavelength()(msl.equipment.resources.bentham.benhw32.Bentham attribute), 129</code>	<code>(msl.equipment.resources.avantes.avaspec.Avantes.SensType attribute), 109</code>
<code>select_wavelength()(msl.equipment.resources.bentham.benhw64.Bentham attribute), 131</code>	<code>SENS_HAMS11155_2048</code>
<code>selected_addresses</code>	<code>(msl.equipment.resources.avantes.avaspec.Avantes.SensType attribute), 109</code>
<code>(msl.equipment.connection_prologix.ConnectionPrologix attribute), 34</code>	<code>SENS_HAMS11638</code>
<code>selectedRange(msl.equipment.resources.thorlabs.kinesis.structure.KNpfdAReadinges.avantes.avaspec.SensType attribute), 586</code>	<code>attribute), 99</code>
<code>selectedRange(msl.equipment.resources.thorlabs.KNpfdAReading attribute), 578</code>	<code>(msl.equipment.resources.avantes.avaspec.Avantes.SensType attribute), 99</code>
<code>send()(msl.equipment.hislip.SyncClient method), 76</code>	<code>SENS_HAMS11639</code>
<code>send()(msl.equipment.resources.bentham.benhw32.Bentham method), 129</code>	<code>attribute), 109</code>
<code>send_down_parameters()</code>	<code>SENS_HAMS12443</code>
<code>(msl.equipment.resources.optronic_laboratories.ol756onthequipH6resources.avantes.avaspec.Avantes.SensType method), 219</code>	<code>attribute), 109</code>
<code>SENS_HAMG13913</code>	<code>SENS_HAMS12443</code>
<code>(msl.equipment.resources.avantes.avaspec.Avantes.SensType equipment.resources.avantes.avaspec.SensType attribute), 109</code>	<code>attribute), 99</code>
<code>SENS_HAMG13913</code>	<code>SENS_HAMS13496</code>
<code>(msl.equipment.resources.avantes.avaspec.SensType attribute), 99</code>	<code>(msl.equipment.resources.avantes.avaspec.Avantes.SensType attribute), 109</code>
<code>SENS_HAMG9208_512</code>	<code>SENS_HAMS13496</code>
<code>(msl.equipment.resources.avantes.avaspec.SensType attribute), 109</code>	<code>(msl.equipment.resources.avantes.avaspec.SensType attribute), 99</code>
<code>SENS_HAMG9208_512</code>	<code>SENS_HAMS7031_1024X122</code>
<code>(msl.equipment.resources.avantes.avaspec.SensType attribute), 99</code>	<code>(msl.equipment.resources.avantes.avaspec.Avantes.SensType attribute), 109</code>
<code>SENS_HAMS10420_2048X64</code>	<code>SENS_HAMS7031_1024X122</code>
<code>(msl.equipment.resources.avantes.avaspec.Avantes.SensType attribute), 109</code>	<code>(msl.equipment.resources.avantes.avaspec.SensType attribute), 99</code>
<code>SENS_HAMS10420_2048X64</code>	<code>SENS_HAMS7031_1024X58</code>
<code>(msl.equipment.resources.avantes.avaspec.SensType attribute), 98</code>	<code>(msl.equipment.resources.avantes.avaspec.Avantes.SensType attribute), 109</code>
<code>SENS_HAMS11071_2048X16</code>	<code>SENS_HAMS7031_1024X58</code>
<code>(msl.equipment.resources.avantes.avaspec.SensType attribute), 109</code>	<code>(msl.equipment.resources.avantes.avaspec.SensType attribute), 99</code>
<code>SENS_HAMS11071_2048X16</code>	<code>SENS_HAMS8378_1024</code>
<code>(msl.equipment.resources.avantes.avaspec.SensType attribute), 99</code>	<code>(msl.equipment.resources.avantes.avaspec.Avantes.SensType attribute), 109</code>
<code>SENS_HAMS11071_2048X64</code>	<code>SENS_HAMS8378_1024</code>

(*msl.equipment.resources.avantes.avaspec.SensType* (*msl.equipment.resources.picotech.picoscope.enums.PS400Attribute*), 98
attribute), 290

SENS_HAMS8378_256 *SensorMode* (class in
(msl.equipment.resources.avantes.avaspec.Avantes.SensType *equipment.resources.thorlabs.fwxx2c*),
attribute), 108
SENS_HAMS8378_256 *SensType* (class in
(msl.equipment.resources.avantes.avaspec.SensType *msl.equipment.resources.avantes.avaspec*),
attribute), 98
SENS_HAMS8378_512 *serial* (*msl.equipment.connection_serial.ConnectionSerial*
(msl.equipment.resources.avantes.avaspec.Avantes.SensType), 40
attribute), 109
SENS_HAMS8378_512 *SERIAL* (*msl.equipment.constants.Interface*
attribute), 50
(msl.equipment.resources.avantes.avaspec.SensType (*msl.equipment.record_types.ConnectionRecord*
attribute), 98
attribute), 86

SENS_HAMS9201 (*msl.equipment.resources.avantes.avaspec.SensType* (*msl.equipment.record_types.EquipmentRecord*
attribute), 109
attribute), 80

SENS_HAMS9201 (*msl.equipment.resources.avantes.avaspec.SensType* (*msl.equipment.resources.avantes.avaspec.Avantes.BroadcastAnswer*),
attribute), 98
attribute), 110

SENS_HAMS9840 (*msl.equipment.resources.avantes.avaspec.SensType* (*msl.equipment.resources.avantes.avaspec.BroadcastAnswer*),
attribute), 109
attribute), 100

SENS_HAMS9840 (*msl.equipment.resources.avantes.avaspec.SensType* (*msl.equipment.resources.avantes.avaspec.SerialScanner*))
attribute), 98
(msl.equipment.resources.energetiq.eq99.EQ99
SENS_ILX511 (*msl.equipment.resources.avantes.avaspec.AvantesSerialType*
attribute), 109
SERIAL_NUMBER_BUFFER_SIZE

SENS_ILX511 (*msl.equipment.resources.avantes.avaspec.SensType* (*msl.equipment.resources.thorlabs.kinesis.motion_control*),
attribute), 98
attribute), 569

SENS_ILX554 (*msl.equipment.resources.avantes.avaspec.SensType* (*msl.equipment.connection_gpib.ConnectionGPII*),
attribute), 109
method), 27

SENS_ILX554 (*msl.equipment.resources.avantes.avaspec.SensType* (*msl.equipment.resources.thorlabs.kinesis structs.TLI_Device*),
attribute), 98
attribute), 571

SENS_SU256LSB (*msl.equipment.resources.avantes.avaspec.SensType* (*msl.equipment.resources.avantes.avaspec.Avantes*),
attribute), 109
attribute), 109

SENS_SU256LSB (*msl.equipment.resources.avantes.avaspec.SensType* (*msl.equipment.resources.avantes.avaspec.AvsIdent*),
attribute), 99
attribute), 99

SENS_SU512LDB (*msl.equipment.resources.avantes.avaspec.SensType* (*msl.equipment.resources.thorlabs.kinesis structs.M*),
attribute), 109
attribute), 573

SENS_SU512LDB (*msl.equipment.resources.avantes.avaspec.SensType* (*msl.equipment.resources.thorlabs.kinesis structs.T*),
attribute), 99
attribute), 571

SENS_TCD1304 (*msl.equipment.resources.avantes.avaspec.SensType* (*msl.equipment.hislip.AsyncInitializeResponse*),
attribute), 109
(msl.equipment.hislip.AsyncInitializeResponse)

SENS_TCD1304 (*msl.equipment.resources.avantes.avaspec.SensType*), 68
attribute), 98
session_id (*msl.equipment.hislip.InitializeResponse*)

SENS_TSL1301 (*msl.equipment.resources.avantes.avaspec.AvanteSerialType*
attribute), 109
set() (*msl.equipment.resources.bentham.benhw32.Bentham32*)

SENS_TSL1301 (*msl.equipment.resources.avantes.avaspec.SensType*), 129
attribute), 98
set() (*msl.equipment.resources.bentham.benhw64.Bentham*)

SENS_TSL1401 (*msl.equipment.resources.avantes.avaspec.AvanteSerialType*
attribute), 109
set_acceleration()

SENS_TSL1401 (*msl.equipment.resources.avantes.avaspec.SensType* (*msl.equipment.resources.thorlabs.fwxx2c.FilterWheelXX*)),
attribute), 98
method), 596

SENSOR_STATE_CONNECTED *set_access_channel()*

```

(msl.equipment.resources.mks_instruments.pr4000b.PRS4000Bpmnt.resources.thorlabs.kinesis.integrated_step_
method), 157
set_actual_value() set_bow_index()
(msl.equipment.resources.mks_instruments.pr4000b.PRS4000Bpmnt.resources.thorlabs.kinesis.kcube_stepper_
method), 157
set_adaptive_integration_time() set_brightness()
(msl.equipment.resources.optronic_laboratories.ol756nskeq2ipOL756resources.energetiq.eq99.EQ99
method), 220
set_address() set_bump_4000a_PR4000B
(msl.equipment.resources.mks_instruments.pr4000b.PRS4000Bpmnt.resources.thorlabs.kinesis.integrated_step_
method), 158
set_adv_trigger_channel_conditions() set_button_2pk_PicoScope2k3k
(msl.equipment.resources.picotech.picoscope.set_button_2pk_PicoScope2k3k
method), 331
set_adv_trigger_channel_directions() set_calibrator_file()
(msl.equipment.resources.picotech.picoscope.set_calibrator_file_PicoScope2k3k
method), 330
set_adv_trigger_channel_properties() set_callback_file()
(msl.equipment.resources.picotech.picoscope.set_callback_file_PicoScope2k3k
method), 331
set_adv_trigger_delay() set_calibration_file()
(msl.equipment.resources.picotech.picoscope.set_calibration_file_PicoScope2k3k
method), 330
set_alarm() set_calibration_file()
(msl.equipment.resources.electron_dynamics.tcmehtod), TCSeries
method), 137
set_analog_out() set_calibration_file()
(msl.equipment.resources.avantes.avaspec.Avantes method), 554
method), 123
set_averaging_number_of_scan() set_callback_device_status()
(msl.equipment.resources.optronic_laboratories.ol756static_B2100V5693
method), 220
set_backlash() set_callback_port_status()
(msl.equipment.resources.nkt.nktpdll.NKT
method), 420
set_backlash() set_callback_register_status()
(msl.equipment.resources.nkt.nktpdll.NKT
method), 420
set_backlash() set_channel()
(msl.equipment.resources.thorlabs.kinesis.integrated_stappmethod), IntegratedStepperMotors
method), 496
set_backlash() set_channel()
(msl.equipment.resources.thorlabs.kinesis.kcube_stappmethod), KCubeStoppersMototech.picoscope.picoo
method), 519
set_bandwidth_filter() set_channel_led()
(msl.equipment.resources.thorlabs.kinesis.kcube_step(psl.equipment.KCubeStoppersMototech.picoscope.ps4000a.Pic
method), 553
set_bandwidth_filter() set_control()
(msl.equipment.resources.picotech.picoscope.picoscope.set(picotech.picoscope.PicScopeApi
method), 336
set_beep() set_current()
(msl.equipment.resources.energetiq.eq99.EQ99 method), 226
method), 142
set_bow_index() set_current_limit()
(msl.equipment.resources.aim_tti.mx_series.MXSeries
method), 421
set_bow_index() set_current_meter_averaging()
(msl.equipment.resources.aim_tti.mx_series.MXSeries
method), 421

```

method), 95
set_current_step_size()
(*msl.equipment.resources.aim_tti.mx_series.MXSeries*).
method), 95
set_cycle_params()
(*msl.equipment.resources.thorlabs.kinesis.kcube_solenoid.KCubeSolenoidResources*).
method), 536
set_cycle_params_block()
(*msl.equipment.resources.thorlabs.kinesis.kcube_solenoid.KCubeSolenoidResources*).
method), 536
set_dark_current_params()
(*msl.equipment.resources.optronic_laboratories.ol756nsl.equipment.resources.princeton_instruments.arc_instrument*).
method), 220
set_data_buffer()
(*msl.equipment.resources.picotech.picoscope.picoscope.PicoScopeApplies*).
method), 336
set_data_buffer_bulk()
(*msl.equipment.resources.picotech.picoscope.picoscope.PicoScopeApplies*).
method), 337
set_data_buffer_with_mode()
(*msl.equipment.resources.picotech.picoscope.ps4000(PicoScope4000Resources*).
method), 351
set_data_buffers()
(*msl.equipment.resources.picotech.picoscope.picoscope.PicoScopeApi*).
method), 337
set_data_buffers_bulk()
(*msl.equipment.resources.picotech.picoscope.ps2000a.PicoScope2000*).
method), 357
set_data_buffers_with_mode()
(*msl.equipment.resources.picotech.picoscope.ps4000(PicoScope4000*).
method), 351
set_dcpid_params()
(*msl.equipment.resources.thorlabs.kinesis.benchtop_stepper*).
method), 520
set_dead_band()
(*msl.equipment.resources.mks_instruments.pr4000b.PR4000B*).
method), 158
set_det_bipolar()
(*msl.equipment.resources.princeton_instruments.arc_instrument.KCubeDCServo*).
method), 394
set_det_current()
(*msl.equipment.resources.princeton_instruments.arc_instrument.KCubePrincetonInstruments*).
method), 394
set_det_hv_off()
(*msl.equipment.resources.princeton_instruments.arc_instrument.KCubePrincetonInstruments*).
method), 395
set_det_hv_on()
(*msl.equipment.resources.princeton_instruments.arc_instrument.KCubePrincetonInstruments*).
method), 395
set_det_hv_volts()
(*msl.equipment.resources.princeton_instruments.arc_instrument.KCubePrincetonInstruments*)

method), 395
set_det_num_avg_read()
method), 395
set_det_photon()
method), 395
set_det_range()
method), 395
set_det_type()
method), 396
set_det_unipolar()
method), 396
set_det_voltage()
method), 396
set_device_resolution()
method), 356
set_dialog()
(*msl.equipment.resources.mks_instruments.pr4000(PicoScope4000*).
method), 356
set_digital_analog_trigger_operand()
(*msl.equipment.resources.picotech.picoscope.ps2000a.PicoScope2000*).
method), 357
set_digital_out()
(*msl.equipment.resources.avantes.avaspec.Avantes*).
method), 158
set_digital_outputs()
(*msl.equipment.resources.thorlabs.kinesis.benchtop_stepper*).
method), 520
set_digital_outputs()
(*msl.equipment.resources.thorlabs.kinesis.kcube_dc_server*).
method), 520
set_digital_outputs()
(*msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo*).
method), 520
set_digital_outputs()
(*msl.equipment.resources.thorlabs.kinesis.benchtop_stepper*).
method), 520
set_digital_port()
(*msl.equipment.resources.thorlabs.kinesis.benchtop_stepper*).
method), 520
set_direction()
(*msl.equipment.resources.picotech.picoscope.picoscope_applies*).
method), 395
set_direction()
(*msl.equipment.resources.thorlabs.kinesis.benchtop_stepper*).
method), 520
set_direction()
(*msl.equipment.resources.thorlabs.kinesis.benchtop_stepper*).
method), 520
set_integrated_step()
(*msl.equipment.resources.princeton_instruments.arc_instrument.KCubePrincetonInstruments*)

`set_direction()` *(msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo method)*, 396
`method), 520`

`set_direction()` *(msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo method)*, 158
`method), 554`

`set_display_text()` *(msl.equipment.resources.mks_instruments.pr4000b.PR4000bcounter method)*, 159
`method), 158`

`set_encoder_counter()` *(msl.equipment.resources.thorlabs.kinesis.benchtop_stepper_motor method)*, 422
`method), 520`

