pr-single-photons Documentation

Release 0.1.0.dev0

Measurement Standards Laboratory of New Zealand

Jan 15, 2024

CONTENTS

Software for single-photon generation and detection.

CHAPTER

ONE

CONTENTS

1.1 photons

1.1.1 photons package

Console script entry points.

photons.**version_info = (0, 1, 0, 'dev0')**

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

Type
namedtuple

photons.**cli_parser**(**args*)

Parse the command line arguments.

Return type
Namespace

photons.**main**(**args*)

Main console script entry point.

Run `photons --help` for more details.

Parameters
args (str) – Command-line arguments.

Return type
None

Examples

- Start the main application using the default configuration path
  `photons`
- Start the main application using the specified configuration file
  `photons my_config.xml`
- Start an equipment Service (using the default configuration path)
  `photons --alias shutter`
- Start an equipment Service using the specified configuration file
  `photons my_config.xml --alias shutter`

- Start a registered Service and specify kwargs
  `photons --name MyService --kwargs "{\"host\":\"localhost\", \"port\":1876}"`
- Start a JupyterLab web server
  `photons --jupyter`
- Find equipment
  `photons --find`
- Find equipment (with a 5-second timeout for network devices)
  `photons --find 5`

photons.**start_app**(*config*, *no_user*)

Start the main application instance.

Parameters

- **config** (Optional[str]) – The path to a configuration file.
- **no_user** (bool) – Whether to call *input('Press <Enter> to exit... ')* if there was an error.

Return type
int

Returns
The exit code (0 for success, 1 for error).

photons.**start_service**(*, *alias=None*, *config=None*, *name=None*, *kwargs=None*, *no_user=False*, ***ignored*)

Start a Service.

Parameters

- **alias** (str) – The alias of an EquipmentRecord to start a generic equipment Service.
- **config** (str) – The path to a configuration file. Not required if *name* is specified.
- **name** (str) – The name of a registered Service to start.
- **kwargs** (str) – The keyword arguments from the command line.
- **no_user** (bool) – Whether to call *input('Press <Enter> to exit... ')* if there was an error.
- **ignored** – All other keyword arguments are ignored.

Return type
int

Returns
The exit code (0 for success, 1 for error).

photons.**start_jupyter**(*config*, *no_user*)

Start a Jupyter web server.

Parameters

- **config** (Optional[str]) – The path to a configuration file.
- **no_user** (bool) – Whether to call *input('Press <Enter> to exit...')* if there was an error.

Return type
int

Returns
The exit code (0 for success, 1 for error).

Subpackages

photons.analysis package

Submodules

photons.analysis.spatial_scan module

photons.equipment package

Custom classes for communicating with equipment.

Subpackages

photons.equipment.widgets package

Custom Qt widgets for equipment.

Submodules

photons.equipment.widgets.daq_counter module

Widget for a NIDAQ to counts pulse edges.

class photons.equipment.widgets.daq_counter.**DAQCounterWidget**(*connection*, *, *parent=None*)

Bases: *BaseEquipmentWidget*

Widget for a NIDAQ to counts pulse edges.

Parameters

- **connection** (*NIDAQ*) – The connection to the NIDAQ.
- **parent** (QWidget) – The parent widget.

on_counts_changed(*samples*)

Update the text in the LineEdit.

Return type
None

on_live_checkbox_changed(*checked*)

Start or stop the QTimer.

Return type
None

on_live_spinbox_changed(*msec*)

Change the timeout interval of the QTimer.

Return type
None

on_timer_timeout()

Start the worker thread.

Return type
None

notification_handler(***kwargs*)

Handle a notification emitted by the NIDAQ Service.

Return type
None

stop_timer_and_thread()

Stop the QTimer and the QThread.

Return type
None

closeEvent(*event*)

Override QtWidgets.QWidget.closeEvent() to stop the QTimer and QThread.

Return type
None

class photons.equipment.widgets.daq_counter.**CountEdgesWorker**(*connection*, *duration*, *edge*, *pfi*, *nsamples*)

Bases: Worker

Count edges in a worker thread.

process()

The expensive or blocking operation to process. :rtype: None

Attention: You must override this method.

class photons.equipment.widgets.daq_counter.**CountEdgesThread**(*parent*)

Bases: Thread

Moves a Worker to a new QThread.

Parameters

worker
A `Worker` subclass that has *not* been instantiated.

Example

See `Sleep` for an example of a `Thread`.

error_handler(*exception*, *traceback*)
If an exception is raised by the `Worker` then the default behaviour is to show the error message in a `critical()` dialog window.

You can override this method to implement your own error handler.

Return type
`None`

Parameters

exception
[`BaseException`] The exception instance

traceback
[`traceback`] A traceback object.

photons.equipment.widgets.dmm module

Widget for a digital multimeter.

class photons.equipment.widgets.dmm.**FetchWorker**(*connection*, *trigger_mode*)
Bases: `Worker`

Fetch samples from the DMM in a worker thread.

process()
Fetch the samples from the DMM.

class photons.equipment.widgets.dmm.**DMMWidget**(*connection*, *, *parent=None*)
Bases: *BaseEquipmentWidget*

Widget for a digital multimeter.

Parameters

- **connection** (*DMM*) – The connection to the digital multimeter.
- **parent** (`QWidget`) – The parent widget.

get_settings()
Get the configuration settings of the DMM.

Return type
Settings

on_digits_spinbox_changed(_)

Change the number of digits in the uncertainty to retain.

Return type
None

on_edit_configuration()

Show the QDialog to edit the configuration settings of the DMM.

Return type
None

on_fetched(*samples*)

Samples were fetched.

Return type
None

on_live_checkbox_changed(*checked*)

Start or stop the QTimer.

Return type
None

on_show_plot()

Show the RealTimePlot widget.

Return type
None

on_settings_changed(*settings*)

Slot for the connection.config_changed signal.

Return type
None

on_timer_timeout()

Start the worker thread.

Return type
None

notification_handler(**args*, ***kwargs*)

Handle a notification emitted by the DMM Service.

Return type
None

restart_timer_and_thread()

Restart the Thread and the QTimer.

Return type
None

stop_timer_and_thread()

Stop the QTimer and the QThread.

Return type
None

update_tooltip()

Update the tooltip of the 'value' LineEdit.

Return type
None

closeEvent(*event*)

Override QtWidgets.QWidget.closeEvent() to stop the QTimer and QThread.

Return type
None

class photons.equipment.widgets.dmm.**ConfigureDialog**(*parent*)

Bases: QDialog

Edit the configuration of the DMM.

auto_delay_changed(*checked*)

The Trigger auto-delay CheckBox was clicked.

Return type
None

on_apply_clicked()

The Apply button was clicked.

Return type
None

on_reset()

Send the *RST command to the digital multimeter.

Return type
None

save_settings()

Save the settings to the digital multimeter.

Return type
None

closeEvent(*event*)

Overrides QtWidgets.QWidget.closeEvent() and maybe prompt to save.

Return type
None

photons.equipment.widgets.hrs_monochromator module

Widget for a HRS500M Monochromator from Princeton Instruments.

class photons.equipment.widgets.hrs_monochromator.**HRSMonochromatorWorker**(*method*, **args*)

Bases: Worker

Execute the call to the Princeton Instruments DLL in a worker thread.

process()

The expensive or blocking operation to process. :rtype: None

> **Attention:** You must override this method.

class photons.equipment.widgets.hrs_monochromator.**HRSMonochromatorWidget**(*connection*, *, *parent=None*)

Bases: *BaseEquipmentWidget*

Widget for a HRS500M Monochromator from Princeton Instruments.

Parameters

- **connection** (*HRSMonochromator*) – The connection to the monochromator.
- **parent** (QWidget) – The parent widget.

on_grating_index_changed(*index*)

Slot for the Grating QComboBox.currentIndexChanged signal.

Parameters
index (int) – The grating index [0, 1, 2].

Return type
None

on_grating_position_changed(*position*)

Slot for the HRSMonochromator.grating_position_changed signal.

Updates the Grating QComboBox without emitting the currentIndexChanged signal.

Parameters
position (int) – The grating position [1, 2, 3].

Return type
None

on_filter_index_changed(*index*)

Slot for the Filter QComboBox.currentIndexChanged signal.

Parameters
index (int) – The filter index [0, 1, 2, 3, 4, 5].

Return type
None

on_filter_position_changed(*position*)

Slot for the HRSMonochromator.filter_position_changed signal.

Updates the Filter QComboBox without emitting the currentIndexChanged signal.

Parameters
position (int) – The filter position [1, 2, 3, 4, 5, 6].

Return type
None

on_wavelength_editing_finished()

Slot for the Wavelength QDoubleSpinBox.editingFinished signal.

Return type
None

on_wavelength_changed(*requested*, *encoder*)

Slot for the HRSMonochromator.wavelength_changed signal.

Parameters

- **requested** (float) – The requested wavelength.
- **encoder** (float) – The wavelength that the encoder indicates that it is at.

Return type
None

on_front_entrance_slit_editing_finished()

Slot for the front entrance slit QSpinBox.editingFinished signal.

Return type
None

on_front_entrance_slit_changed(*width*)

Slot for the HRSMonochromator.front_entrance_slit_changed signal.

Parameters
width (int) – The front entrance slit width, in um.

Return type
None

on_front_exit_slit_editing_finished()

Slot for the front exit slit QSpinBox.editingFinished signal.

Return type
None

on_front_exit_slit_changed(*width*)

Slot for the HRSMonochromator.front_exit_slit_changed signal.

Parameters
width (int) – The front exit slit width, in um.

Return type
None

on_home_front_entrance_slit()

Home the front entrance slit.

Return type
None

on_home_front_exit_slit()

Home the front exit slit.

Return type
None

on_home_filter_wheel()

Home the filter wheel.

Return type
None

on_thread_finished()

Called when the worker thread is finished.

Return type
None

notification_handler(**args*, ***kwargs*)

Handle notifications emitted by the HRSMonochromator Service.

Return type
None

prepare_thread()

Prepare to start a new worker thread.

Return type
bool

Returns
Whether a new thread can be started.

photons.equipment.widgets.keithley_6430 module

Widget for a Keithley 6430 sub-femtoAmp SourceMeter.

class photons.equipment.widgets.keithley_6430.**Keithley6430Widget**(*connection*, ***, *parent=None*)

Bases: *BaseEquipmentWidget*

Widget for a Keithley 6430 sub-femtoAmp SourceMeter.

Parameters

- **connection** (*Keithley6430*) – The connection to the SourceMeter.
- **parent** (QWidget) – The parent widget.

connection: *Keithley6430*

on_function_text_changed(*text*)

Slot for the Function QComboBox.currentTextChanged signal.

Return type
None

on_source_settings_changed(*settings*)

Slot for the Keithley6430.source_settings_changed signal.

Return type
None

on_configure_source()

Slot to configure the Source subsystem.

Return type
None

photons.equipment.widgets.laser_superk module

Widget for a SuperK Fianium laser from NKT Photonics.

class photons.equipment.widgets.laser_superk.**Watchdog**(*connection*, *qu*)

Bases: Worker

Handle NKTDLL callbacks.

emission_changed: SignalInstance

level_changed: SignalInstance

mode_changed: SignalInstance

process()

The expensive or blocking operation to process. :rtype: None

> **Attention:** You must override this method.

class photons.equipment.widgets.laser_superk.**SuperKWidget**(*connection*, ***, *parent=None*)

Bases: *BaseEquipmentWidget*

Widget for a SuperK Fianium laser from NKT Photonics.

Parameters

- **connection** (*SuperK*) – The connection to the laser.
- **parent** (QWidget) – The parent widget.

connection: *SuperK*

closeEvent(*event*)

Stop the Watchdog thread and close the Widget.

Return type
None

notification_handler(**args*, ***kwargs*)

Handle notifications emitted by the SuperK Service.

Return type
None

on_mode_changed(*text*)

Change the operating mode.

Return type
None

on_level_editing_finished()
Set the level when the DoubleSpinBox loses focus.

Return type
None

on_level_changed(*level*)
Update the value of the DoubleSpinBox without emitting the signal.

Return type
None

on_device_status_changed(*status*)
Called by the DeviceStatusCallback from the DLL.

Return type
None

on_emission_changed(*state*)
Update the ToggleSwitch without emitting the signal.

Return type
None

on_user_text_changed(*text*)
Set the user text.

Return type
None

update_mode(*mode*)
Update the ComboBox without emitting the signal.

Return type
None

photons.equipment.widgets.shot702_controller module

Widget for an OptoSigma SHOT-702 controller.

class photons.equipment.widgets.shot702_controller.**OptoSigmaSHOT702Widget**(*connection*, *, *parent=None*)
Bases: *BaseEquipmentWidget*

Widget for an OptoSigma SHOT-702 controller.