`set_encoder_counter()` *(msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo method)*, 520
`method), 554`

`set_encoder_counter()` *(msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo method)*, 555
`method), 159`

`set_ets()` *(msl.equipment.resources.picotech.picoscope.set_gain method)*, 221
`method), 331`

`set_ets()` *(msl.equipment.resources.picotech.picoscope.set_homing_params_block method)*, 422
`method), 337`

`set_ets_time_buffer()` *(msl.equipment.resources.picotech.picoscope.set_homing_params_block method)*, 422
`method), 337`

`set_ets_time_buffers()` *(msl.equipment.resources.picotech.picoscope.set_homing_params_block method)*, 498
`method), 337`

`set_exception_class()` *(msl.equipment.connection.Connection set_homing_params_block method)*, 521
`method), 21`

`set_exposure_mode()` *(msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo set_homing_params_block method)*, 555
`method), 144`

`set_exposure_time()` *(msl.equipment.resources.thorlabs.kinesis.benchtop_stepper_motor set_homing_params_block method)*, 422
`method), 144`

`set_ext_trigger_range()` *(msl.equipment.resources.picotech.picoscope.set_homing_params_block method)*, 498
`method), 351`

`set_external_clock()` *(msl.equipment.resources.picotech.picoscope.set_homing_params_block method)*, 521
`method), 357`

`set_external_input_range()` *(msl.equipment.resources.mks_instruments.pr4000b.PR4000bset_time method)*, 555
`method), 158`

`set_external_output_range()` *(msl.equipment.resources.mks_instruments.pr4000b.PR4000bset_time method)*, 159
`method), 158`

`set_filter_position()` *(msl.equipment.resources.princeton_instruments.set_integration_time method)*, 132
`method), 158`

```

(msl.equipment.resources.optronic_laboratories.ol756onsteqquip0H6resources.thorlabs.kinesis.integrated_step
    method), 221
    method), 499
set_interface_mode() set_jog_vel_params()
    (msl.equipment.resources.mks_instruments.pr4000b.PRA000Bpmment.resources.thorlabs.kinesis.kcube_dc_servo
        method), 159
        method), 522
set_io_settings() set_jog_vel_params()
    (msl.equipment.resources.thorlabs.kinesis.filter_flipperFilequnFipperat.resources.thorlabs.kinesis.kcube_steerer_
        method), 480
        method), 556
set_ip_details() set_joystick_params()
    (msl.equipment.resources.picotech.pt104.PT104   (msl.equipment.resources.thorlabs.kinesis.benchtop_stopp
        method), 373
        method), 423
set_jog_mode() set_lamptime()
    (msl.equipment.resources.thorlabs.kinesis.benchtop_staphequiptonBreschutteSteppregMioter99.EQ99
        method), 422
        method), 144
set_jog_mode() set_led() (msl.equipment.resources.picotech.picoscope.ps2000.P
    (msl.equipment.resources.thorlabs.kinesis.integrated_stappd) mtoors.IntegratedStepperMotors
        method), 498
        set_led_switches()
set_jog_mode() (msl.equipment.resources.thorlabs.kinesis.integrated_step
    (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servoK400DCServo
        method), 521
        set_led_switches())
set_jog_mode() (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo
    (msl.equipment.resources.thorlabs.kinesis.kcube_stepperhod)52KCubeStepperMotor
        method), 555
        set_led_switches())
set_jog_params_block() (msl.equipment.resources.thorlabs.kinesis.kcube_solenoid
    (msl.equipment.resources.thorlabs.kinesis.benchtop_stapped)motor.BenchtopStepperMotor
        method), 423
        set_legacy_bus_scanning()
set_jog_params_block() (msl.equipment.resources.nkt.nktpdll.NKT
    (msl.equipment.resources.thorlabs.kinesis.integrated_stappmetho)rs.193IntegratedStepperMotors
        method), 498
        set_light() (msl.equipment.resources.picotech.picoscope.ps2000
        method), 343
set_jog_params_block() (msl.equipment.resources.thorlabs.kinesis.kcube_limitermodeibeDCServo
    (msl.equipment.resources.mks_instruments.pr4000b.PR40
        method), 521
        method), 159
set_jog_params_block() (msl.equipment.resources.thorlabs.kinesis.kcube_limiterswitchK6abanc0perMotor
    (msl.equipment.resources.thorlabs.kinesis.benchtop_stappswit)opStepperMotor
        method), 555
        method), 424
set_jog_step_size() (msl.equipment.resources.thorlabs.kinesis.benchtop_stappswit)opStepperMotor
    (msl.equipment.resources.thorlabs.kinesis.integrated_step
        method), 423
        method), 499
set_jog_step_size() (msl.equipment.resources.thorlabs.kinesis.integrated_stappmetho)rs.IntegratedStepperMotors
    (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo
        method), 499
        method), 522
set_jog_step_size() (msl.equipment.resources.thorlabs.kinesis.kcube_limiterswitchK6abanc0perParams()
    (msl.equipment.resources.thorlabs.kinesis.kcube_steerer_
        method), 522
        method), 556
set_jog_step_size() (msl.equipment.resources.thorlabs.kinesis.kcube_limiterswitchK6abanc0perMotor
    (msl.equipment.resources.thorlabs.kinesis.benchtop_stappswit)opStepperMotor
        method), 556
        method), 424
set_jog_vel_params() (msl.equipment.resources.thorlabs.kinesis.benchtop_stappswit)opStepperMotor
    (msl.equipment.resources.thorlabs.kinesis.integrated_step
        method), 423
        method), 500
set_jog_vel_params()

```

```

set_limit_switch_params_block() (msl.equipment.resources.thorlabs.kinesis.kcube_solenoid
    (msl.equipment.resources.thorlabs.kinesis.kcube_dc_method) 523
    method), 523
set_limit_switch_params_block() set_mmi_params_block() (msl.equipment.resources.thorlabs.kinesis.kcube_stepper_
    (msl.equipment.resources.thorlabs.kinesis.kcube_stepper_method) 557
    method), 557
set_limits_software_approach_policy() (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo
    (msl.equipment.resources.thorlabs.kinesis.benchtop_stepper_method) 523
    method), 424
set_limits_software_approach_policy() set_mmi_params_ext() (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo
    (msl.equipment.resources.thorlabs.kinesis.integrated_stepper_motors) 523
    method), 500
set_limits_software_approach_policy() set_mmi_params_ext() (msl.equipment.resources.thorlabs.kinesis.kcube_solenoid
    (msl.equipment.resources.thorlabs.kinesis.kcube_dc_method) 523
    method), 523
set_limits_software_approach_policy() set_mode() (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo
    (msl.equipment.resources.thorlabs.kinesis.kcube_dc_method) 523
    method), 202
set_limits_software_approach_policy() set_mmi_params_ext() (msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo
    (msl.equipment.resources.thorlabs.kinesis.kcube_dc_method) 523
    method), 557
set_linearization_point() set_mmi_params_ext() (msl.equipment.resources.princeton_instruments.arc_instrument
    (msl.equipment.resources.mks_instruments.set100mBR400Br_position)
    method), 159
set_linearization_size() set_mmi_params_ext() (msl.equipment.resources.princeton_instruments.arc_instrument
    (msl.equipment.resources.mks_instruments.set100mBR400Br)
    method), 159
set_lower_limit() set_mmi_params_ext() (msl.equipment.resources.princeton_instruments.arc_instrument
    (msl.equipment.resources.mks_instruments.set100mBR400Br_gadjust)
    method), 160
set_mains() (msl.equipment.resources.picotech.pt104.PT104method), 397
set_mains() set_mono_grating_offset() (msl.equipment.resources.princeton_instruments.arc_instrument
    (msl.equipment.resources.princeton_instruments.arc_instrument
    method), 373
set_max_velocity() set_mono_init_grating() (msl.equipment.resources.princeton_instruments.arc_instrument
    (msl.equipment.resources.thorlabs.fwxx2c.FilterWheel1XX2C), 397
    method), 596
set_message_buffer() set_mono_init_scan_rate_nm() (msl.equipment.resources.princeton_instruments.arc_instrument
    (msl.equipment.resources.energetiq.eq99.EQ99
    method), 146
set_min_velocity() set_mono_init_wave_nm() (msl.equipment.resources.princeton_instruments.arc_instrument
    (msl.equipment.resources.thorlabs.fwxx2c.FilterWheel1XX2C), 397
    method), 597
set_mmi_params() set_mono_int_led() (msl.equipment.resources.princeton_instruments.arc_instrument
    (msl.equipment.resources.thorlabs.kinesis.kcube_dc_method) 523
    method), 523
set_mmi_params() set_mono_scan_rate_nm_min() (msl.equipment.resources.princeton_instruments.arc_instrument
    (msl.equipment.resources.thorlabs.kinesis.kcube_solenoid) 523
    method), 537
set_mmi_params() set_mono_shutter_closed() (msl.equipment.resources.princeton_instruments.arc_instrument
    (msl.equipment.resources.thorlabs.kinesis.kcube_dc_method) 523
    method), 557
set_mmi_params_block() set_mono_shutter_open() (msl.equipment.resources.princeton_instruments.arc_instrument
    (msl.equipment.resources.thorlabs.kinesis.kcube_dc_method) 523
    method), 523
set_mmi_params_block() (msl.equipment.resources.princeton_instruments.arc_instrument
    (msl.equipment.resources.princeton_instruments.arc_instrument
    method), 523

```

method), 398
set_mono_slit_width()
 (*msl.equipment.resources.princeton_instruments.arc*(*instaquipmPrinrestnresthordals.kinesis.integrated_step*
 method), 398
set_mono_turret()
 (*msl.equipment.resources.princeton_instruments.arc*(*instaquipmPrinrestnresthordals.kinesis.kcube_dc_servo*
 method), 398
set_mono_wavelength_abs_cm()
 (*msl.equipment.resources.princeton_instruments.arc*(*instaquipmPrinrestnresthordals.kinesis.kcube stepper*
 method), 399
set_mono_wavelength_ang()
 (*msl.equipment.resources.princeton_instruments.arc*(*instaquipmPrinrestnresthordals.kinesis.benchtop_stepp*
 method), 399
set_mono_wavelength_ev()
 (*msl.equipment.resources.princeton_instruments.arc*(*instaquipmPrinrestnresthordals.kinesis.integrated_step*
 method), 399
set_mono_wavelength_micron()
 (*msl.equipment.resources.princeton_instruments.arc*(*instaquipmPrinrestnresthordals.kinesis.kcube_dc_servo*
 method), 399
set_mono_wavelength_nm()
 (*msl.equipment.resources.princeton_instruments.arc*(*instaquipmPrinrestnresthordals.kinesis.kcube stepper*
 method), 399
set_mono_wavelength_rel_cm()
 (*msl.equipment.resources.princeton_instruments.arc*(*instaquipmPrinrestnresthordals.kinesis.benchtop_stepp*
 method), 399
set_motor_params()
 (*msl.equipment.resources.thorlabs.kinesis.benchtop_staphequiptonBreschtrapeSteppedMotkinesis.integrated_step*
 method), 425
set_motor_params()
 (*msl.equipment.resources.thorlabs.kinesis.integrated_stishpqipmots:disaggregatedSorptlerMotosis.kcube_dc_servo*
 method), 500
set_motor_params()
 (*msl.equipment.resources.thorlabs.kinesis.kcube_dc*(*verstaeKipheDG8sources.thorlabs.kinesis.kcube stepper*
 method), 524
set_motor_params()
 (*msl.equipment.resources.thorlabs.kinesis.kcube stepper*(*msl.equipment.KCubesStappes.Motabs.kinesis.benchtop_stepp*
 method), 558
set_motor_params_ext()
 (*msl.equipment.resources.thorlabs.kinesis.benchtop_staphequiptonBreschtrapeSteppedMotkinesis.integrated_step*
 method), 425
set_motor_params_ext()
 (*msl.equipment.resources.thorlabs.kinesis.integrated_stishpqipmots:disaggregatedSorptlerMotosis.kcube_dc_servo*
 method), 501
set_motor_params_ext()
 (*msl.equipment.resources.thorlabs.kinesis.kcube_dc*(*verstaeKipheDG8sources.thorlabs.kinesis.kcube stepper*
 method), 524
set_motor_params_ext()
 (*msl.equipment.resources.thorlabs.kinesis.kcube stepper*(*msl.equipment.KCubesStappes.Motabs.kinesis.benchtop_stepp*
 method), 558
set_motor_travel_limits()
 (*msl.equipment.resources.thorlabs.kinesis.benchtop_staphequiptonBreschtrapeSteppedMotkinesis.integrated_step*
 method), 425
set_motor_travel_limits()
 (*msl.equipment.resources.princeton_instruments.arc*(*instaquipmPrinrestnresthordals.kinesis.kcube_dc_servo*
 method), 501
set_motor_travel_limits()
 (*msl.equipment.resources.princeton_instruments.arc*(*instaquipmPrinrestnresthordals.kinesis.kcube stepper*
 method), 525
set_motor_travel_limits()
 (*msl.equipment.resources.princeton_instruments.arc*(*instaquipmPrinrestnresthordals.kinesis.benchtop_stepp*
 method), 425
set_motor_travel_mode()
 (*msl.equipment.resources.princeton_instruments.arc*(*instaquipmPrinrestnresthordals.kinesis.integrated_step*
 method), 501
set_motor_travel_mode()
 (*msl.equipment.resources.princeton_instruments.arc*(*instaquipmPrinrestnresthordals.kinesis.kcube_dc_servo*
 method), 525
set_motor_travel_mode()
 (*msl.equipment.resources.princeton_instruments.arc*(*instaquipmPrinrestnresthordals.kinesis.kcube stepper*
 method), 559
set_motor_velocity_limits()
 (*msl.equipment.resources.princeton_instruments.arc*(*instaquipmPrinrestnresthordals.kinesis.benchtop_stepp*
 method), 426
set_motor_velocity_limits()
 (*msl.equipment.resources.thorlabs.kinesis.integrated_stishpqipmots:disaggregatedSorptlerMotosis.kcube_dc_servo*
 method), 501
set_motor_velocity_limits()
 (*msl.equipment.resources.thorlabs.kinesis.kcube_dc*(*verstaeKipheDG8sources.thorlabs.kinesis.kcube stepper*
 method), 559
set_move_absolute_position()
 (*msl.equipment.resources.thorlabs.kinesis.kcube stepper*(*msl.equipment.KCubesStappes.Motabs.kinesis.benchtop_stepp*
 method), 426
set_move_absolute_position()
 (*msl.equipment.resources.thorlabs.kinesis.benchtop_staphequiptonBreschtrapeSteppedMotkinesis.integrated_step*
 method), 502
set_move_absolute_position()
 (*msl.equipment.resources.thorlabs.kinesis.integrated_stishpqipmots:disaggregatedSorptlerMotosis.kcube_dc_servo*
 method), 525
set_move_absolute_position()
 (*msl.equipment.resources.thorlabs.kinesis.kcube_dc*(*verstaeKipheDG8sources.thorlabs.kinesis.kcube stepper*
 method), 559
set_move_absolute_position()
 (*msl.equipment.resources.thorlabs.kinesis.kcube stepper*(*msl.equipment.KCubesStappes.Motabs.kinesis.benchtop_stepp*
 method), 426
set_move_absolute_position()
 (*msl.equipment.resources.thorlabs.kinesis.benchtop_staphequiptonBreschtrapeSteppedMotkinesis.integrated_step*
 method), 559
set_move_absolute_position()
 (*msl.equipment.resources.thorlabs.kinesis.integrated_stishpqipmots:disaggregatedSorptlerMotosis.kcube_dc_servo*
 method), 525
set_move_absolute_position()
 (*msl.equipment.resources.thorlabs.kinesis.kcube_dc*(*verstaeKipheDG8sources.thorlabs.kinesis.kcube stepper*
 method), 559
set_move_relative_distance()
 (*msl.equipment.resources.thorlabs.kinesis.kcube stepper*(*msl.equipment.KCubesStappes.Motabs.kinesis.benchtop_stepp*
 method), 426
set_move_relative_distance()
 (*msl.equipment.resources.thorlabs.kinesis.benchtop_staphequiptonBreschtrapeSteppedMotkinesis.integrated_step*
 method), 426