Parameters

- **connection** (*OptoSigmaSHOT702*) – The connection to the SHOT-702 controller.
- **parent** (QWidget) – The parent widget.

connection: *OptoSigmaSHOT702*

has_move_started()

Return whether the rotation stage is moving.

Return type
bool

notification_handler(*position*, *angle*, *is_moving*)

Handle notifications emitted by the OptoSigmaSHOT702 Service.

Return type
None

on_angle_editing_finished()

Set the angle of the stage.

Return type
None

on_edit_settings()

Show the Settings Dialog.

Return type
None

on_home()

Home the stage.

Return type
None

on_stop()

Stop the stage from moving.

Return type
None

on_timer_timeout()

Slot for the QTimer timeout signal.

Return type
None

update_angle_spinbox(*position*, *angle*)

Update the value and tooltip Angle spinbox.

Return type
None

class photons.equipment.widgets.shot702_controller.**SettingsDialog**(*parent*)

Bases: QDialog

Edit the rotation rate and the acceleration settings.

Parameters
parent (*OptoSigmaSHOT702Widget*) – The parent widget.

on_apply_clicked()

The Apply button was clicked.

Return type
None

save_settings()

Save the settings to the controller.

Return type
None

closeEvent(*event*)

Overrides QtWidgets.QWidget.closeEvent().

Return type
None

photons.equipment.widgets.shutter module

Widget for a shutter.

class photons.equipment.widgets.shutter.**ShutterWidget**(*connection*, *, *parent=None*)

Bases: *BaseEquipmentWidget*

Widget for a shutter.

Parameters

- **connection** (*Shutter*) – The connection to the shutter controller.
- **parent** (QWidget) – The parent widget.

connection: *Shutter*

notification_handler(*state*)

Handle notifications emitted by the Shutter Service.

Return type
None

on_toggled(*state*)

Toggle the state of the shutter.

Return type
None

on_state_changed(*state*)

Update the ToggleSwitch without emitting the signal.

Return type
None

update_tooltip()

Update the tooltip of the ToggleSwitch.

Return type
None

photons.equipment.widgets.sia_cmi module

Widget for a Switched Integrator Amplifier from CMI.

class photons.equipment.widgets.sia_cmi.**SIA3CMIWidget**(*connection*, *, *parent=None*)

Bases: *BaseEquipmentWidget*

Widget for a Switched Integrator Amplifier from CMI.

Parameters

- **connection** (*SIA3CMI*) – The connection to the amplifier.
- **parent** (QWidget) – The parent widget.

connection: *SIA3CMI*

notification_handler(*integration_time*)

Handle notifications from the SIA3CMI Service.

Return type
None

on_index_changed(*index*)

Set the integration time.

Return type
None

on_integration_time_changed(*value*)

Update the combobox without emitting the signal.

Return type
None

photons.equipment.widgets.thorlabs_flipper module

Widget for a Thorlabs filter flipper (MFF101 or MFF102).

class photons.equipment.widgets.thorlabs_flipper.**ThorlabsFlipperWidget**(*connection*, *, *parent=None*)

Bases: *BaseEquipmentWidget*

Widget for a Thorlabs filter flipper (MFF101 or MFF102).

Parameters

- **connection** (*ThorlabsFlipper*) – The connection to the flipper.
- **parent** (QWidget) – The parent widget.

connection: *ThorlabsFlipper*

notification_handler(*info*)

Handle the notifications from a MotionControlCallback.

Return type
None

on_index_changed(*index*)

Slot for the QComboBox.currentIndexChanged signal.

Return type
None

on_callback(*info*)

Slot for the MotionControlCallback signal.

Return type
None

photons.equipment.widgets.thorlabs_stage module

Widget for a Thorlabs translational/rotational stage.

class photons.equipment.widgets.thorlabs_stage.**ThorlabsStageWidget**(*connection*, *, *parent=None*)

Bases: *BaseEquipmentWidget*

Widget for a Thorlabs translational/rotational stage.

Parameters

- **connection** (*ThorlabsStage*) – The connection to the stage controller.
- **parent** (QWidget) – The parent widget.

connection: *ThorlabsStage*

notification_handler(*info*)

Handle the notifications from a MotionControlCallback.

Return type
None

on_callback(*info*)

Slot for the MotionControlCallback.

Return type
None

on_edit_settings()

Slot for the self.settings_button click.

Return type
None

on_editing_finished()

Slot for the DoubleSpinBox.editingFinished signal.

Return type
None

on_home()

Home the stage.

Return type
None

on_stop()

Stop the stage from moving.

Return type
None

class photons.equipment.widgets.thorlabs_stage.**Settings**(*parent*)

Bases: QDialog

Edit the jog step size.

closeEvent(*event*)

Overrides QtWidgets.QWidget.closeEvent() and maybe prompt to save.

Return type
None

on_apply()

Apply the settings.

Return type
None

save_settings()

Save the step size to the DoubleSpinBox.

Return type
None

Submodules

photons.equipment.arroyo_6305 module

ComboSource 6305 laser-diode controller from Arroyo Instruments.

class photons.equipment.arroyo_6305.**ComboSource**(*record*, ***kwargs*)

Bases: *BaseEquipment*

ComboSource 6305 laser-diode controller from Arroyo Instruments.

Parameters

- **record** (EquipmentRecord) – The equipment record.
- ****kwargs** – Keyword arguments. Can be specified as attributes of an XML element in a configuration file (with the tag of the element equal to the alias of *record*).

connection: ConnectionMessageBased

CURRENT_LIMIT = 1

VOLTAGE_LIMIT = 2

SENSOR_LIMIT = 4

PHOTODIODE_CURRENT_LIMIT = 4

PHOTODIODE_POWER_LIMIT = 8

TEMPERATURE_HIGH_LIMIT = 8

TEMPERATURE_LOW_LIMIT = 16

INTERLOCK_DISABLED = 16

SENSOR_SHORTED = 32

SENSOR_OPEN = 64

OPEN_CIRCUIT = 128

LASER_SHORT_CIRCUIT = 256

OUT_OF_TOLERANCE = 512

OUTPUT_ON = 1024

THERMAL_RUN_AWAY = 4096

R_LIMIT = 8192

T_LIMIT = 16384

clear()

Clears the standard event status register, all event registers, and the error queue.

Return type
None

condition_register_laser()

Returns the condition-register value of the laser.

Bit	Value	Description
0	1	Current limit
1	2	Voltage limit
2	4	Photodiode current limit
3	8	Photodiode power limit
4	16	Interlock disabled
5	32	Unused
6	64	Unused
7	128	Laser open circuit
8	256	Laser short circuit
9	512	Out of tolerance
10	1024	Output on
11	2048	Unused
12	4096	Unused
13	8192	R limit
14	16384	T limit
15	32768	Unused

Return type
int

`condition_register_tec()`

Returns the condition-register value of the TEC.

Bit	Value	Description
0	1	Current limit
1	2	Voltage limit
2	4	Sensor limit
3	8	Temperature high limit
4	16	Temperature low limit
5	32	Sensor shorted
6	64	Sensor open
7	128	TEC open circuit
8	256	Unused
9	512	Out of tolerance
10	1024	Output on
11	2048	Unused
12	4096	Thermal run-away
13	8192	Unused
14	16384	Unused
15	32768	Unused

Return type
int

`disable_error_checking()`

Disable error checking.

Return type
None

`enable_error_checking()`

Enable error checking.

Return type
None

`get_laser_current()`

Returns the applied current (in milliamps) to the laser diode.

Return type
float

`get_laser_current_setpoint()`

Returns the setpoint current (in milliamps) of the laser diode.

Return type
float

`get_laser_temperature()`

Returns the temperature (in Celsius) of the laser diode.

Return type
float

`get_laser_tolerance()`

Returns the *tolerance* (in milliamps) and the time window (in seconds) that is required to achieve *tolerance*.

Return type
tuple[float, float]

`get_tec_temperature_setpoint()`

Returns the setpoint temperature of the TEC.

Return type
float

`get_tec_tolerance()`

Returns the *tolerance* value (in Celsius) and the time window (in seconds) that is required to achieve *tolerance*.

Return type
tuple[float, float]

`is_laser_enabled()`

Checks if the laser is lasing.

Return type
bool

`is_tec_enabled()`

Checks if the TEC is enabled or disabled.

Return type
bool

`laser_off`()

Turn the laser output off.

Return type
None

`laser_on`(*, *wait=True*, *timeout=120*)

Turn the laser output on.

Parameters

- **wait** (bool) – Whether to wait for the laser diode current and temperature to stabilize before returning to the calling program.
- **timeout** (float) – The maximum number of seconds to wait before raising an exception.

Return type
None

`photodiode_current`()

Returns the current (in uA) of the monitoring photodiode.

Return type
float

`set_laser_current`(*milliamps*, *, *wait=True*, *timeout=120*)

Set the laser diode current.

Parameters

- **milliamps** (float) – The laser diode setpoint current, in mA.
- **wait** (bool) – Whether to wait for the laser diode current and temperature to stabilize before returning to the calling program.
- **timeout** (float) – The maximum number of seconds to wait before raising an exception.

Return type
None

`set_laser_tolerance`(*tolerance*, *duration*)

Set the laser tolerance criteria.

Allows control over when the output of the laser driver is considered in tolerance (or stable), in order to satisfy the tolerance condition of the operation complete definition.

Parameters

- **tolerance** (float) – Tolerance value (in milliamps) for the measured laser diode current to be within the setpoint. Must be between 0 and 100.
- **duration** (float) – The measured current must be within the setpoint value plus or minus the *tolerance* value for *duration* seconds. Must be between 0.1 and 50 seconds.

Return type
None

set_tec_temperature(*celsius*, ***, *wait=True*, *timeout=120*)

Set the laser diode current.

Parameters

- **celsius** (float) – The TEC temperature, in Celsius.
- **wait** (bool) – Whether to wait for the laser diode current and temperature to stabilize before returning to the calling program.
- **timeout** (float) – The maximum number of seconds to wait before raising an exception.

Return type
None

set_tec_tolerance(*tolerance*, *duration*)

Set the TEC tolerance criteria.

Allows control over when the output of the temperature controller is considered in tolerance (or stable), in order to satisfy the tolerance condition of the operation complete definition.

Parameters

- **tolerance** (float) – Tolerance value, in Celsius, for the measured laser temperature to be within the setpoint. Must be between 0.01 and 10.
- **duration** (float) – The measured temperature must be within the setpoint value plus or minus the *tolerance* value for *duration* seconds. Must be between 0.1 and 50 seconds.

Return type
None

tec_off()

Disable TEC control.

Return type
None

tec_on(***, *wait=True*, *timeout=120*)

Enable TEC control.

Parameters

- **wait** (bool) – Whether to wait for the laser diode current and temperature to stabilize before returning to the calling program.
- **timeout** (float) – The maximum number of seconds to wait before raising an exception.

Return type
None

wait(*timeout=120*)

Wait for the laser diode current and temperature to be stable.

The condition-register value must be okay and the laser diode current and temperature values must be within tolerance of the setpoint value.

Parameters
timeout (float) – The maximum number of seconds to wait before raising an exception.

Return type
None

photons.equipment.base module

Base classes and decorators for equipment.

class photons.equipment.base.**BaseEquipment**(*record*, ***kwargs*)

Bases: QObject, Service

Base class for all equipment connections.

Parameters

- **record** (EquipmentRecord) – The equipment record.
- ****kwargs** – Keyword arguments that a subclass requires. Can be specified as attributes of an XML element in a configuration file (with the tag of the element equal to the alias of *record*).

__getattr__(*item*)

Pass all attributes that do not exist to the connection object.

Return type
Any

static convert_to_enum(*obj*, *enum*, *prefix=None*, *to_upper=False*)

See convert_to_enum() for more details.

Return type
TypeVar(E, bound= Enum)

disconnect_equipment()

Disconnect from the equipment and log an INFO message.

Return type
None

property logger

Reference to the package logger.

maybe_emit_notification(**args*, ***kwargs*)

Emit a notification to all Clients that are linked with this Service.

Return type
None

property notifications_allowed: bool

Returns whether notifications are allowed to be sent to Clients.

raise_exception(*message*)

Log the message then raise an exception.

Return type
None

record_to_json()
Returns the EquipmentRecord as a JSON-serializable object.

Return type
dict

property timeout: float | None
The timeout, in seconds, for read and write operations.

This property is valid only if the underlying connection is ConnectionMessageBased.

class photons.equipment.base.**BaseEquipmentWidget**(*connection*, *, *parent=None*, ***kwargs*)

Bases: QWidget

Base class for all Qt widgets that connect to equipment.

Parameters

- **connection** (Link | *BaseEquipment* | Connection) – The connection to the equipment.
- **parent** (QWidget) – The parent widget.
- ****kwargs** – All keyword arguments are passed to super().

closing: SignalInstance

closeEvent(*event*)
Overrides QtWidgets.QWidget.closeEvent().