method), 502
set_move_relative_distance()
(msl.equipment.resources.thorlabs.kinesis.kcube_origin(), KCubeStepperMotor
method), 526
set_move_relative_distance()
(msl.equipment.resources.thorlabs.kinesis.kcube_stepper(), KCubeStepperMotor
method), 560
set_multi_off_action()
(msl.equipment.resources.aim_tti.mx_series.MXSeriesOutputDrive()
method), 96
set_multi_off_delay()
(msl.equipment.resources.aim_tti.mx_series.MXSeriesOutputRange()
method), 96
set_multi_on_action()
(msl.equipment.resources.aim_tti.mx_series.MXSeriesOutputStatus()
method), 95
set_multi_on_delay()
(msl.equipment.resources.aim_tti.mx_series.MXSeriesOvervoltageProtection()
method), 95
set_ncl_filter_position()
(msl.equipment.resources.princeton_instruments.sets_overvoltage_protection(), PrincetonInstruments
method), 399
set_ncl_filter_present()
(msl.equipment.resources.princeton_instruments.sets_parameter(), PrincetonInstruments
method), 399
set_ncl_shutter_closed()
(msl.equipment.resources.princeton_instruments.sets_pid_loopenLoopCoefficients(), ThorlabsKinesisBenchtopStepper)
method), 400
set_ncl_shutter_open()
(msl.equipment.resources.princeton_instruments.sets_pid_loopenLoopCoefficients(), ThorlabsKinesisBenchtopStepper)
method), 400
set_ncl_ttl_out_off()
(msl.equipment.resources.princeton_instruments.sets_pid_loopenLoopParams(), ThorlabsKinesisBenchtopStepper)
method), 400
set_ncl_ttl_out_on()
(msl.equipment.resources.princeton_instruments.sets_pid_loopenLoopParams(), ThorlabsKinesisBenchtopStepper)
method), 400
set_no_of_captures()
(msl.equipment.resources.picotech.picoscope.sets_analogInputOverloadVoltage(), OptonicLaboratoriesOl7560cx)
method), 338
set_oem_parameter()
(msl.equipment.resources.avantes.avaspec.Asentiment_hi_voltage(), OptonicLaboratoriesOl7560cx)
method), 125
set_offset()
(msl.equipment.resources.mks_instruments.pr4000bd.R4000BD.set_position(), FilterWheelXX)
method), 160
set_operating_mode()
(msl.equipment.resources.thorlabs.kinesis.kcube_solenoid.KCubeSolenoid
method), 537
set_operating_state()
(msl.equipment.resources.thorlabs.kinesis.kcube_solenoid.KCubeSolenoid
method), 537
set_option() (msl.equipment.resources.optonic_laboratories.ol7560cx.set_origin(), KCubeStepperMotor
method), 226
set_output() (msl.equipment.resources.electron_dynamics.tc_series.TCOutput()
method), 203
set_output() (msl.equipment.resources.energetiq.eq99.EQ99.set_output_drive(), MKSInstrumentOutputDrive()
method), 144
set_output() (msl.equipment.resources.electron_dynamics.tc_series.TCOutput(), MKSInstrumentOutputRange())
method), 139
set_output() (msl.equipment.resources.mks_instruments.pr4000b.PR4000B.set_output_status(), MKSInstrumentOutputStatus())
method), 160
set_overvoltage_protection() (msl.equipment.resources.aim_tti.mx_series.MXSeriesOvervoltageProtection(), PrincetonInstruments)
method), 96
set_parameter() (msl.equipment.resources.avantes.avaspec.Avantes.sets_parameter(), PrincetonInstruments)
method), 125
set_pid_loopenLoopCoefficients() (msl.equipment.resources.thorlabs.kinesis.benchtop_stepper.sets_pid_loopenLoopCoefficients(), ThorlabsKinesisBenchtopStepper)
method), 426
set_pid_loopenLoopParams() (msl.equipment.resources.thorlabs.kinesis.benchtop_stepper.sets_pid_loopenLoopParams(), ThorlabsKinesisBenchtopStepper)
method), 560
set_position() (msl.equipment.resources.thorlabs.kinesis.benchtop_stepper.sets_position(), ThorlabsKinesisBenchtopStepper)
method), 427
set_position() (msl.equipment.resources.thorlabs.kinesis.kcube_stepper.sets_position(), ThorlabsKinesisBenchtopStepper)
method), 560
set_analogInputOverloadVoltage() (msl.equipment.resources.optonic_laboratories.ol7560cx.set_analogInputOverloadVoltage(), OptonicLaboratoriesOl7560cx)
method), 222
set_loopenLoopCoefficients() (msl.equipment.resources.optonic_laboratories.ol7560cx.set_loopenLoopCoefficients(), OptonicLaboratoriesOl7560cx)
method), 226
set_loopenLoopParams() (msl.equipment.resources.optonic_laboratories.ol7560cx.set_loopenLoopParams(), OptonicLaboratoriesOl7560cx)
method), 227
set_position_count() (msl.equipment.resources.thorlabs.fwxx2c.FilterWheelXX.set_position_count(), FilterWheelXX)
method), 537
set_position_counter() (msl.equipment.resources.thorlabs.fwxx2c.FilterWheelXX.set_position_counter(), FilterWheelXX)
method), 537

(*msl.equipment.resources.thorlabs.kinesis.benchtopstepper().method*) (*method*) *set_rack_digital_outputs()*
method), 427
method), 125

set_position_counter()
(*msl.equipment.resources.thorlabs.kinesis.integrated_stethometers.integrated_dc_servo().method*) *set_rack_digital_outputs()*
method), 502
method), 427

set_position_counter()
(*msl.equipment.resources.thorlabs.kinesis.kcube_dc_method().KCubeDCServo*)
method), 526
method), 125

set_position_counter()
(*msl.equipment.resources.thorlabs.kinesis.kcube_stepper().40KCubeStepperMotor*)
method), 560
method), 125

set_potentiometer_params()
(*msl.equipment.resources.thorlabs.kinesis.integrated_stethometers.integrated_dc_servo().method*) *set_reference_white_point()*
method), 502
method), 125

set_potentiometer_params_block()
(*msl.equipment.resources.thorlabs.kinesis.integrated_stethometers.integrated_dc_servo().method*)
method), 503
method), 125

set_power_off_time()
(*msl.equipment.resources.greisinger.gmh3000.set_GMMode_error().method*)
method), 149
method), 125

set_power_params()
(*msl.equipment.resources.thorlabs.kinesis.benchtopstepper().BenchtopStepperMotor*)
method), 427
method), 125

set_power_params()
(*msl.equipment.resources.thorlabs.kinesis.integrated_stethometers.integrated_dc_servo().method*)
method), 503
method), 125

set_power_params()
(*msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo().BenchtopStepperMotor*)
method), 561
method), 125

set_prescan_mode()
(*msl.equipment.resources.avantes.avaspec.Aset_annotation_modes().method*)
method), 125
method), 125

set_probe() (*msl.equipment.resources.picotech.picoscope.ps4000PicScope4000.set_rotation_modes().method*)
method), 351
method), 125

set_ps() (*msl.equipment.resources.cmi.sia3.SIA3.set_rtd_offset().method*)
method), 132
method), 125

set_pulse_width_digital_port_properties()
(*msl.equipment.resources.picotech.picoscope.ps3000.set_rtd_offset().method*)
method), 348
method), 125

set_pulse_width_qualifier()
(*msl.equipment.resources.picotech.picoscope.picoscope4000A.set_rtd_offset().method*)
method), 329
method), 125

set_pulse_width_qualifier_conditions()
(*msl.equipment.resources.picotech.picoscope.ps4000.set_rtd_offset().method*)
method), 353
method), 125

set_pulse_width_qualifier_properties()
(*msl.equipment.resources.picotech.picoscope.ps4000.set_rtd_offset().method*)
method), 353
method), 125

set_pulse_width_qualifier_v2()
(*msl.equipment.resources.picotech.picoscope.set_rtd_offset().method*)
method), 349
method), 125

(*msl.equipment.resources.optronic_laboratories.ol756ocx.set_annotation_modes().method*)
method), 125

set_relays() (*msl.equipment.resources.mks_instruments.pr4000b.PR4000b.set_annotation_modes().method*)
method), 160
method), 125

set_rack_digital_outputs()
(*msl.equipment.resources.thorlabs.kinesis.benchtopstepper().BenchtopStepperMotor*)
method), 427
method), 125

set_range() (*msl.equipment.resources.mks_instruments.pr4000b.PR4000b.set_annotation_modes().method*)
method), 160
method), 125

set_readout_itime_ms()
(*msl.equipment.resources.princeton_instruments.arc_instrument.set_annotation_modes().method*)
method), 125

set_reference_white_point()
(*msl.equipment.resources.optronic_laboratories.ol756ocx.set_annotation_modes().method*)
method), 125

set_relays() (*msl.equipment.resources.mks_instruments.pr4000b.PR4000b.set_annotation_modes().method*)
method), 160
method), 125

set_rtd_offset()
(*msl.equipment.resources.thorlabs.kinesis.benchtopstepper().BenchtopStepperMotor*)
method), 161
method), 125

set_rotation_modes()
(*msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo().BenchtopStepperMotor*)
method), 125

set_sensitivity_mode()
(*msl.equipment.resources.optronic_laboratories.ol756ocx.set_annotation_modes().method*)
method), 125

set_sensor() (*msl.equipment.resources.electron_dynamics.tc_set_annotation_modes().method*)
method), 139
method), 125

set_sensitivity_mode()
(*msl.equipment.resources.thorlabs.fwxx2c.FilterWheelXX.set_annotation_modes().method*)
method), 125

method), 597
set_setpoint() *(msl.equipment.resources.electron_dynamics.set_speed(), method), 140*
set_setpoint() *(msl.equipment.resources.mks_instruments.set1000labPR4000B_limits(), method), 161*
set_setpoint() *(msl.equipment.resources.raicol.raicol_tec.RetroStage_axis_limits(), method), 401*
set_settling_time() *(msl.equipment.resources.optronic_laboratory.setStageAxisLimits(), method), 222*
set_shutter_init() *(msl.equipment.resources.energetiq.eq99.EQ99_stage_axis_limits(), method), 145*
set_shutter_state() *(msl.equipment.resources.energetiq.eq99.EQ99_steps(), msl.equipment.resources.optosigma.shot702.SHOT702_stage_axis_limits(), method), 145*
set_sig_gen_arbitrary() *(msl.equipment.resources.picotech.picoscope.picoscope.setSyncMode(), method), 338*
set_sig_gen_arbitrary() *(msl.equipment.resources.picotech.picoscope.ps2000(PicoScope2000Resources.optronic_laboratories.ol756ocx.setTabDelimitedMode(), method), 343*
set_sig_gen_builtin() *(msl.equipment.resources.picotech.picoscope.ps2000(PicoScope2000Resources.energetiq.eq99.EQ99.setTermination(), method), 343*
set_sig_gen_builtin() *(msl.equipment.resources.picotech.picoscope.picoscope.setTest(), msl.equipment.resources.electron_dynamics.tc_series.setTimebase(), method), 339*
set_sig_gen_builtin_v2() *(msl.equipment.resources.picotech.picoscope.picoscope.setTimeout(), PicoScopeApi.setTransitTime(), method), 339*
set_sig_gen_properties_arbitrary() *(msl.equipment.resources.picotech.picoscope.picoscope.setTransitTime(), RialSequippment.vxi11.RPCClient.setTimeout(), method), 340*
set_sig_gen_properties_builtin() *(msl.equipment.resources.picotech.picoscope.picoscope.setTransitTime(), thorlabs.kinesis.filter_flipper.FilterWheelXX.setTrigger(), method), 340*
set_siggen() *(msl.equipment.resources.picotech.picoscope.setTrigger(), RialSequippment.resources.picotech.picoscope.picoscope.setTrigger(), method), 346*
set_signal_mode() *(msl.equipment.resources.mks_instruments.pr4000b.PR4000B.setTrigger(), method), 161*
set_simple_trigger() *(msl.equipment.resources.picotech.picoscope.picoscope.setTrigger(), PicoScopeApi.setTriggerChannelConditions(), method), 340*
set_speed() *(msl.equipment.resources.optosigma.shot702.SHOT702.setSpeed(), msl.equipment.resources.picotech.picoscope.picoscope.setTriggerChannelConditions(), method), 203*
set_speed_mode() *(msl.equipment.resources.picotech.picoscope.picoscope.setTriggerChannelConditionsV2(), method), 341*