Return type
None

property logger
Reference to the package logger.

notification_handler(**args*, ***kwargs*)
Override in subclass to handle notifications emitted by a Service.

Return type
None

class photons.equipment.base.**EquipmentMatcher**(*cls*, *manufacturer*, *model*, *flags*)

Bases: object

Performs match operations on an EquipmentRecord.

Parameters

- **cls** (type[*BaseEquipment* | *BaseEquipmentWidget*]) – The class to associate with the matcher. The class must not be instantiated.
- **manufacturer** (Optional[str]) – The name of the manufacturer. Can be a regex pattern.
- **model** (Optional[str]) – The model number of the equipment. Can be a regex pattern.

- **flags** (int) – The flags to use to compile the regex patterns.

matches(*record*)

Checks if *record* is a match.

Parameters
record (EquipmentRecord) – The equipment record to check if the manufacturer and the model number are a match.

Return type
bool

Returns
Whether *record* is a match.

photons.equipment.base.**equipment**(*, *manufacturer=None*, *model=None*, *flags=0*)

A decorator to register equipment (for connections).

Parameters

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

photons.equipment.base.**widget**(*, *manufacturer=None*, *model=None*, *flags=0*)

A decorator to register a widget (for equipment).

Parameters

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

photons.equipment.coherent_fieldmaster module

Coherent FieldMaster GS power meter.

class photons.equipment.coherent_fieldmaster.**FieldMasterGS**(*record*, ***kwargs*)

Bases: *BaseEquipment*

Coherent FieldMaster GS power meter.

Parameters

- **record** (EquipmentRecord) – The equipment record.
- ****kwargs** – Keyword arguments. Can be specified as attributes of an XML element in a configuration file (with the tag of the element equal to the alias of *record*).

connection: ConnectionMessageBased

detector()

Returns the detector type.

Return type
str

`get_attenuation()`

Returns the attenuation factor.

Return type
float

`get_offset()`

Returns the offset.

Return type
float

`get_wavelength()`

Returns the wavelength (in nm).

Return type
float

`power(nsamples=1)`

Returns the power readings (in Watts).

Parameters
nsamples (int) – The number of samples to acquire.

Return type
Samples

`restart()`

Restart the system.

Return type
None

`set_attenuation(attenuation)`

Sets the attenuation factor.

Return type
None

`set_offset(enabled)`

Whether to use the current reading as the offset or to turn the offset off.

Return type
None

`set_wavelength(nm)`

Sets the wavelength (in nm).

Return type
None

photons.equipment.dmm module

Base class for a digital multimeter.

class photons.equipment.dmm.**Auto**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: StrEnum

For settings that can be in an auto mode (e.g., RANGE, ZERO).

OFF = 'OFF'

ON = 'ON'

ONCE = 'ONCE'

class photons.equipment.dmm.**Edge**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: StrEnum

The trigger edge.

RISING = 'RISING'

FALLING = 'FALLING'

BOTH = 'BOTH'

class photons.equipment.dmm.**Function**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: StrEnum

The measurement function.

DCV = 'DCV'

DCI = 'DCI'

class photons.equipment.dmm.**Mode**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: StrEnum

The trigger mode.

IMMEDIATE = 'IMMEDIATE'

BUS = 'BUS'

EXTERNAL = 'EXTERNAL'

class photons.equipment.dmm.**Range**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: StrEnum

Function range.

AUTO = 'AUTO'

MINIMUM = 'MINIMUM'

MAXIMUM = 'MAXIMUM'

DEFAULT = 'DEFAULT'

class photons.equipment.dmm.**Trigger**(***kwargs*)

Bases: object

The trigger settings of a DMM.

to_json()

Return type
dict[str, bool | int | float | str]

class photons.equipment.dmm.**Settings**(***kwargs*)

Bases: object

The configuration settings of a DMM.

to_json()

Return type
dict[str, int | float | str | dict]

class photons.equipment.dmm.**DMM**(*record*, ***kwargs*)

Bases: *BaseEquipment*

Base class for a digital multimeter.

Parameters

- **record** (EquipmentRecord) – The equipment record.
- ****kwargs** – Keyword arguments. Can be specified as attributes of an XML element in a configuration file (with the tag of the element equal to the alias of *record*).

class Auto(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: StrEnum

For settings that can be in an auto mode (e.g., RANGE, ZERO).

OFF = 'OFF'

ON = 'ON'

ONCE = 'ONCE'

class Edge(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: StrEnum

The trigger edge.

RISING = 'RISING'

FALLING = 'FALLING'

BOTH = 'BOTH'

class Function(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: StrEnum

The measurement function.

DCV = 'DCV'

DCI = 'DCI'

class Mode(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: StrEnum

The trigger mode.

IMMEDIATE = 'IMMEDIATE'

BUS = 'BUS'

EXTERNAL = 'EXTERNAL'

class Range(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: StrEnum

Function range.

AUTO = 'AUTO'

MINIMUM = 'MINIMUM'

MAXIMUM = 'MAXIMUM'

DEFAULT = 'DEFAULT'

connection: ConnectionMessageBased

fetched: SignalInstance

settings_changed: SignalInstance

abort()

Abort a measurement in progress.

Return type
None

acquisition_time(*, *settings=None*, *all_triggers=True*, *line_freq=50.0*)

Get the approximate number of seconds it takes to acquire samples.

Parameters

- **settings** (*Settings*) – The configuration settings of the digital multimeter.
- **all_triggers** (bool) – Whether to include the time to acquire all triggers.
- **line_freq** (float) – The line frequency, in Hz.

Return type
float

check_errors()

Query the error queue.

Raises an exception if there is an error.

Return type
None

clear()

Clears the event registers in all register groups and the error queue.

Return type
None

configure(*, *function=Function.DCV*, *range=10*, *nsamples=10*, *nplc=10*, *auto_zero=Auto.ON*, *trigger=Mode.IMMEDIATE*, *edge=Edge.FALLING*, *ntriggers=1*, *delay=None*)

Configure the digital multimeter.

Return type
Settings

Returns
The result of *settings()* after applying the configuration.

fetch(*initiate=False*)

Fetch the samples.

Parameters
initiate (bool) – Whether to call *initiate()* before fetching the data.

Return type
Samples

initiate()

Put the digital multimeter in the wait-for-trigger state (arm the trigger).

Return type
None

reset()

Resets the digital multimeter to the factory default state.

Return type
None

settings()

Returns the configuration settings of the digital multimeter.

Return type
Settings

`trigger()`

Send a software trigger.

Return type
None

`zero()`

Reset the zero value.

When the multimeter is configured with *auto_zero* set to OFF, the multimeter may gradually drift out of specification. To minimize the drift, you may call this method to take a new zero measurement.

Return type
None

photons.equipment.dmm_34401a module

Hewlett Packard 34401A digital multimeter.

`class photons.equipment.dmm_34401a.HP34401A(record, **kwargs)`

Bases: *DMM*

Hewlett Packard 34401A digital multimeter.

Parameters

- **record** (EquipmentRecord) – The equipment record.
- ****kwargs** – Keyword arguments. Can be specified as attributes of an XML element in a configuration file (with the tag of the element equal to the alias of *record*).

`abort()`

Abort a measurement in progress.

Return type
None

`check_errors()`

Query the error queue.

Raises an exception if there is an error.

Return type
None

`configure(*, function=Function.DCV, range=10, nsamples=10, nplc=10, auto_zero=Auto.ON, trigger=Mode.IMMEDIATE, edge=Edge.FALLING, ntriggers=1, delay=None)`

Configure the digital multimeter.

Parameters

- **function** (*Function* | str) – The measurement function.

- **range** (*Range* | str | float) – The range to use for the measurement.
- **nsamples** (int) – The number of samples to acquire after a trigger event.
- **nplc** (float) – The number of power-line cycles.
- **auto_zero** (*Auto* | bool | int | str) – The auto-zero mode.
- **trigger** (*Mode* | str) – The trigger mode.
- **edge** (*Edge* | str) – The edge to trigger on.
- **ntriggers** (int) – The number of triggers that are accepted before returning to the wait-for-trigger state.
- **delay** (float) – The number of seconds to wait after a trigger event before acquiring samples. If None, then the auto-delay feature is enabled where the digital multimeter automatically determines the delay based on the function, range and NPLC.

Return type
Settings

Returns
The result of *settings()* after applying the configuration.

disconnect_equipment()

Set the digital multimeter to be in LOCAL mode and then close the connection.

Return type
None

fetch(*initiate=False*)

Fetch the samples.

Parameters
initiate (bool) – Whether to call *initiate()* before fetching the data.

Return type
Samples

local_mode()

Set the multimeter to be in LOCAL mode for the RS-232 interface.

All keys on the front panel are fully functional.

Return type
None

remote_mode()

Set the multimeter to be in REMOTE mode for the RS-232 interface.

All keys on the front panel, except the LOCAL key, are disabled.

Return type
None

settings()

Returns the configuration settings of the digital multimeter.

Return type
Settings

photons.equipment.dmm_344xxA module

Keysight 344(60|61|65|70)A digital multimeter.

class photons.equipment.dmm_344xxA.**Keysight344XXA**(*record*, ***kwargs*)

Bases: *DMM*

Keysight 344(60|61|65|70)A digital multimeter.

Parameters

- **record** (EquipmentRecord) – The equipment record.
- ****kwargs** – Keyword arguments. Can be specified as attributes of an XML element in a configuration file (with the tag of the element equal to the alias of *record*).

check_errors()

Query the error queue.

Raises an exception if there is an error.

Return type
None

configure(*, *function=Function.DCV*, *range=10*, *nsamples=10*, *nplc=10*, *auto_zero=Auto.ON*, *trigger=Mode.IMMEDIATE*, *edge=Edge.FALLING*, *ntriggers=1*, *delay=None*)

Configure the digital multimeter.

Parameters

- **function** (*Function* | str) – The measurement function.
- **range** (*Range* | str | float) – The range to use for the measurement.
- **nsamples** (int) – The number of samples to acquire after a trigger event.
- **nplc** (float) – The number of power-line cycles.
- **auto_zero** (*Auto* | bool | int | str) – The auto-zero mode.
- **trigger** (*Mode* | str) – The trigger mode.
- **edge** (*Edge* | str) – The edge to trigger on.
- **ntriggers** (int) – The number of triggers that are accepted before returning to the wait-for-trigger state.
- **delay** (float) – The number of seconds to wait after a trigger event before acquiring samples. If None, then the auto-delay feature is enabled where the digital multimeter automatically determines the delay based on the function, range and NPLC.

Return type
Settings

Returns
The result of *settings()* after applying the configuration.

fetch(*initiate=False*)

Fetch the samples.

Parameters
initiate (bool) – Whether to call *initiate()* before fetching the data.

Return type
Samples

settings()

Returns the configuration settings of the digital multimeter.

Return type
Settings

photons.equipment.dmm_3458A module

Keysight 3458A digital multimeter.

class photons.equipment.dmm_3458A.**Keysight3458A**(*record*, ***kwargs*)

Bases: *DMM*

Keysight 3458A digital multimeter.

Parameters

- **record** (EquipmentRecord) – The equipment record.
- ****kwargs** – Keyword arguments. Can be specified as attributes of an XML element in a configuration file (with the tag of the element equal to the alias of *record*).

abort()

Abort a measurement in progress.

Return type
None

check_errors()

Query the error queue.

Raises an exception if there is an error.

Return type
None

clear()

Clears the event registers in all register groups and the error queue.

Return type
None

configure(**, function=Function.DCV, range=10, nsamples=10, nplc=10, auto_zero=Auto.ON, trigger=Mode.IMMEDIATE, edge=Edge.FALLING, ntriggers=1, delay=None*)

Configure the digital multimeter.

Parameters

- **function** (*Function* | str) – The measurement function.
- **range** (*Range* | str | float) – The range to use for the measurement.
- **nsamples** (int) – The number of samples to acquire after a trigger event.
- **nplc** (float) – The number of power-line cycles.
- **auto_zero** (*Auto* | bool | int | str) – The auto-zero mode.
- **trigger** (*Mode* | str) – The trigger mode.
- **edge** (*Edge* | str) – The edge to trigger on.
- **ntriggers** (int) – The number of triggers that are accepted before returning to the wait-for-trigger state.
- **delay** (float) – The number of seconds to wait after a trigger event before acquiring samples. A value of None is equivalent to zero.

Return type
Settings

Returns
The result of *settings()* after applying the configuration.

fetch(*initiate=False*)

Fetch the samples.

Parameters
initiate (bool) – Whether to call *initiate()* before fetching the samples.

Return type
Samples

reset()

Resets the digital multimeter to the factory default state.

Return type
None

settings()

Returns the configuration settings of the digital multimeter.

Return type
Settings

temperature()

Returns the temperature (in Celsius) of the digital multimeter.