(*msl.equipment.resources.picotech.picoscope.ps3000*(~~analogInput~~^{method})_{3000A}*sources.thorlabs.kinesis.kcube_dc_servo*
method), 349
set_trigger_channel_directions() *set_trigger_params_params_block()*
(*msl.equipment.resources.picotech.picoscope.picoscopemsl.picoscope*(~~analogInput~~^{method})_{3000A}*sources.thorlabs.kinesis.kcube stepper*
method), 341 *set_trigger_params_params_block()*
set_trigger_channel_directions() *set_trigger_switches()*
(*msl.equipment.resources.picotech.picoscope.ps4000*(~~analogInput~~^{method})_{4000A}*sources.thorlabs.kinesis.benchtop_stepper*
method), 353 *set_trigger_switches()*
set_trigger_channel_properties() *set_trigger_switches()*
(*msl.equipment.resources.picotech.picoscope.picoscopemsl.picoscope*(~~analogInput~~^{method})_{3000A}*sources.thorlabs.kinesis.integrated_step*
method), 342 *set_trigger_switches()*
set_trigger_config_params() *set_tweak_control()*
(*msl.equipment.resources.thorlabs.kinesis.kcube_dc_gearsteadiKClipheDCServo*_{mks_instruments.pr4000b.PR4000}
method), 527 *set_tweak_control()*
set_trigger_config_params() *set_upper_limit()*
(*msl.equipment.resources.thorlabs.kinesis.kcube_solenoidKClipheSolenoid*_{mks_instruments.pr4000b.PR4000}
method), 538 *set_upper_limit()*
set_trigger_config_params() *set_user_defined_integration_time()*
(*msl.equipment.resources.thorlabs.kinesis.kcube_steppeKClipheStepperMotor*_{Metric_laboratories.ol756ocx}
method), 561 *set_user_defined_integration_time()*
set_trigger_config_params_block() *set_valves()* (*msl.equipment.resources.mks_instruments.pr4000b.PR4000*
(*msl.equipment.resources.thorlabs.kinesis.kcube_dc_gearsteadiKClipheDCServo*
method), 527 *set_valves()*
set_trigger_config_params_block() *set_vel_params()* (*msl.equipment.resources.thorlabs.kinesis.benchtop_stepper*
(*msl.equipment.resources.thorlabs.kinesis.kcube_solenoidKClipheSolenoid*
method), 538 *set_vel_params()*
set_trigger_config_params_block() *set_valves()* (*msl.equipment.resources.thorlabs.kinesis.integrated_step*
(*msl.equipment.resources.thorlabs.kinesis.kcube_steppe*~~method~~^{method})_{500KCubeStepperMotor}
method), 562 *set_valves()*
set_trigger_delay() *set_valves()* (*msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo*
(*msl.equipment.resources.picotech.picoscope.picoscope*(~~analogInput~~^{method})_{PICOSCOPEAPI}
method), 341 *set_valves()*
set_trigger_digital_port_properties() *set_valves()* (*msl.equipment.resources.thorlabs.kinesis.kcube stepper*
(*msl.equipment.resources.picotech.picoscope.ps2000*(~~analogInput~~^{method})₅₆₂*ps2000A*
method), 345 *set_valves()*
set_trigger_digital_port_properties() *set_valves()* (*msl.equipment.resources.thorlabs.kinesis.benchtop_stepper*
(*msl.equipment.resources.picotech.picoscope.ps3000*(~~analogInput~~^{method})₄₂₉*ps3000A*
method), 349 *set_valves()*
set_trigger_mode() *set_valves()* (*msl.equipment.resources.thorlabs.kinesis.integrated_step*
(*msl.equipment.resources.energetiq.eq99.EQ99*(~~analogInput~~^{method})₅₀₄*EQ99*
method), 145 *set_valves()*
set_trigger_mode() *set_valves()* (*msl.equipment.resources.thorlabs.kinesis.kcube_dc_servo*
(*msl.equipment.resources.thorlabs.fwxx2c.FilterWheel*(~~analogInput~~^{method})₅₂₈*FWXX2C*
method), 597 *set_valves()*
set_trigger_params_params() *set_valves()* (*msl.equipment.resources.thorlabs.kinesis.kcube stepper*
(*msl.equipment.resources.thorlabs.kinesis.kcube_dc_gearsteadiKClipheDCServo*
method), 527 *set_valves()*
set_trigger_params_params() *set_voltage()* (*msl.equipment.resources.aim_tti.mx_series.MXS*
(*msl.equipment.resources.thorlabs.kinesis.kcube_dc_gearsteadiKClipheDCServo*
method), 562 *set_voltage()*
set_trigger_params_params_block() *set_voltage_range()*

```

(msl.equipment.resources.aim_tti.mx_series.MXSeriesMXSGen.frequency_to_phase())
    method), 97
set_voltage_step_size() (msl.equipment.resources.picotech.picoscope.picoscope_a_
    method), 341
set_voltage_tracking_mode() (msl.equipment.resources.aim_tti.mx_series.MXSeriesMXSGen.software_control())
    method), 97
set_voltage_tracking_mode() (msl.equipment.resources.aim_tti.mx_series.MXSeriesMXSGen.GENERATOR)
    method), 97
set_wattage() (msl.equipment.resources.optronic_laboratory.invisible_light_source.OLCurrentSource
    method), 227
set_waveform_limiter() (msl.equipment.resources.picotech.picoscope.ps6000.Picoscope2000
    method), 357
settings (msl.equipment.resources.thorlabs.kinesis.motion_controller.motion_constraints.picotech.picoscope.enums.PS400
    property), 569
SETTINGS_RESERVED_LEN SIGNAL_GENERATOR_AWG_DELTA_PHASE_INCREMENT
(msl.equipment.resources.avantes.avaspec.Avantes
    attribute), 106
settledError (msl.equipment.resources.thorlabs.kinesis.motion_controller.motion_constraints.picotech.picoscope.enums.PS400
    attribute), 575
settleSamples (msl.equipment.resources.thorlabs.kinesis.stability.DFARangeParameters
    attribute), 586
settleSamples (msl.equipment.resources.thorlabs.kinesis.stability.DFARangeParameters.picotech.picoscope.enums.PS400
    attribute), 578
SEVEN (msl.equipment.constants.DataBits
    attribute), 51
shared_released (msl.equipment.hislip.AsyncLockResponse
    property), 63
SHORT_PRESS (msl.equipment.resources.picotech.picoscope.enums.PS2000.ButtonState
    attribute), 237
SHOT702 (class in msl.equipment.resources.optosigma.shot702,
    attribute), 200
SHOT_SWEEP_TRIGGER_CONTINUOUS_RUN (msl.equipment.resources.picotech.picoscope.ps2000a.Attributes)
    attribute), 344
SHOT_SWEEP_TRIGGER_CONTINUOUS_RUN (msl.equipment.resources.picotech.picoscope.ps3000a.Attributes)
    attribute), 348
SHOT_SWEEP_TRIGGER_CONTINUOUS_RUN (msl.equipment.resources.picotech.picoscope.ps5000a.Attributes)
    attribute), 355
shutdown_handler() (msl.equipment.resources.dataray.datarayocx_32.DatarayOCX32)
    method), 133
SIA3 (class in msl.equipment.resources.cmi.sia3),
    132
sig_gen_arbitrary_min_max_values() (msl.equipment.resources.picotech.picoscope.picoscope_attributes.picotech.picoscope.enums.PS400
    method), 341

```

SIGNAL_GENERATOR_EXT_IN_THRESHOLD	SINC_MAX_FREQUENCY (<i>msl.equipment.resources.picotech.picoscope.enums.PS4000APriceStringValues.picotech.picoscope.ps3000a.Price attribute</i>), 296	SINC_MAX_FREQUENCY (<i>msl.equipment.resources.picotech.picoscope.enums.PS4000APriceStringValues.picotech.picoscope.ps4000a.Price attribute</i>), 348
SIGNAL_GENERATOR_OFFSET_VOLTAGE	SINC_MAX_FREQUENCY (<i>msl.equipment.resources.picotech.picoscope.enums.PS4000APriceStringValues.picotech.picoscope.ps4000a.Price attribute</i>), 295	SINC_MAX_FREQUENCY (<i>msl.equipment.resources.picotech.picoscope.enums.PS4000APriceStringValues.picotech.picoscope.ps4000a.Price attribute</i>), 352
SIGNAL_GENERATOR_OPERATION	SINC_MAX_FREQUENCY (<i>msl.equipment.resources.picotech.picoscope.enums.PS4000APriceStringValues.picotech.picoscope.ps5000.Price attribute</i>), 295	SINC_MAX_FREQUENCY (<i>msl.equipment.resources.picotech.picoscope.enums.PS4000APriceStringValues.picotech.picoscope.ps5000.Price attribute</i>), 354
SIGNAL_GENERATOR_PK_TO_PK	SINC_MAX_FREQUENCY (<i>msl.equipment.resources.picotech.picoscope.enums.PS4000APriceStringValues.picotech.picoscope.ps5000a.Price attribute</i>), 295	SINC_MAX_FREQUENCY (<i>msl.equipment.resources.picotech.picoscope.enums.PS4000APriceStringValues.picotech.picoscope.ps5000a.Price attribute</i>), 355
SIGNAL_GENERATOR_SETTINGS	SINC_MAX_FREQUENCY (<i>msl.equipment.resources.picotech.picoscope.enums.PS4000APriceTypesources.picotech.picoscope.ps6000.Price attribute</i>), 286	SINC_MAX_FREQUENCY (<i>msl.equipment.resources.picotech.picoscope.enums.PS4000APriceTypesources.picotech.picoscope.ps6000.Price attribute</i>), 357
SIGNAL_GENERATOR_SHOTS	SINE (<i>msl.equipment.resources.picotech.picoscope.enums.PS2000APriceStringValue</i> <i>msl.equipment.resources.picotech.picoscope.enums.PS4000APriceStringValue</i>)	SINE (<i>msl.equipment.resources.picotech.picoscope.enums.PS2000APriceStringValue</i> <i>msl.equipment.resources.picotech.picoscope.enums.PS4000APriceStringValue</i>)
SIGNAL_GENERATOR_SWEEP_TYPE	SINE (<i>msl.equipment.resources.picotech.picoscope.enums.PS4000APriceStringValue</i> <i>msl.equipment.resources.picotech.picoscope.enums.PS3000APriceStringValue</i>)	SINE (<i>msl.equipment.resources.picotech.picoscope.enums.PS3000APriceStringValue</i> <i>msl.equipment.resources.picotech.picoscope.enums.PS2000APriceStringValue</i>)
SIGNAL_GENERATOR_SWEEPS	SINE (<i>msl.equipment.resources.picotech.picoscope.enums.PS4000APriceStringValue</i> <i>msl.equipment.resources.picotech.picoscope.enums.PS4000APriceStringValue</i>)	SINE (<i>msl.equipment.resources.picotech.picoscope.enums.PS3000WPriceStringValue</i> <i>msl.equipment.resources.picotech.picoscope.enums.PS4000WPriceStringValue</i>)
SIGNAL_GENERATOR_TRIGGER_SOURCE	SINE (<i>msl.equipment.resources.picotech.picoscope.enums.PS4000APriceStringValue</i> <i>msl.equipment.resources.picotech.picoscope.enums.PS4000WPriceStringValue</i>)	SINE (<i>msl.equipment.resources.picotech.picoscope.enums.PS4000WPriceStringValue</i> <i>msl.equipment.resources.picotech.picoscope.enums.PS4000APriceStringValue</i>)
SIGNAL_GENERATOR_TRIGGER_TYPE	SINE (<i>msl.equipment.resources.picotech.picoscope.enums.PS4000APriceStringValue</i> <i>msl.equipment.resources.picotech.picoscope.enums.PS5000APriceStringValue</i>)	SINE (<i>msl.equipment.resources.picotech.picoscope.enums.PS5000APriceStringValue</i> <i>msl.equipment.resources.picotech.picoscope.enums.PS5000WPriceStringValue</i>)
SIGNAL_MODES (<i>msl.equipment.resources.mks_instruments.pr4000b/pr4000B</i> <i>attribute</i>), 151	SINE (<i>msl.equipment.resources.picotech.picoscope.enums.PS6000WquietType</i> <i>msl.equipment.resources.picotech.picoscope.enums.PS2000WquietType</i>)	SINE (<i>msl.equipment.resources.picotech.picoscope.enums.PS6000WquietType</i> <i>msl.equipment.resources.picotech.picoscope.enums.PS2000quietType</i>)
SINC (<i>msl.equipment.resources.picotech.picoscope.enums.PS2000AWaveType</i> <i>attribute</i>), 246	SINE_MAX_FREQUENCY	SINE_MAX_FREQUENCY
SINC (<i>msl.equipment.resources.picotech.picoscope.enums.PS2000WquietType</i> <i>attribute</i>), 237	SINE (<i>msl.equipment.resources.picotech.picoscope.ps2000a.PriceStringValue</i> <i>msl.equipment.resources.picotech.picoscope.ps2000a.PriceStringValue</i>)	SINE (<i>msl.equipment.resources.picotech.picoscope.ps2000a.PriceStringValue</i> <i>msl.equipment.resources.picotech.picoscope.ps2000a.PriceStringValue</i>)
SINC (<i>msl.equipment.resources.picotech.picoscope.enums.PS3000WquietType</i> <i>attribute</i>), 263	SINE_MAX_FREQUENCY	SINE_MAX_FREQUENCY
SINC (<i>msl.equipment.resources.picotech.picoscope.enums.PS4000AWaveType</i> <i>attribute</i>), 285	SINE_MAX_FREQUENCY	SINE_MAX_FREQUENCY
SINC (<i>msl.equipment.resources.picotech.picoscope.enums.PS4000WquietType</i> <i>attribute</i>), 275	SINE (<i>msl.equipment.resources.picotech.picoscope.ps4000a.PriceStringValue</i> <i>msl.equipment.resources.picotech.picoscope.ps4000a.PriceStringValue</i>)	SINE (<i>msl.equipment.resources.picotech.picoscope.ps4000a.PriceStringValue</i> <i>msl.equipment.resources.picotech.picoscope.ps4000a.PriceStringValue</i>)
SINC (<i>msl.equipment.resources.picotech.picoscope.enums.PS5000WquietType</i> <i>attribute</i>), 309	SINE_MAX_FREQUENCY	SINE_MAX_FREQUENCY
SINC (<i>msl.equipment.resources.picotech.picoscope.enums.PS5000AWaveType</i> <i>attribute</i>), 299	SINE_MAX_FREQUENCY	SINE_MAX_FREQUENCY
SINC (<i>msl.equipment.resources.picotech.picoscope.enums.PS6000WquietType</i> <i>attribute</i>), 318	SINE (<i>msl.equipment.resources.picotech.picoscope.ps5000a.PriceStringValue</i> <i>msl.equipment.resources.picotech.picoscope.ps5000a.PriceStringValue</i>)	SINE (<i>msl.equipment.resources.picotech.picoscope.ps5000a.PriceStringValue</i> <i>msl.equipment.resources.picotech.picoscope.ps5000a.PriceStringValue</i>)
SINC_MAX_FREQUENCY	SINE_MAX_FREQUENCY	SINE_MAX_FREQUENCY
	(<i>msl.equipment.resources.picotech.picoscope.ps2000a.PriceStringValue</i> <i>msl.equipment.resources.picotech.picoscope.ps2000a.PriceStringValue</i>)	(<i>msl.equipment.resources.picotech.picoscope.ps6000a.PriceStringValue</i> <i>msl.equipment.resources.picotech.picoscope.ps6000a.PriceStringValue</i>)

SINGLE (*msl.equipment.resources.picotech.picoscope.socket.PS3000AIndexMode*.connection_socket.ConnectionSocket attribute), 248
 SINGLE (*msl.equipment.resources.picotech.picoscope.socket.PS3000AIndexMode*.connection_tcpip_vxi11.ConnectionTCPPIP attribute), 265
 SINGLE (*msl.equipment.resources.picotech.picoscope.socket.PS4000AIndexMode*.connection_zeromq.ConnectionZeroMQ attribute), 288
 SINGLE (*msl.equipment.resources.picotech.picoscope.SOCKET_PS4000AIndexMode*.constants.Interface attribute), 50
 SINGLE (*msl.equipment.resources.picotech.picoscope.socket.PS5000AIndexMode*.HiSLIPClient property), 74
 SINGLE (*msl.equipment.resources.picotech.picoscope.socket.PS5000AIndexMode*.RPCClient property), 604
 SINGLE (*msl.equipment.resources.picotech.picoscope.SoftTRIGMode*.resources.picotech.picoscope.enums.PS attribute), 247
 SINGLE_ENDED_TO_115MV
 (msl.equipment.resources.picotech.pt104.Pt104DataType attribute), 265
 (attribute), 371
 SINGLE_ENDED_TO_2500MV
 (msl.equipment.resources.picotech.pt104.PtSOFTTRIG attribute), 276
 (attribute), 371
 SIX (*msl.equipment.constants.DataBits* attribute), 51
 SIX (*msl.equipment.resources.thorlabs.fwxx2c.Filter* SOFT_TRIG (*msl.equipment.resources.picotech.picoscope.enums.PS* attribute), 300
 size (*msl.equipment.hislip.Message* property), 60
 SLOW (*msl.equipment.resources.picotech.picoscope.enums.PS2000AEtsMode*.attribute), 244
 softLimitMode (*msl.equipment.resources.thorlabs.kinesis.structs.PS2000AEtsMode*.attribute), 276
 SLOW (*msl.equipment.resources.picotech.picoscope.enums.PS2000AEtsMode*.attribute), 236
 Solenoid (in module
 SLOW (*msl.equipment.resources.picotech.picoscope.enums.PS3000AEtsMode*.resources.thorlabs.kinesis.messages), 566
 SLOW (*msl.equipment.resources.picotech.picoscope.enums.PS3000AEtsMode*.attribute), 262
 SLOW (*msl.equipment.resources.picotech.picoscope.enums.PS3000AEtsMode*.attribute), 255
 SLOW (*msl.equipment.resources.picotech.picoscope.enums.PS3000AEtsMode*.attribute), 284
 SLOW (*msl.equipment.resources.picotech.picoscope.enums.PS3000AEtsMode*.attribute), 273
 SLOW (*msl.equipment.resources.picotech.picoscope.enums.PS3000AEtsMode*.attribute), 307
 SLOW (*msl.equipment.resources.picotech.picoscope.enums.PS3000AEtsMode*.attribute), 298
 SLOW (*msl.equipment.resources.picotech.picoscope.enums.PS3000AEtsMode*.attribute), 317
 SLOW (*msl.equipment.resources.thorlabs.fwxx2c.SpeedMode* attribute), 102
 SmoothingSamples
 (msl.equipment.resources.thorlabs.kinesis.structs.KS69TriggerConfig attribute), 592
 SmoothingType (class in msl.equipment.resources.avantes.avaspec), 101
 SQUARE (*msl.equipment.resources.picotech.picoscope.enums.PS2000AEtsMode*.attribute), 246