Return type
float

zero()

Reset the zero value.

When the multimeter is configured with *auto_zero* set to OFF, the multimeter may gradually drift out of specification. To minimize the drift, you may call this method to take a new zero measurement.

Return type
None

photons.equipment.highfinesse module

Communicate with a Wavemeter or Laser Spectrum Analyser from HighFinesse.

class photons.equipment.highfinesse.**WLMData64**(*record*)

Bases: Connection

Wrapper around the *WLMData32* class.

Parameters
record (EquipmentRecord) – The equipment record.

disconnect()

Disconnect from the device.

Return type
None

class photons.equipment.highfinesse.**Range**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: IntEnum

Wavelength ranges that are supported.

nm245_325 = 0

nm320_420 = 1

nm410_610 = 2

nm600_1190 = 3

class photons.equipment.highfinesse.**RangeModel**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: IntEnum

Range models that are supported.

OLD = 65535

ORDER = 65534

WAVELENGTH = 65533

class photons.equipment.highfinesse.**HighFinesse**(*record*, ***kwargs*)

Bases: *BaseEquipment*

Communicate with a Wavemeter or Laser Spectrum Analyser from HighFinesse.

Parameters

- **record** (EquipmentRecord) – The equipment record.
- ****kwargs** – Keyword arguments. Can be specified as attributes of an XML element in a configuration file (with the tag of the element equal to the alias of *record*).

connection: *WLMData32*

class Range(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: IntEnum

Wavelength ranges that are supported.

nm245_325 = 0

nm320_420 = 1

nm410_610 = 2

nm600_1190 = 3

class RangeModel(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: IntEnum

Range models that are supported.

OLD = 65535

ORDER = 65534

WAVELENGTH = 65533

get_analysis_mode()

Whether analysis mode is enabled or disabled.

Return type
bool

get_auto_exposure_mode()

Whether auto-exposure mode is enabled or disabled.

Return type
bool

get_exposure_time()

Returns the exposure time, in milliseconds.

Return type
int

get_pattern_data(*index=0*, *timeout=10*)

Returns the interferometer pattern data.

If this class is running as a Service, a list is returned.

Parameters

- **index** (int) – The index of the data type to receive.
  - 0 - Fizeau interferometers or diffraction grating
  - 1 - Additional long interferometer or grating analyzing versions (spectrum analysis)
  - 2 - Fizeau interferometers that support double pulses
  - 3 - Additional interferometer for second pulse
- **timeout** (float) – The number of seconds to wait for the pattern data to be available.

Return type
ndarray | list[int]

get_pulse_mode()

Returns whether pulse mode is enabled (False=CW, True=Pulsed).

Return type
bool

get_wavelength_range()

Returns the currently-selected wavelength range or range model.

Return type
Range | *RangeModel* | int

get_wide_mode()

Returns the measurement precision mode (False=fine, True=wide).

Return type
bool

linewidth(*in_air=True*)

Returns the linewidth, in nm.

Parameters
in_air (bool) – Whether to return the linewidth value in air (True) or vacuum (False).

Return type
float

set_analysis_mode(*mode*)

Whether to enable or disable analysis mode.

Parameters
mode (bool) – Enable (True) or disable (False) analysis mode.

Return type
None

set_auto_exposure_mode(*mode*)

Whether to enable or disable auto-exposure mode.

Parameters
mode (bool) – Enable (True) or disable (False) auto-exposure mode.

Return type
None

set_exposure_time(*ms*)

Set the exposure time, in milliseconds.

This method will disable auto-exposure mode.

Parameters
ms (int) – The exposure time, in milliseconds.

Return type
None

set_linewidth_mode(*mode*)

Whether to enable or disable linewidth mode.

Parameters
mode (bool) – Enable (True) or disable (False) linewidth mode.

Return type
None

set_pulse_mode(*mode*)

Set the pulse mode.

Parameters
mode (int | bool) – CW=0|False, Pulsed=1|True

Return type
None

set_wavelength_range(*value*)

Set the wavelength range.

Important: The *set_wavelength_range_model()* must be called before this method is called in order to select the range model.

Parameters
value (*Range* | int) – If the range model is *RangeModel.ORDER*, the wavelength range is set by a *Range* enum value. If the range model is *RangeModel.WAVELENGTH*, the wavelength range is set by a wavelength value, in nm, as an int data type.

Return type
None

set_wavelength_range_model(*model*)

Set the wavelength range model.

Parameters
model (*RangeModel* | int) – The wavelength range model.

Return type
None

`set_wide_mode`(*mode*)

Set the measurement precision mode.

Parameters
mode (bool) – The precision mode (e.g., False=fine, True=wide).

Return type
None

`start_measurement`()

Start measurements.

Return type
None

`stop_measurement`()

Stop all measurements.

Return type
None

`temperature`()

Get the temperature inside the optical unit, in Celsius.

Return type
float

`wait`(*stable*, *timeout=30*)

Wait for a valid wavelength to be measured and for the exposure time to be stable.

Parameters

- **stable** (float) – The number of seconds the device must be stable for.
- **timeout** (float) – The maximum number of seconds to wait.

Return type
None

`wavelength`(*in_air=True*)

Returns the wavelength, in nanometers.

Parameters
in_air (bool) – Whether to return the wavelength in air (True) or vacuum (False).

Return type
float

`static wavelength_ranges`()

Returns the available wavelength ranges for the Laser Spectrum Analyser.

Return type
dict[str, int]

`static wavelength_range_models`()

Returns the available wavelength range models for the Laser Spectrum Analyser.

Return type
dict[str, int]

photons.equipment.highfinesse_sdk module

Wrapper around the 32-bit wlmData.dll library from HighFinesse.

photons.equipment.highfinesse_sdk.**check**(*r*)

Check the result for an error.

Returns the result if there is no error.

Return type
bool | int | float

photons.equipment.highfinesse_sdk.**check_set**(*r*)

Check the result of a "Set*" function for an error.

Returns the result if there is no error.

Return type
int | float

class photons.equipment.highfinesse_sdk.**WLMData32**(*host*, *port*)

Bases: Server32

Wrapper around the 32-bit wlmData.dll library from HighFinesse.

convert_unit(*value*, *frm*, *to*)

Convert a value into a representation of another unit.

Parameters

- **value** (float) – The value to convert. Must be >= 0.
- **frm** (int) – The unit index of that *value* is currently in.
- **to** (int) – The unit index to convert *value* to.

Return type
float

get_analysis_mode()

Whether analysis mode is enabled or disabled.