status() (*msl.equipment.resources.optosigma.shot702.SHOT702Method*), 563
method, 205
stop_profiled()
stepSize() (*msl.equipment.resources.thorlabs.kinesis structs.MOT_JogParameter*), 572
attribute, 572
stop() (*msl.equipment.resources.dataray.datarayoc_stopDatalogMode*), 32
method, 133
stop() (*msl.equipment.resources.dataray.datarayocx_64.DatrayOCXMethod*), 505
method, 136
stop() (*msl.equipment.resources.optosigma.shot702.SHOT702Method*), 529
method, 205
stop() (*msl.equipment.resources.picotech.picoscope_stopScopePulse*), 328
method, 328
stop_bits() (*msl.equipment.connection_serial.ConnectionSerialMethod*), 564
property, 40
stop_immediate()
(msl.equipment.resources.thorlabs.kinesis bitsSteplessInOneStepModeSteplessMethod), 51
method, 430
stop_immediate()
(msl.equipment.resources.thorlabs.kinesis integratedStepModeSteplessMethod), 572
method, 504
stop_immediate()
(msl.equipment.resources.thorlabs.kinesis integratedStepModeSteplessMethod), 529
method, 529
stop_immediate()
(msl.equipment.resources.thorlabs.kinesis streamReaders), 32
method, 563
stop_log() (*msl.equipment.resources.bentham.benthamLogManager2getIntervalStateless*), 130
method, 130
stop_measure()
(msl.equipment.resources.avantes.avaspec.Measure), 347
method, 126
stop_measurement()
(msl.equipment.resources.optronic_laboratories.ol75ps3000.PicoScope), 566
method, 223
stop_polling()
(msl.equipment.resources.thorlabs.kinesis.benchtopStepperSteplessSteplessMethod), 70
method, 430
stop_polling()
(msl.equipment.resources.thorlabs.kinesis.filter_flipperFilterFlipper), 70
method, 482
stop_polling()
(msl.equipment.resources.thorlabs.kinesis.integratedEquipmentInterfaceSteplessSteplessMethod), 602
method, 505
stop_polling()
(msl.equipment.resources.thorlabs.kinesis.kcube_dc_gyroKCubeDCMethod), 126
method, 529
stop_polling()
(msl.equipment.resources.thorlabs.kinesis.kcube_dc_gyroKCubeDCMethod), 430
method, 538
stop_polling()
(msl.equipment.resources.thorlabs.kinesis.kcube_dc_gyroKCubeDCMethod), 590
method, 590
suppress_stray_light()
(msl.equipment.resources.thorlabs.kinesis.kcube_dc_gyroKCubeDCMethod), 126
method, 126
suspend_move_messages()
(msl.equipment.resources.thorlabs.kinesis.kcube_dc_gyroKCubeDCMethod), 430
method, 430
suspend_move_messages()
(msl.equipment.resources.thorlabs.kinesis.kcube_dc_gyroKCubeDCMethod), 590
method, 590

method), 529

suspend_move_messages() *(msl.equipment.resources.thorlabs.kinesis.enums.TST_Stages attribute), 476*

(msl.equipment.resources.thorlabs.kinesis.enums.TST_Stages attribute), 564

SW_TRIGGER_MODE *TBD2 (msl.equipment.resources.thorlabs.kinesis.enums.TST_Stages attribute), 476*

(msl.equipment.resources.avantes.avaspec.Avantes attribute), 476

SwitchBreaks *(msl.equipment.resources.thorlabs.kinesis.enums.KIM_Mobile_Switch_Modes attribute), 477*

SwitchBreaks_HomeOnly *TC_ActualTemperature*

(msl.equipment.resources.thorlabs.kinesis.enums.KIM_Mobile_Switch_Modes attribute), 477

SwitchMakes *(msl.equipment.resources.thorlabs.kinesis.enums.KIM_Mobile_Switch_Modes attribute), 477*

SwitchMakes_HomeOnly *TC_Current (msl.equipment.resources.thorlabs.kinesis.enums.TC_Current attribute), 477*

(msl.equipment.resources.thorlabs.kinesis.enums.KIM_Mobile_Switch_Modes attribute), 477

SWOnly *TC_PERSO_CtrlMode (class in msl.equipment.resources.thorlabs.kinesis.enums), 477*

SYNC_TRIGGER *(msl.equipment.resources.avantes.avaspec.Avantes attribute), 107*

SyncClient *(class in msl.equipment.hislip), 74*

synchronous *(msl.equipment.connection_tcpip_hislip.ConnectionTCP_IP_HISLIP property), 43*

system *(msl.equipment.connection_nidaq.ConnectionNIDAQ (msl.equipment.resources.thorlabs.kinesis.enums.TC_Dispatcher attribute), 477*

system_status_byte *TC_TH200kOhm (msl.equipment.resources.thorlabs.kinesis.enums.TC_Dispatcher attribute), 477*

(msl.equipment.resources.optronic_laboratories.ol_current_source.OLCurrentSource property), 227

T

TAGS *(msl.equipment.resources.mks_instruments.pr4000b.PR4000Bute), 477*

take_point_to_point_calibration() *(msl.equipment.resources.avantes.avaspec.Avantes attribute), 151*

(msl.equipment.resources.optronic_laboratories.ol756nch12e.OLV56 method), 223

take_point_to_point_measurement() *(msl.equipment.resources.avantes.avaspec.Avantes attribute), 151*

(msl.equipment.resources.optronic_laboratories.ol756nch12e.OLV56 method), 223

take_quick_scan_calibration() *(msl.equipment.resources.avantes.avaspec.Avantes attribute), 151*

(msl.equipment.resources.optronic_laboratories.ol756nch12e.OLV56 method), 224

take_quick_scan_measurement() *(msl.equipment.resources.avantes.avaspec.Avantes attribute), 151*

(msl.equipment.resources.optronic_laboratories.ol756nch12e.OLV56 method), 224

target_info() *(msl.equipment.resources.optronic_laboratories.ol_current_source.OLCurrentSource attribute), 227*

Task *(msl.equipment.connection_nidaq.ConnectionNIDAQ property), 33*

TBD1 *(msl.equipment.resources.thorlabs.kinesis.enums.TST_Stages attribute), 476*

TBD2 *(msl.equipment.resources.thorlabs.kinesis.enums.TST_Stages attribute), 476*

TCD_FIRST_USED_DARK_PIXEL *(msl.equipment.resources.avantes.avaspec.Avantes attribute), 151*

TCD_TOTAL_DARK_PIXELS *(msl.equipment.resources.avantes.avaspec.Avantes attribute), 151*

TCD_USED_DARK_PIXELS *(msl.equipment.resources.avantes.avaspec.Avantes attribute), 151*

TCPIP_HISLIP *(msl.equipment.constants.Interface attribute), 50*

TCube_Brushless_Motor *(msl.equipment.resources.thorlabs.kinesis.motion_control attribute), 568*

TCube_DC_Servo	TEMPERATURE_UPTO_100
(msl.equipment.resources.thorlabs.kinesis.motion_controller.MotionControllerResources.picotech.picoscope.enums.PS4000Attribute), 568	(attribute), 271
TCube_Inertial_Motor	TEMPERATURE_UPTO_130
(msl.equipment.resources.thorlabs.kinesis.motion_controller.MotionControllerResources.picotech.picoscope.enums.PS4000Attribute), 568	(attribute), 271
TCube_LaserDiode	TEMPERATURE_UPTO_40
(msl.equipment.resources.thorlabs.kinesis.motion_controller.MotionControllerResources.picotech.picoscope.enums.PS4000Attribute), 568	(attribute), 271
TCube_LaserSource	TEMPERATURE_UPTO_70
(msl.equipment.resources.thorlabs.kinesis.motion_controller.MotionControllerResources.picotech.picoscope.enums.PS4000Attribute), 568	(attribute), 271
TCube_NanoTrak	TEMPERATURES (msl.equipment.resources.picotech.picoscope.enums.PS4000Attribute), 271
(msl.equipment.resources.thorlabs.kinesis.motion_controller.MotionController), 568	TempSensorType (class) in
TCube_Quad	(msl.equipment.resources.thorlabs.kinesis.motion_controller.MotionController.avantes.avaspec), 103
(attribute), 568	
TCube_Solenoid	TERMCHRSET (msl.equipment.vxi11.OperationFlag)
(msl.equipment.resources.thorlabs.kinesis.motion_controller.MotionController), 568	(attribute), 60
(attribute), 568	THEN (msl.equipment.resources.picotech.picoscope.enums.PS2000Attribute), 241
TCube_Stepper_Motor	<thorlabs.motioncontrol< td=""></thorlabs.motioncontrol<>
(msl.equipment.resources.thorlabs.kinesis.motion_controller.MotionController), 568	thresholdDeflection
TCube_Strain_Gauge	(msl.equipment.resources.thorlabs.kinesis structs.MOT_P
(msl.equipment.resources.thorlabs.kinesis.motion_controller.MotionController), 568	(attribute), 58
(attribute), 568	thresholdLower
TCube_TEC	(msl.equipment.resources.thorlabs.kinesis.motion_controller.MotionController.picotech.picoscope structs.PS2000Attribute), 360
(attribute), 568	
team	(msl.equipment.record_types.EquipmentRecord thresholdLower
(attribute), 81	(msl.equipment.resources.picotech.picoscope structs.PS3000Attribute), 360
TEC_ID	(msl.equipment.resources.avantes.avaspec.Avantes attribute), 363
(attribute), 108	thresholdLower
TecControlType	(msl.equipment.resources.picotech.picoscope structs.PS4000Attribute), 366
(msl.equipment.resources.avantes.avaspec), 103	thresholdLower
TECCtlr	(in module
(msl.equipment.resources.thorlabs.kinesis.messages), 565	(attribute), 365
temperature()	thresholdLower
(method), 196	(attribute), 369
temperature()	(msl.equipment.resources.raicol.raiclient.TEC
(method), 401	(attribute), 369
temperature_humidity()	(msl.equipment.resources.picotech.picoscope structs.PS6000Attribute), 370
(msl.equipment.resources.omega.ithx.iTHX thresholdLowerHysteresis	
(method), 197	(attribute), 370
temperature_humidity_dewpoint()	(msl.equipment.resources.picotech.picoscope structs.PS2000Attribute), 360
(msl.equipment.resources.omega.ithx.iTHX thresholdLowerHysteresis	
(method), 197	(attribute), 360
TEMPERATURE_SENSOR	(msl.equipment.resources.picotech.picoscope thresholds.PS4000Hysteresis
(attribute), 272	(attribute), 272

attribute), 366

thresholdLowerHysteresis *(msl.equipment.resources.picotech.picoscope structs.PS4000ATriggerChannelProperties.picoscope.structs.PS4000ATriggerChannelProperties.attribute), 365*

thresholdLowerHysteresis *(msl.equipment.resources.picotech.picoscope structs.PS5000ATriggerChannelProperties.picoscope.structs.PS5000ATriggerChannelProperties.attribute), 369*

thresholdMajor *(msl.equipment.resources.picotech.picoscope structs.PS6000ATriggerChannelProperties.picoscope.structs.PS6000ATriggerChannelProperties.attribute), 358*

thresholdMajor *(msl.equipment.resources.picotech.picoscope structs.PS2000ATriggerChannelProperties.picoscope.structs.PS2000ATriggerChannelProperties.attribute), 360*

thresholdMajor *(msl.equipment.resources.picotech.picoscope structs.PS3000ATriggerChannelProperties.picoscope.structs.PS3000ATriggerChannelProperties.attribute), 367*

thresholdMinor *(msl.equipment.resources.picotech.picoscope structs.PS4000ATriggerChannelProperties.picoscope.structs.PS4000ATriggerChannelProperties.attribute), 358*

thresholdMinor *(msl.equipment.resources.picotech.picoscope structs.PS5000ATriggerChannelProperties.picoscope.structs.PS5000ATriggerChannelProperties.attribute), 360*

thresholdMinor *(msl.equipment.resources.picotech.picoscope structs.PS5500ATriggerChannelProperties.picoscope.structs.PS5500ATriggerChannelProperties.attribute), 367*

thresholdMode *(msl.equipment.resources.picotech.picoscope structs.PS2000ATriggerChannelProperties attribute), 360*

thresholdMode *(msl.equipment.resources.picotech.picoscope structs.PS3000ATriggerChannelProperties attribute), 358*

thresholdMode *(msl.equipment.resources.picotech.picoscope structs.PS4000ATriggerChannelProperties attribute), 363*

thresholdMode *(msl.equipment.resources.picotech.picoscope structs.PS4000ATriggerChannelProperties attribute), 366*

thresholdMode *(msl.equipment.resources.picotech.picoscope structs.PS4000ATriggerChannelProperties attribute), 365*

thresholdMode *(msl.equipment.resources.picotech.picoscope structs.PS5000ATriggerChannelProperties attribute), 369*

thresholdMode *(msl.equipment.resources.picotech.picoscope structs.PS5000ATriggerChannelProperties attribute), 367*

thresholdMode *(msl.equipment.resources.picotech.picoscope structs.PS5000ATriggerChannelProperties attribute), 370*

thresholdUpper *(msl.equipment.resources.picotech.picoscope structs.PS4000ATriggerChannelProperties.picoscope.structs.PS4000ATriggerChannelProperties.attribute), 366*

thresholdUpper *(msl.equipment.resources.picotech.picoscope structs.PS5000ATriggerChannelProperties.picoscope.structs.PS5000ATriggerChannelProperties.attribute), 365*

thresholdUpper *(msl.equipment.resources.picotech.picoscope structs.PS6000ATriggerChannelProperties.picoscope.structs.PS6000ATriggerChannelProperties.attribute), 370*

thresholdUpper *(msl.equipment.resources.picotech.picoscope structs.PS2000ATriggerChannelProperties.picoscope.structs.PS2000ATriggerChannelProperties.attribute), 360*

thresholdUpper *(msl.equipment.resources.picotech.picoscope structs.PS3000ATriggerChannelProperties.picoscope.structs.PS3000ATriggerChannelProperties.attribute), 363*

thresholdUpperHysteresis *(msl.equipment.resources.picotech.picoscope structs.PS4000ATriggerChannelProperties.picoscope.structs.PS4000ATriggerChannelProperties.attribute), 366*

thresholdUpperHysteresis *(msl.equipment.resources.picotech.picoscope structs.PS5000ATriggerChannelProperties.picoscope.structs.PS5000ATriggerChannelProperties.attribute), 369*

thresholdUpperHysteresis *(msl.equipment.resources.picotech.picoscope structs.PS6000ATriggerChannelProperties.picoscope.structs.PS6000ATriggerChannelProperties.attribute), 363*

thresholdUpperHysteresis *(msl.equipment.resources.picotech.picoscope structs.PS2000ATriggerChannelProperties.picoscope.structs.PS2000ATriggerChannelProperties.attribute), 360*

thresholdUpperHysteresis *(msl.equipment.resources.picotech.picoscope structs.PS3000ATriggerChannelProperties.picoscope.structs.PS3000ATriggerChannelProperties.attribute), 363*

TIM_ButonsMode *(class in msl.equipment.resources.thorlabs.kinesis.structs), 593*

TIM_Direction *(class in msl.equipment.resources.thorlabs.kinesis.enums), 474*

TIM_JogMode *(class in msl.equipment.resources.thorlabs.kinesis.enums), 475*

TIM_Status *(class in msl.equipment.resources.thorlabs.kinesis.enums), 593*

TIME *(msl.equipment.resources.picotech.picoscope.enums.PS2000ATriggerChannelProperties attribute), 366*

TIME *(msl.equipment.resources.picotech.picoscope.structs.PS4000ATriggerChannelProperties attribute), 363*

TIME (msl.equipment.resources.picotech.picoscope.enums.PS3000HoldOffType
attribute), 269
timeStampCounter

TIME (msl.equipment.resources.picotech.picoscope.enums.PS4000HoldOffTypes.picotech.picoscope structs.PS4000HoldOffType
attribute), 279
attribute), 364

time (msl.equipment.resources.thorlabs.kinesis.structs.TimeStampTypesTrackSettleParameters in
attribute), 575
msl.equipment.resources.avantes.avaspec),

TIME_1 (msl.equipment.resources.cmi.sia3.IntegrationTime 103
attribute), 132
timeUnits (msl.equipment.resources.picotech.picoscope.structs.
TIME_100m (msl.equipment.resources.cmi.sia3.IntegrationTime attribute), 364
attribute), 132
timeUnits (msl.equipment.resources.picotech.picoscope.structs.
TIME_100u (msl.equipment.resources.cmi.sia3.IntegrationTime attribute), 368
attribute), 132
TLI_DeviceInfo (class in
TIME_10m (msl.equipment.resources.cmi.sia3.IntegrationTime msl.equipment.resources.thorlabs.kinesis.structs),
attribute), 132
570

TIME_1m (msl.equipment.resources.cmi.sia3.IntegrationTime msl.equipment.resources.thorlabs.kinesis.structs),
attribute), 132
TLI_HardwareInformation (class in
TIME_2 (msl.equipment.resources.cmi.sia3.IntegrationTime 571
attribute), 132
to_bytes() (in module msl.equipment.utils), 600

TIME_200m (msl.equipment.resources.cmi.sia3.IntegrationTime msl.equipment.record_types.CalibrationRecord
attribute), 132
method), 82

TIME_20m (msl.equipment.resources.cmi.sia3.IntegrationTime msl.equipment.record_types.ConnectionRecord
attribute), 132
method), 86

TIME_500m (msl.equipment.resources.cmi.sia3.IntegrationTime msl.equipment.record_types.EquipmentRecord
attribute), 132
method), 82

TIME_50u (msl.equipment.resources.cmi.sia3.IntegrationTime msl.equipment.record_types.MaintenanceRecord
attribute), 131
method), 85

time_since_last_msg_received() to_dict() (msl.equipment.record_types.MeasurandRecord
(msl.equipment.resources.thorlabs.kinesis.benchtop_stepper_motor), 80
method), 430

time_since_last_msg_received() to_dict() (msl.equipment.record_types.Record
(msl.equipment.resources.thorlabs.kinesis.filter_json), 81
method), 482

time_since_last_msg_received() to_json() (msl.equipment.record_types.ConnectionRecord
(msl.equipment.resources.thorlabs.kinesis.integrated_stepper_motors), 86
method), 505

time_since_last_msg_received() to_json() (msl.equipment.record_types.EquipmentRecord
(msl.equipment.resources.thorlabs.kinesis.kcube_solenoid), 81
method), 529

time_since_last_msg_received() to_json() (msl.equipment.record_types.MaintenanceRecord
(msl.equipment.resources.thorlabs.kinesis.kcube_solenoid), 85
method), 529

time_since_last_msg_received() to_json() (msl.equipment.record_types.MeasurandRecord
(msl.equipment.resources.thorlabs.kinesis.kcube_solenoid), 85
method), 538

time_since_last_msg_received() to_json() (msl.equipment.record_types.Record
(msl.equipment.resources.thorlabs.kinesis.kcube_solenoid), 87
method), 538

time_since_last_msg_received() to_json() (msl.equipment.record_types.RecordDict
(msl.equipment.resources.thorlabs.kinesis.kcube_solenoid), 87
method), 564

timeout (msl.equipment.connection_message_based_Conversion) (msl.equipment.resources.thorlabs.kinesis.motion
property), 30
static method), 570

timeout (msl.equipment.connection_prologix.ConnectionPrologix), 83
property), 34

timeout (msl.equipment.resources.thorlabs.kinesis.stepper_motors_button_record_types.ConnectionRecord
attribute), 582
method), 86

timer() (msl.equipment.resources.energetiq.eq99.E99xml) (msl.equipment.record_types.EquipmentRecord
method), 100

method), 82
`to_xml()` (*msl.equipment.record_types.MaintenanceRecord*)
method), 85
`to_xml()` (*msl.equipment.record_types.MeasurandRecord*)
method), 84
`to_xml()` (*msl.equipment.record_types.Record*)
method), 87
`to_xml()` (*msl.equipment.record_types.RecordDict*)
method), 87
TPZ_IOSettings (class in *msl.equipment.resources.thorlabs.kinesis.structs*)
attribute), 587
`trace()` (*msl.equipment.resources.bentham.benhw3*)
method), 129
`transitTime` (*msl.equipment.resources.thorlabs.kinesis.structs*)
attribute), 582
TRIANGLE (*msl.equipment.resources.picotech.picoscope.enums*)
attribute), 246
TRIANGLE (*msl.equipment.resources.picotech.picoscope.enums*)
attribute), 237
TRIANGLE (*msl.equipment.resources.picotech.picoscope.enums*)
attribute), 263
TRIANGLE (*msl.equipment.resources.picotech.picoscope.enums*)
attribute), 253
TRIANGLE (*msl.equipment.resources.picotech.picoscope.enums*)
attribute), 285
TRIANGLE (*msl.equipment.resources.picotech.picoscope.enums*)
attribute), 275
TRIANGLE (*msl.equipment.resources.picotech.picoscope.enums*)
attribute), 309
TRIANGLE (*msl.equipment.resources.picotech.picoscope.enums*)
attribute), 299
TRIANGLE (*msl.equipment.resources.picotech.picoscope.enums*)
attribute), 318
TRIANGLE_MAX_FREQUENCY
(msl.equipment.resources.picotech.picoscope.enums)
attribute), 345
TRIANGLE_MAX_FREQUENCY
(msl.equipment.resources.picotech.picoscope.enums)
attribute), 348
TRIANGLE_MAX_FREQUENCY
(msl.equipment.resources.picotech.picoscope.ps4000anRla
attribute), 352
TRIANGLE_MAX_FREQUENCY
(msl.equipment.resources.picotech.picoscope.ps5000(Rid
attribute), 354
TRIANGLE_MAX_FREQUENCY
(msl.equipment.resources.picotech.picoscope.ps5000(Rid
attribute), 355
TRIANGLE_MAX_FREQUENCY
(msl.equipment.resources.picotech.picoscope.ps6000(Rid
attribute), 356
attribute), 357
trig1DiffThreshold
(msl.equipment.resources.thorlabs.kinesis structs.QD_KP
trig1Mode (*msl.equipment.resources.thorlabs.kinesis structs.QD_*
attribute), 590
trig1Polarity (*msl.equipment.resources.thorlabs.kinesis structs.QD_*
attribute), 590
trig1SumMax (*msl.equipment.resources.thorlabs.kinesis structs.QD_*
attribute), 590
trig1SumMin (*msl.equipment.resources.thorlabs.kinesis structs.QD_*
attribute), 590
trig2DiffThreshold
(msl.equipment.resources.thorlabs.kinesis structs.QD_KP
trig2Mode (*msl.equipment.resources.thorlabs.kinesis structs.QD_*
attribute), 590
trig2Polarity (*msl.equipment.resources.thorlabs.kinesis structs.QD_*
attribute), 590
trig2SumMax (*msl.equipment.resources.thorlabs.kinesis structs.QD_*
attribute), 590
trig2SumMin (*msl.equipment.resources.thorlabs.kinesis structs.QD_*
attribute), 590
Trig_Disabled (*msl.equipment.resources.thorlabs.kinesis enums*)
Trig_High (*msl.equipment.resources.thorlabs.kinesis enums.KIM_*
Trig_In_GPI (*msl.equipment.resources.thorlabs.kinesis enums.KIM_*
Trig_InAbsoluteMove
Trig_InRevMove
(msl.equipment.resources.thorlabs.kinesis enums.KIM_Tr
attribute), 455
Trig_InRelMove
(msl.equipment.resources.thorlabs.kinesis enums.KIM_Tr
attribute), 455
Trig_OutAtScope2000A
(msl.equipment.resources.thorlabs.kinesis enums.KIM_Tr
attribute), 455
Trig_OutAtScope5000A
(msl.equipment.resources.thorlabs.kinesis enums.KIM_Tr
attribute), 456
Trig_OutAtEitherLimit
(msl.equipment.resources.picotech.picoscope.ps4000anRla
attribute), 455
Trig_OutAtFwdLimit
(msl.equipment.resources.picotech.picoscope.ps5000(Rid
attribute), 455
Trig_OutAtMaxVelocity
(msl.equipment.resources.picotech.picoscope.ps5000(Rid
attribute), 455
Trig_OutAtRevLimit
(msl.equipment.resources.picotech.picoscope.ps6000(Rid
attribute), 455

attribute), 455
 Trig_Out_GP0 (msl.equipment.resources.thorlabs.kinesis.enums.KIM1ModeTriggerModes
 attribute), 455
 Trig_Out_InMotion
 (msl.equipment.resources.thorlabs.kinesis.enums.KIM1ModeTriggerModes
 attribute), 455
 Trig_Out_PosStepBoth
 (msl.equipment.resources.thorlabs.kinesis.enums.KIM1ModeTriggerModes
 attribute), 455
 Trig_Out_PosStepFwd
 (msl.equipment.resources.thorlabs.kinesis.enums.KIM1ModeTriggerModes
 attribute), 455
 Trig_Out_PosStepRev
 (msl.equipment.resources.thorlabs.kinesis.enums.KIM1ModeTriggerModes
 attribute), 455
 Trigger (class in msl.equipment.hislip), 66
 Trigger (msl.equipment.hislip.MessageType at-
 tribute), 57
 TRIGGER (msl.equipment.resources.picotech.picoscope.TriggerPolarityPicoStringValue
 attribute), 293
 trigger() (msl.equipment.connection_gpib.ConnectionGPIBAttribute), 588
 method), 28
 trigger() (msl.equipment.connection_tcpip_hislip.ConnectionTCPPIPHISLIPResources.thorlabs.kinesis.structs.KSC_Trigger1Mode
 method), 44
 trigger() (msl.equipment.connection_tcpip_vxi11.ConnectionTCPPIPVXI11Trigger1Mode
 method), 47
 trigger() (msl.equipment.hislip.SyncClient
 method), 76
 Trigger1Mode (msl.equipment.resources.thorlabs.kinesis.structs.KSG_Trigger1ModeConfigPicotech.picoscope.enums.PS400
 attribute), 583
 Trigger1Mode (msl.equipment.resources.thorlabs.kinesis.structs.KSG_Trigger1ModeConfigPicotech.picoscope.enums.PS400
 attribute), 587
 Trigger1Mode (msl.equipment.resources.thorlabs.kinesis.structs.KSG_Trigger1ModeConfigPicotech.picoscope.enums.PS400
 attribute), 588
 Trigger1Mode (msl.equipment.resources.thorlabs.kinesis.structs.KSG_Trigger1ModeConfigPicotech.picoscope.enums.PS400
 attribute), 591
 Trigger1Mode (msl.equipment.resources.thorlabs.kinesis.structs.KSG_Trigger1ModeConfigPicotech.picoscope.enums.PS400
 attribute), 592
 Trigger1Polarity
 (msl.equipment.resources.thorlabs.kinesis.structs.KSG_Trigger1PolarityConfigState
 attribute), 583
 Trigger1Polarity
 (msl.equipment.resources.thorlabs.kinesis.structs.KSG_Trigger1PolarityConfigfig
 attribute), 587
 Trigger1Polarity
 (msl.equipment.resources.thorlabs.kinesis.structs.KSG_Trigger1PolarityConfigDelay
 attribute), 588
 Trigger1Polarity
 (msl.equipment.resources.thorlabs.kinesis.structs.KSG_Trigger1PolarityConfigipConfig
 attribute), 591
 Trigger1Polarity

 (msl.equipment.resources.thorlabs.kinesis.structs.KSG_Trigger1PolarityConfigresources.picotech.picoscope.enums.PS400
 attribute), 294
 Trigger2Mode (msl.equipment.resources.thorlabs.kinesis.structs.KSG_Trigger2Mode
 attribute), 583
 Trigger2Mode (msl.equipment.resources.thorlabs.kinesis.structs.KSG_Trigger2Mode
 attribute), 587
 Trigger2Mode (msl.equipment.resources.thorlabs.kinesis.structs.KSG_Trigger2Mode
 attribute), 588
 Trigger2Mode (msl.equipment.resources.thorlabs.kinesis.structs.KSG_Trigger2Mode
 attribute), 592
 Trigger2Mode (msl.equipment.resources.thorlabs.kinesis.structs.KSG_Trigger2Mode
 attribute), 593
 Trigger2Polarity
 (msl.equipment.resources.thorlabs.kinesis.structs.KNA_Trigger2Polarity
 attribute), 583
 Trigger2Polarity
 (msl.equipment.resources.thorlabs.kinesis.structs.KNA_Trigger2Polarity
 attribute), 587
 Trigger2PolarityPicoStringValue
 (msl.equipment.resources.thorlabs.kinesis.structs.KPZ_Trigger2Polarity
 attribute), 293
 Trigger2Polarity
 (msl.equipment.resources.thorlabs.kinesis.structs.KSG_Trigger2Polarity
 attribute), 593
 TRIGGER_ARRAY_OF_BLOCK_CONDITIONS
 TRIGGER_AUTO_TRIGGER_MS
 TRIGGER_AUXIO_OUTPUT_ENABLED
 TRIGGER_AUXIO_TRIGGER_MS
 TRIGGER_CONDITION_SOURCEFig
 TRIGGER_CONDITION_STATE
 TRIGGER_CONDITIONSfig
 TRIGGER_DELAY_MS
 TRIGGER_DELAY_MS
 TRIGGER_DIRECTION

(<i>msl.equipment.resources.picotech.picoscope</i> .TriggerPS4000APicoStringValue attribute), 294	<i>in</i> <i>msl.equipment.resources.thorlabs.fwxx2c</i> , 594
TRIGGER_DIRECTION_CHANNEL	
(<i>msl.equipment.resources.picotech.picoscope</i> .TriggerPS4000APicoStringValue attribute), 294	<i>in</i> <i>msl.equipment.resources.thorlabs.kinesis structs.KMOT_</i> attribute), 583
TRIGGER_DIRECTION_DIRECTION	
(<i>msl.equipment.resources.picotech.picoscope</i> .TriggerPS4000APicoStringValue attribute), 294	<i>in</i> <i>msl.equipment.resources.thorlabs.kinesis structs.KMOT_</i> attribute), 583
TRIGGER_NO_OF_BLOCK_CONDITIONS	
(<i>msl.equipment.resources.picotech.picoscope</i> .TriggerPS4000APicoStringValue attribute), 294	<i>in</i> <i>msl.equipment.resources.thorlabs.kinesis structs.KMOT_</i> attribute), 584
TRIGGER_NO_OF_CONDITIONS	
(<i>msl.equipment.resources.picotech.picoscope</i> .TriggerPS4000APicoStringValue attribute), 294	<i>in</i> <i>msl.equipment.resources.thorlabs.kinesis structs.KMOT_</i> attribute), 584
TRIGGER_NO_OF_DIRECTIONS	
(<i>msl.equipment.resources.picotech.picoscope</i> .TriggerPS4000APicoStringValue attribute), 294	<i>in</i> <i>msl.equipment.resources.thorlabs.kinesis structs.KMOT_</i> attribute), 584
TRIGGER_PROPERTIES	
(<i>msl.equipment.resources.picotech.picoscope</i> .triggerPS4000APeqSpringValue attribute), 293	<i>in</i> <i>msl.equipment.resources.picotech.picoscope structs</i> attribute), 364
TRIGGER_PROPERTIES_CHANNEL	triggerTime (<i>msl.equipment.resources.picotech.picoscope</i> .structs attribute), 293
	TriggerType (class <i>in</i> <i>msl.equipment.resources.avantes.avaspec</i>),
TRIGGER_PROPERTIES_THRESHOLD_LOWER	msl.equipment.resources.picotech.picoscope.enums.PS4000APicoStringValue attribute), 293
	TRUE (<i>msl.equipment.resources.picotech.picoscope</i> .enums.PS2000A attribute), 239
TRIGGER_PROPERTIES_THRESHOLD_LOWER_HYSTESIS	attribute), 250
	TRUE (<i>msl.equipment.resources.picotech.picoscope</i> .enums.PS2000A attribute), 239
TRIGGER_PROPERTIES_THRESHOLD_MODE	TRUE (<i>msl.equipment.resources.picotech.picoscope</i> .enums.PS3000A attribute), 293
	TRUE (<i>msl.equipment.resources.picotech.picoscope</i> .enums.PS3000A attribute), 267
TRIGGER_PROPERTIES_THRESHOLD_UPPER	attribute), 256
	TRUE (<i>msl.equipment.resources.picotech.picoscope</i> .enums.PS4000A attribute), 293
TRIGGER_PROPERTIES_THRESHOLD_UPPER_HYSTESIS	attribute), 256
	TRUE (<i>msl.equipment.resources.picotech.picoscope</i> .enums.PS5000A attribute), 293
TRIGGER_RAW	(<i>msl.equipment.resources.picotech.picoscope</i> .enum.PS6000SigGenTrigSource attribute), 320
	TRUE (<i>msl.equipment.resources.picotech.picoscope</i> .enums.PS5000T attribute), 320
trigger_within_pre_trigger_samples()	attribute), 302
	TRUE (<i>msl.equipment.resources.picotech.picoscope</i> .methods.PS6000APicoStringValue method), 341
triggerIndex	attribute), 322
	attribute), 322
TriggerIntervalFwd	attribute), 368
	attribute), 473
TriggerIntervalRev	TSG_Hub_Analogue_Modes (class <i>in</i> <i>msl.equipment.resources.thorlabs.kinesis</i> .structs.KMOT_eTriggerFwd attribute), 473
	<i>resources.thorlabs.kinesis</i> .enums.TSG_Hub_Analogue_Modes attribute), 473

TSG_HubChannel11
 (msl.equipment.resources.thorlabs.kinesis.enums.**TSG_HubChannel11ModeLockInfoResponse**
 attribute), 473
 tribute), 63

TSG_HubChannel12
 (msl.equipment.resources.thorlabs.kinesis.enums.**TSG_HubAnalogue_Modes**
 attribute), 473
 type (msl.equipment.hislip.AsyncLockResponse
 type (msl.equipment.hislip.AsyncMaximumMessageSize
 attribute), 64

TSG_IOSettings
 (class in attribute), 66
 msl.equipment.resources.thorlabs.kinesis.str
 type (msl.equipment.hislip.AsyncMaximumMessageSizeResponse
 attribute), 67

592

TSG_Position (msl.equipment.resources.thorlabs.kin
 type (**msl.equipment.hislip.RemoteLocalControl**
 attribute), 473
 attribute), 64

TSG_Undefined (msl.equipment.resources.thorlabs.kin
 type (**msl.equipment.hislip.RemoteLocalResponse**
 attribute), 473
 attribute), 65

TSG_Voltage (msl.equipment.resources.thorlabs.kin
 type (**msl.equipment.hislip.AsyncStartTLS**
 attribute), 473
 attribute), 69

TST_Stages
 (class in type (msl.equipment.hislip.AsyncStartTLSResponse
 msl.equipment.resources.thorlabs.kinesis.enums), attribute), 70
 476
 type (msl.equipment.hislip.AsyncStatusQuery at-

turn_off() (msl.equipment.resources.aim_tti.mx_series.**MXSe**
 method), 93
 type (msl.equipment.hislip.AsyncStatusResponse

turn_off() (msl.equipment.resources.optronic_laboratories.**OLCurrentSource**
 method), 227
 type (msl.equipment.hislip.AuthenticationExchange
 attribute), 72

turn_off_multi()
 (msl.equipment.resources.aim_tti.mx_series.**MXSer**
 method), 94
 type (msl.equipment.hislip.AuthenticationResult
 attribute), 73

turn_on() (msl.equipment.resources.aim_tti.mx_series.**MXSe**
 method), 93
 type (msl.equipment.hislip.AuthenticationStart
 attribute), 72

turn_on() (msl.equipment.resources.optronic_laboratory.**msl.equipment.hislip.DeviceClearInfo**
 method), 227
 type (msl.equipment.hislip.DataEnd attribute), 62

turn_on_multi()
 (msl.equipment.resources.aim_tti.mx_series.**MXSeries**
 method), 93
 type (msl.equipment.hislip.DeviceClearAcknowledge
 attribute), 66

TWELVE (msl.equipment.resources.thorlabs.fwxx2c.**FilterCount**
 attribute), 594
 type (msl.equipment.hislip.EndTLS attribute), 70

TWO (msl.equipment.constants.StopBits attribute),
 51
 type (msl.equipment.hislip.ErrorMessage attribute), 61

type (msl.equipment.hislip.AsyncDeviceClear attribute), 65
 type (msl.equipment.hislip.FatalErrorMessage attribute), 60

type (msl.equipment.hislip.AsyncDeviceClearAcknowle
 attribute), 65
 type (msl.equipment.hislip.GetDescriptors attribute), 67

type (msl.equipment.hislip.AsyncEndTLS attribute), 71
 type (msl.equipment.hislip.GetDescriptorsResponse attribute), 67

type (msl.equipment.hislip.AsyncEndTLSResponse attribute), 71
 type (msl.equipment.hislip.GetSaslMechanismList attribute), 72

type (msl.equipment.hislip.AsyncInitialize attribute), 68
 type (msl.equipment.hislip.GetSaslMechanismListResponse attribute), 72

type (msl.equipment.hislip.AsyncInitializeResponse attribute), 68
 type (msl.equipment.hislip.Initialize attribute), 61

type (msl.equipment.hislip.AsyncLock attribute), 62
 type (msl.equipment.hislip.InitializeResponse attribute), 61

type (msl.equipment.hislip.AsyncLockInfo attribute), 62
 type (msl.equipment.hislip.Message attribute), 59

type (msl.equipment.hislip.AsyncLockInfo attribute), 69
 type (msl.equipment.hislip.StartTLS attribute), 69

type (*msl.equipment.hislip.Trigger attribute*), 66 UnitcmA (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType*)
type (*msl.equipment.record_types.MeasurandRecord attribute*), 170 UnitcmA (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitTypes*)
 attribute), 84
type (*msl.equipment.resources.thorlabs.kinesis structs.TLI_HandshakeInformation attribute*), 571 UnitcmW (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType*)
typeID (*msl.equipment.resources.thorlabs.kinesis structs.TLI_dedicatedInfo attribute*), 571 UnitcmW (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitTypes*)
types (*msl.equipment.connection_nidaq.ConnectionNIDAQ attribute*), 165 Unitcv (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType*)
 property), 33 attribute), 170
U
ULimit (*msl.equipment.resources.nkt.nktpdll.ParameterSetType attribute*), 165 Unitcv (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitTypes*)
 attribute), 163 attribute), 170
underOrOverRead (*msl.equipment.resources.thorlabs.kinesis structs.NT_dedicatedInfo attribute*), 586 UnitdB (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType*)
 attribute), 165
underOrOverRead (*msl.equipment.resources.thorlabs.kinesis structs.NT_dedicatedInfo attribute*), 579 attribute), 170
UNIDENTIFIED (*msl.equipment.hislip.ErrorType attribute*), 58 UnitdBm (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType*)
unique_key (*msl.equipment.record_types.EquipmentRecord attribute*), 170 attribute), 165
unit (*msl.equipment.record_types.MeasurandRecord attribute*), 165 Unitdc (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType*)
 attribute), 84 attribute), 165
Unit (*msl.equipment.resources.nkt.nktpdll.ParameterSetType attribute*), 170 Unitdlm (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType*)
 attribute), 163 attribute), 170
unit () (*msl.equipment.resources.greisinger.gmh3000.GMH3000Attribute*), 166 UnitdmA (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType*)
 method), 150
UNIT_INFO (*msl.equipment.resources.picotech.picoscope.enum_PSA4000AMetaType attribute*), 286 UnitdmA (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType*)
 attribute), 165
UnitA (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType*), 165 UnitdmW (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType*)
 attribute), 169
UnitA (*msl.equipment.resources.nkt.nktpdll.ParamSetUnitType*), 170 UnitdmW (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType*)
 attribute), 165
UnitcB (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType*), 165 Unitdnm (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType*)
 attribute), 170
UnitcB (*msl.equipment.resources.nkt.nktpdll.ParamSetUnitType*), 170 Unitdnm (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType*)
 attribute), 165
UnitcBm (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType*), 165 Unitdpm (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType*)
 attribute), 170
UnitcBm (*msl.equipment.resources.nkt.nktpdll.ParamSetUnitType*), 170 Unitdpm (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType*)
 attribute), 165
UnitcC (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType*), 165 Unitdv (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType*)
 attribute), 170
UnitcC (*msl.equipment.resources.nkt.nktpdll.ParamSetUnitType*), 170 Unitdv (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType*)
 attribute), 165
Unitclm (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType*), 165 Unitlm (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType*)
 attribute), 170
Unitclm (*msl.equipment.resources.nkt.nktpdll.ParamSetUnitType*), 170 Unitlm (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType*)
 attribute), 166

attribute), 165

UnitImA (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitTypes*)
attribute), 169

UnitImA (*msl.equipment.resources.nkt.nktpdll.ParamSetUnitType*)
attribute), 165

UnitImB (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType*)
attribute), 170

UnitImB (*msl.equipment.resources.nkt.nktpdll.ParamSetUnitType*)
attribute), 165

UnitImBm (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType*)
attribute), 170

UnitImBm (*msl.equipment.resources.nkt.nktpdll.ParamSetUnitType*)
attribute), 165

UnitImC (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType*)
attribute), 170

UnitImC (*msl.equipment.resources.nkt.nktpdll.ParamSetUnitType*)
attribute), 165

UnitImLm (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType*)
attribute), 170

UnitImLm (*msl.equipment.resources.nkt.nktpdll.ParamSetUnitType*)
attribute), 166

UnitImV (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType*)
attribute), 169

UnitImV (*msl.equipment.resources.nkt.nktpdll.ParamSetUnitType*)
attribute), 164

UnitImW (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType*)
attribute), 170

UnitImW (*msl.equipment.resources.nkt.nktpdll.ParamSetUnitType*)
attribute), 165

UnitIm (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType*)
attribute), 170

UnitIm (*msl.equipment.resources.nkt.nktpdll.ParamSetUnitType*)
attribute), 165

UnitNone (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitTypes*)
attribute), 169

UnitNone (*msl.equipment.resources.nkt.nktpdll.ParamSetUnitType*)
attribute), 164

UnitPerCent (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType*)
attribute), 170

UnitPerCent (*msl.equipment.resources.nkt.nktpdll.ParamSetUnitType*)
attribute), 165

UnitPerMille (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType*)
attribute), 170

UnitPerMille (*msl.equipment.resources.nkt.nktpdll.ParamSetUnitType*)
attribute), 165

UnitIm (*msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType*)
attribute), 170

UnitIm (*msl.equipment.resources.nkt.nktpdll.ParamSetUnitType*)
attribute), 165

UnitRPM (*msl.equipment.resources.nkt.nktpdll.ParamSetUnitType*)
attribute), 170

UnitRPM (*msl.equipment.resources.nkt.nktpdll.ParamSetUnitType*)
attribute), 165

UNITS (*msl.equipment.resources.mks_instruments.pr4000b.PR4000b*)
attribute), 478

UnitType (class) in
msl.equipment.resources.thorlabs.kinesis.enums,
attribute), 169

UnitType (class) in
msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType,
attribute), 165

UnitType (class) in
msl.equipment.resources.nkt.nktpdll.NKT.ParamSetUnitType,
attribute), 169

UnitType (class) in
msl.equipment.constants.Backend,
attribute), 50

UnitType (class) in
msl.equipment.resources.avantes.avaspec.Avantes.DeviceStatus,
attribute), 108

UnitType (class) in
msl.equipment.resources.avantes.avaspec.DeviceStatus,
attribute), 97

UnitType (class) in
msl.equipment.resources.picotech.picoscope.enums.PS4000b,
attribute), 272

UnitType (class) in
msl.equipment.resources.nkt.nktpdll,
attribute), 165

unlock() (*msl.equipment.connection_tcpip_hislip.ConnectionTCP*)
attribute), 44

unlock() (*msl.equipment.connection_tcpip_vx11.ConnectionTCP*)
attribute), 164

unlock() (*msl.equipment.resources.mks_instruments.pr4000b.PR4000b*)
attribute), 165

unpack_opaque()
attribute), 162

RPCClient static
method), 604

ParsedSerialType (*msl.equipment.resources.thorlabs.kinesis.structs.KSC_MM*)
attribute), 591

MOT_Bu (*msl.equipment.resources.thorlabs.kinesis.structs.MOT_Bu*)
attribute), 582

NT_IOSer (*msl.equipment.resources.thorlabs.kinesis.structs.NT_IOSer*)
attribute), 579

KNA_Ti (*msl.equipment.resources.thorlabs.kinesis.structs.KNA_Ti*)
attribute), 587

unused2 (*msl.equipment.resources.thorlabs.kinesis structs.KNATriggerConfig*
attribute), 587 US (*msl.equipment.resources.picotech.picoscope.enums.PS3000Tim*

UP (*msl.equipment.resources.picotech.picoscope.enums.PS2000AShutterType*
attribute), 245 US (*msl.equipment.resources.picotech.picoscope.enums.PS4000ATi*

UP (*msl.equipment.resources.picotech.picoscope.enums.PS2000AShutterType*
attribute), 237 US (*msl.equipment.resources.picotech.picoscope.enums.PS4000Tim*

UP (*msl.equipment.resources.picotech.picoscope.enums.PS3000AShutterType*
attribute), 263 US (*msl.equipment.resources.picotech.picoscope.enums.PS5000ATi*

UP (*msl.equipment.resources.picotech.picoscope.enums.PS4000AShutterType*
attribute), 285 US (*msl.equipment.resources.picotech.picoscope.enums.PS5000Tim*

UP (*msl.equipment.resources.picotech.picoscope.enums.PS4000AShutterType*
attribute), 274 US (*msl.equipment.resources.picotech.picoscope.enums.PS6000Tim*

UP (*msl.equipment.resources.picotech.picoscope.enums.PS5000AShutterType*
attribute), 308 USB5216 (*msl.equipment.resources.avantes.avaspec.Avantes.Interface*

UP (*msl.equipment.resources.picotech.picoscope.enums.PS5000AShutterType*
attribute), 299 USB5216 (*msl.equipment.resources.avantes.avaspec.InterfaceType*

UP (*msl.equipment.resources.picotech.picoscope.enums.PS6000AShutterType*
attribute), 318 USB7010 (*msl.equipment.resources.avantes.avaspec.Avantes.Interface*

update_eth_devices()
(*msl.equipment.resources.avantes.avaspec.USB7010* (*msl.equipment.resources.avantes.avaspec.InterfaceType*
method), 127 attribute), 98

update_usb_devices()
(*msl.equipment.resources.avantes.avaspec.Avantes* attribute), 108
method), 127 USB_AVAILABLE (*msl.equipment.resources.avantes.avaspec.Avante*

UPDOWN (*msl.equipment.resources.picotech.picoscope.enums.R82000ASleepType*
attribute), 245 USB_IN_USE_BY_APPLICATION

UPDOWN (*msl.equipment.resources.picotech.picoscope.enums.R82000ASleepType*
attribute), 237 resources.avantes.avaspec.Avantes.DeviceStatus
attribute), 108

UPDOWN (*msl.equipment.resources.picotech.picoscope enums.R84000ASleepType*
attribute), 263 resources.avantes.avaspec.DeviceStatus
attribute), 108

UPDOWN (*msl.equipment.resources.picotech.picoscope.enums.R84000ASleepType*
attribute), 285 USB_IN_USE_BY_OTHER

UPDOWN (*msl.equipment.resources.picotech.picoscope enums.R84000ASleepType*
attribute), 274 resources.avantes.avaspec.Avantes.DeviceStatus
attribute), 108

UPDOWN (*msl.equipment.resources.picotech.picoscope enums.R85000ASleepType*
attribute), 308 USB_IN_USE_BY_OTHER
attribute), 108

UPDOWN (*msl.equipment.resources.picotech.picoscope enums.R85000ASleepType*
attribute), 299 USB_VERSION (*msl.equipment.resources.picotech.picoscope.enums*

UPDOWN (*msl.equipment.resources.picotech.picoscope enums.R86000ASleepType*
attribute), 318 USB_VERSION (*msl.equipment.resources.picotech.picoscope.enums*

upLimit (*msl.equipment.resources.thorlabs.kinesis structs.KNATRangeParameters*
attribute), 586 USB_VERSION (*msl.equipment.resources.picotech.picoscope.enums*

upLimit (*msl.equipment.resources.thorlabs.kinesis structs.NTATRangeParameters*
attribute), 578 USBMINI (*msl.equipment.resources.avantes.avaspec.Avantes.Interface*

UpperLimit (*msl.equipment.resources.thorlabs.kinesis structs.KS64TriggerConfig*
attribute), 593 USBMINI (*msl.equipment.resources.avantes.avaspec.InterfaceType*

US (*msl.equipment.resources.picotech.picoscope.enums.PS2000ATbndly*, 98
attribute), 245 use_group() (*msl.equipment.resources.bentham.benhw32.Bentha*

US (*msl.equipment.resources.picotech.picoscope.enums.PS2000EthadUnit*, 139
attribute), 234 use_high_res_adc()

US (*msl.equipment.resources.picotech.picoscope.enums.PS3000ATeqdlimit*, 137 resources.avantes.avaspec.Avantes

method), 127
user_defined(*msl.equipment.record_types.EquipmentRecord*,
attribute), 81
USER_ID_LEN(*msl.equipment.resources.avantes.avaspec.AvaSpecParams*,
attribute), 106
UserFriendlyName
(msl.equipment.resources.avantes.avaspec.Avantes.Avante
AttributeType), 380
UserFriendlyName
(msl.equipment.resources.avantes.avaspec.Avante
AttributeType), 99
uses_pid_loop_encoding()
(msl.equipment.resources.thorlabs.kinesis.benchtop_steppe
motor.BenchtopStepperMotor
method), 430
uses_pid_loop_encoding()
(msl.equipment.resources.thorlabs.kinesis.kine
steppe(motor.BenchtopStepperMotor, eq99.EQ99
method), 564
utils(*msl.equipment.connection_nidaq.ConnectionNIDAQ*
info (in module msl.equipment), 17
property), 33
V
valid_det_num()
(msl.equipment.resources.princeton_instruments.arc_instrument
PrincetonInstruments
method), 379
valid_enum()
(msl.equipment.resources.princeton_instrumen
msl.equipment.instruments.drives.motors.kinesis.structs.NT_HVC
static method), 379
valid_filter_enum()
(msl.equipment.resources.princeton_instruments.arc_instrument
PrincetonInstruments
static method), 380
valid_mono_enum()
(msl.equipment.resources.princeton_instruments.arc_instrument
PrincetonInstruments
static method), 379
valid_readout_enum()
(msl.equipment.resources.princeton_instruments.arc_instrument
PrincetonInstruments
static method), 380
value()
(msl.equipment.config.Config method), 19
value()
(msl.equipment.resources.greisinger.gmh3000.GMH3000
property), 232
method), 150
VARIANT_INFO(*msl.equipment.resources.picotech.picoscope*,
attribute), 233
VARIANT_INFO(*msl.equipment.resources.picotech.picoscope*,
enums.PS2000Info
attribute), 235
VARIANT_INFO(*msl.equipment.resources.picotech.picoscope*,
ps2000Info
attribute), 254
VELOCITY(*msl.equipment.resources.thorlabs.kinesis.vxi11*,
attribute), 478
velocity(*msl.equipment.resources.thorlabs.kinesis.vxi11*,
attribute), 572
velocity(*msl.equipment.resources.thorlabs.kinesis.structs.MOTThrottle*,
attribute), 582
velocityFeedForward
(msl.equipment.resources.thorlabs.kinesis.structs.MOT_B
attribute), 575
ver()
(msl.equipment.resources.princeton_instruments.arc_instru
ment), 380
version(*msl.equipment.connection_nidaq.ConnectionNIDAQ*
property), 33
version(*msl.equipment.connection_gpib.ConnectionGPIB*
method), 28
version(*msl.equipment.connection_prologix.ConnectionProlog*
method), 36
version(*msl.equipment.resources.bentham.benhw64.Bentham*
method), 131
VerticalStage
(msl.equipment.resources.thorlabs.kinesis.motion_control
VerticalStage
verticalComponent
(msl.equipment.instruments.drives.motors.kinesis.structs.NT_HVC
attribute), 577
voltage_offset
(msl.equipment.resources.picotech.picoscope.channel.Pico
scope.channel.Picoscope.property), 231
voltage_range(*msl.equipment.resources.picotech.picoscope.chan*
nelle.Picoscope.property), 231
VoltageAdjustRate
(msl.equipment.resources.thorlabs.kinesis.structs.KPZ_M
ethod), 380
VoltageStep(*msl.equipment.resources.thorlabs.kinesis.structs.KPZ_M*
attribute), 588
volts(*msl.equipment.resources.picotech.picoscope.channel.Pico*
scope.channel.Picoscope.volts_per_adu), 232
volts_per_adu(*msl.equipment.resources.picotech.picoscope.cha*
nnel.Picoscope.volts_per_adu), 232
VXIClient (*class in msl.equipment.vxi11*), 605
W
WAVEFORM_PARAMETERS(*msl.equipment.resources.optosigma.shot702.SHOT702*,
method), 205
wait_for_message()
(msl.equipment.resources.thorlabs.kinesis.benchtop_steppe
motor.BenchtopStepperMotor
method), 28

(*msl.equipment.resources.thorlabs.kinesis.filter_flippattFilter*)^{filter}
method), 483
WHITE_NOISE (*msl.equipment.resources.picotech.picoscope.enums*
attribute), 300

wait_for_message()
(*msl.equipment.resources.thorlabs.kinesis.iWHITENOISE*)^{WHITENOISE}
method), 505
msl.equipment.resources.picotech.picoscope.enums
attribute), 246

wait_for_message()
(*msl.equipment.resources.thorlabs.kinesis.kcube_dc_attribute*)^{KCubeDC}
method), 529
msl.equipment.resources.picotech.picoscope.enums
attribute), 280

wait_for_message()
(*msl.equipment.resources.thorlabs.kinesis.kcube_sphericalStepperMotor*)^{WHITENOISE}
method), 538
msl.equipment.resources.picotech.picoscope.enums
attribute), 274

wait_for_message()
(*msl.equipment.resources.thorlabs.kinesis.kcube_stepMotor*)^{WHITENOISE}
method), 564
msl.equipment.resources.picotech.picoscope.enums
attribute), 319

wait_for_sq()
(*msl.equipment.connection_gpib.Connection*)^{WINDOW}
method), 28
msl.equipment.resources.picotech.picoscope.enums.PS2000
attribute), 248

wait_to_configure()
(*msl.equipment.resources.dataray.datarayocx_32.DatarayOCX32*)^{WINDOW}
method), 133
msl.equipment.resources.picotech.picoscope.enums.PS3000
attribute), 266

wait_to_configure()
(*msl.equipment.resources.dataray.datarayo*)^{WINDOW}
method), 134
msl.equipment.resources.picotech.picoscope.enums.PS3000
attribute), 256

wait_until_ready()
(*msl.equipment.resources.picotech.picoscope.picoscatterHiliteScope*)^{WINDOW}
method), 329
msl.equipment.resources.picotech.picoscope.enums.PS4000
attribute), 277

WAITLOCK (*msl.equipment.vxi11.OperationFlag*)
attribute), 602
msl.equipment.resources.picotech.picoscope.enums.PS5000
attribute), 277

wavelength (*msl.equipment.resources.bentham.benhw64.Benhw*)
attribute), 310
property), 131
msl.equipment.resources.picotech.picoscope.enums.PS5000
attribute), 277

wDigOPs (*msl.equipment.resources.thorlabs.kinesis.structs.QD_DIGITALIO*)
attribute), 591
msl.equipment.resources.picotech.picoscope.enums.PS6000
attribute), 320

WheelAcceleration
(*msl.equipment.resources.thorlabs.kinesis.structs.KMCKEYREADMI*)^{wReserved}
attribute), 583
msl.equipment.resources.thorlabs.kinesis.structs.QD_
attribute), 106

WheelAdjustRate
(*msl.equipment.resources.thorlabs.kinesis.structs.KNATTMIDI*)^{wReserved}
attribute), 586
msl.equipment.resources.thorlabs.kinesis.structs.QD_
attribute), 591

WheelDirectionSense
(*msl.equipment.resources.thorlabs.kinesis.structs.KRTEKMQEMPI*)^{wReserved}
attribute), 583
msl.equipment.resources.picotech.picoscope.enums.PS4000
attribute), 287

WheelMaxVelocity
(*msl.equipment.resources.thorlabs.kinesis.structs.KMOTTHIMI*)^{write()}
attribute), 583
msl.equipment.connection_message_based.ConnectionPrologix
attribute), 583

WheelMode (*msl.equipment.resources.thorlabs.kinesis.structs.KMOTLMPI*)
attribute), 583
msl.equipment.hislip.HiSLIPClient
attribute), 583

WHITE_NOISE (*msl.equipment.resources.picotech.picoscope.enums*)^{write()}
attribute), 285
msl.equipment.vxi11.RPCClient
attribute), 285

WHITE_NOISE (*msl.equipment.resources.picotech.picoscope.enums*)^{write()}
attribute), 275
msl.equipment.connection_gpib.ConnectionGPII
attribute), 275

WHITE_NOISE (*msl.equipment.resources.picotech.picoscope.enums*)^{write_async()}
attribute), 285

<code>write_authentication_exchange()</code>	<code>(msl.equipment.hislip.SyncClient method), 76</code>	ZFS225 (<code>msl.equipment.resources.thorlabs.kinesis.enums.KST_Stages attribute</code>), 472
<code>write_termination</code>	<code>(msl.equipment.connection_message_based.ZMQInMessageBosstants.Interface attribute), 50</code>	ZFS225 (<code>msl.equipment.resources.thorlabs.kinesis.enums.TST_Stages attribute</code>), 476
<code>write_termination</code>	<code>(msl.equipment.connection_prologix.ConnectionPrologix attribute), 472</code>	ZST13 (<code>msl.equipment.resources.thorlabs.kinesis.enums.KST_Stages attribute</code>), 476
<code>write_termination</code>	<code>(msl.equipment.connection_prologix.ConnectionPrologix attribute), 472</code>	ZST13 (<code>msl.equipment.resources.thorlabs.kinesis.enums.TST_Stages attribute</code>), 476
X		ZST206 (<code>msl.equipment.resources.thorlabs.kinesis.enums.KST_Stages attribute</code>), 472
<code>x</code> (<code>msl.equipment.resources.thorlabs.kinesis.structs.QD_Position attribute</code>), 590		ZST206 (<code>msl.equipment.resources.thorlabs.kinesis.enums.TST_Stages attribute</code>), 476
<code>xFeedbackSignedGain</code>	<code>(msl.equipment.resources.thorlabs.kinesis.structs.QD_Position attribute), 590</code>	ZST206 (<code>msl.equipment.resources.thorlabs.kinesis.enums.KST_Stages attribute</code>), 472
<code>XML</code> (<code>msl.equipment.resources.picotech.picoscope.enums.ZTSPS4000AInterface resources.thorlabs.kinesis.enums.TST_Stages attribute</code>), 287		ZST206 (<code>msl.equipment.resources.thorlabs.kinesis.enums.TST_Stages attribute</code>), 476
<code>xml_comment()</code> (in module <code>msl.equipment.utils</code>), 599		ZST225 (<code>msl.equipment.resources.thorlabs.kinesis.enums.KST_Stages attribute</code>), 472
<code>xml_element()</code> (in module <code>msl.equipment.utils</code>), 599		ZST225 (<code>msl.equipment.resources.thorlabs.kinesis.enums.TST_Stages attribute</code>), 476
<code>XMLType</code> (in module <code>msl.equipment.config</code>), 17		ZST25 (<code>msl.equipment.resources.thorlabs.kinesis.enums.KST_Stages attribute</code>), 472
Y		ZST25 (<code>msl.equipment.resources.thorlabs.kinesis.enums.TST_Stages attribute</code>), 476
<code>y</code> (<code>msl.equipment.resources.thorlabs.kinesis.structs.QD_Position attribute</code>), 590		ZST6 (<code>msl.equipment.resources.thorlabs.kinesis.enums.KST_Stages attribute</code>), 472
<code>Year</code> (<code>msl.equipment.resources.nkt.nktpdll.DateTimeType</code> attribute), 163		ZST6 (<code>msl.equipment.resources.thorlabs.kinesis.enums.TST_Stages attribute</code>), 476
<code>yFeedbackSignedGain</code>	<code>(msl.equipment.resources.thorlabs.kinesis.structs.QD_PositionDemandParameters attribute), 590</code>	ZST6 (<code>msl.equipment.resources.thorlabs.kinesis.enums.TST_Stages attribute</code>), 476
Z		
<code>zero_calibration()</code>	<code>(msl.equipment.resources.bentham.benhw32.Bentham32 method), 129</code>	
<code>zero_calibration()</code>	<code>(msl.equipment.resources.bentham.benhw64.Bentham method), 131</code>	
<code>zero_voltage_monitor()</code>	<code>(msl.equipment.resources.optronic_laboratories.ol_current_source.OLCurrentSource method), 227</code>	
<code>ZFS206</code> (<code>msl.equipment.resources.thorlabs.kinesis.enums.KST_Stages attribute</code>), 472		
<code>ZFS206</code> (<code>msl.equipment.resources.thorlabs.kinesis.enums.TST_Stages attribute</code>), 476		
<code>ZFS213</code> (<code>msl.equipment.resources.thorlabs.kinesis.enums.KST_Stages attribute</code>), 472		
<code>ZFS213</code> (<code>msl.equipment.resources.thorlabs.kinesis.enums.TST_Stages attribute</code>), 476		