Return type
bool

get_auto_exposure_mode()

Whether auto-exposure mode is enabled or disabled.

Return type
bool

get_exposure_time(*channel=1*, *index=1*)

Returns the exposure time (in ms).

Parameters

- **channel** (int) – The signal channel for devices with a multichannel switcher. Should be set to 1 for devices that do not have this option.

- **index** (int) – The CCD array index for devices with more than one CCD array. Can be 1 or 2. For devices with only one CCD array set the value to be 1.

Return type
int

get_linewidth(*index*)

Returns the linewidth in the specified unit.

Parameters
index (int) – The unit to return the linewidth in.

Return type
float

get_linewidth_mode()

Whether linewidth mode is enabled or disabled.

Return type
bool

get_pattern_data(*index*, *channel=1*)

Returns the interferometer pattern data.

Parameters

- **index** (int) – The index of the data type to receive.
  - 0 - Fizeau interferometers or diffraction grating
  - 1 - Additional long interferometer or grating analyzing versions (spectrum analysis)
  - 2 - Fizeau interferometers that support double pulses
  - 3 - Additional interferometer for second pulse
- **channel** (int) – Identifies the switcher channel number. Versions without a switcher must use 1.

Return type
list[int]

Returns
The interferometer pattern data.

get_pulse_mode()

Returns the pulse mode.

Return type
int

get_range()

Returns the wavelength range that is selected.

Return type
int

`get_wide_mode()`

Returns the measurement precision mode.

Return type
`int`

`get_wlm_count()`

Returns the number of wavemeter and spectrum-analyser applications that are running.

Return type
`int`

`get_wlm_version()`

Returns version information about the device.

Return type
`dict[str, int]`

`instantiate(rfc, mode, p1, p2)`

Checks whether the Wavelength Meter or Laser Spectrum Analyser server application is running, changes the return mode of the measurement values, installs/removes an extended exporting mechanism, changes the appearance of the server application window or starts/terminates the server application.

See the manual for more details about the input parameters.

Return type
`int`

Returns
If the function succeeds and at least one Wavelength Meter or Laser Spectrum Analyser is active or terminated due to this instantiating operation the function returns a value greater than 0, else 0.

`operation(mode)`

Set the operation mode.

Parameters
mode (`int`) – Controls how a measurement or file accessing activity will be started or stopped. See manual for more details.

Return type
`int`

`set_analysis_mode(mode)`

Set the analysis mode.

Parameters
mode (`bool`) – Whether to enable (True) or disable (False) analysis mode.

Return type
`None`

`set_auto_exposure_mode(mode)`

Set the auto-exposure mode.

Parameters
mode (`bool`) – Whether to enable (True) or disable (False) auto-exposure mode.

Return type
None

`set_exposure_time`(*ms*, *channel=1*, *index=1*)

Set the exposure time.

Parameters

- **ms** (int) – The exposure time, in ms.
- **channel** (int) – The signal channel for devices with a multichannel switcher. Should be set to 1 for devices that do not have this option.
- **index** (int) – The CCD array index for devices with more than one CCD array. Can be 1 or 2. For devices with only one CCD array set the value to be 1.

`set_linewidth_mode`(*mode*)

Set the linewidth mode.

Parameters
mode (bool) – Whether to enable (True) or disable (False) linewidth mode.

Return type
None

`set_pulse_mode`(*mode*)

Set the pulse mode.

Parameters
mode (int) – The pulse mode (e.g., 0=CW, 1=Pulsed).

Return type
None

`set_range`(*value*)

Set the wavelength range.

Parameters
value (int) – The enum value of the wavelength range.

Return type
None

`set_wide_mode`(*mode*)

Set the measurement precision mode.

Parameters
mode (int) – The precision mode (e.g., 0=fine, 1=wide, 2=grating analysis).

Return type
None

`temperature`()

Returns the temperature of the device, in Celsius.

Return type
float

wait(*duration*, *timeout=30*)

Wait for a valid wavelength to be measured and for the exposure time to be stable.

Parameters

- **duration** (float) – The number of seconds the device must be stable for.
- **timeout** (float) – The maximum number of seconds to wait.

Return type
None

wavelength(*number=0*)

Returns the wavelength (in nm)

Parameters
number (int) – The signal number (1 to 8) if the wavemeter has a multichannel switcher or contains the double-pulse option. For wavemeters without these options set to 0.

Return type
float

photons.equipment.hrs_monochromator module

HRS500M Monochromator from Princeton Instruments.

class photons.equipment.hrs_monochromator.**HRSMonochromator**(*record*, ***kwargs*)

Bases: *BaseEquipment*

HRS500M Monochromator from Princeton Instruments.

Parameters

- **record** (EquipmentRecord) – The equipment record.
- ****kwargs** – Keyword arguments. Can be specified as attributes of an XML element in a configuration file (with the tag of the element equal to the alias of *record*).

connection: PrincetonInstruments

FRONT_ENTRANCE_SLIT = 2

FRONT_EXIT_SLIT = 3

grating_position_changed: SignalInstance

filter_position_changed: SignalInstance

wavelength_changed: SignalInstance

front_entrance_slit_changed: SignalInstance

front_exit_slit_changed: SignalInstance

`filter_info()`

Returns a description of the filters that are installed in each position.

The keys are the position numbers of each filter.

Return type
dict[int, str]

`get_filter_position()`

Returns the filter position, in the range [1, 6].

Return type
int

`get_front_entrance_slit_width()`

Returns the front entrance slit width (in microns).

Return type
int

`get_front_exit_slit_width()`

Returns the front exit slit width (in microns).

Return type
int

`get_grating_position()`

Returns the current grating position. Either 1, 2 or 3.

Return type
int

`get_wavelength()`

Returns the current wavelength (in nm).

Return type
float

`grating_info()`

Returns the density and blaze values for each grating.

The keys are the position numbers of each grating.

Return type
dict[int, dict[str, str]]

`home_filter_wheel()`

Home the filter wheel.

Return type
int

Returns
The filter wheel position after homing.

`home_front_entrance_slit()`

Home the front entrance slit.

Return type
int

Returns
The slit width (in microns) after homing.

`home_front_exit_slit()`
Home the front exit slit.

Return type
`int`

Returns
The slit width (in microns) after homing.

`set_filter_position(position)`
Set the filter wheel position.

Parameters
position (`int`) – The filter wheel position, in the range [1, 6].

Return type
`int`

Returns
The actual filter wheel position after it has finished moving.

`set_front_entrance_slit_width(um)`
Set the front entrance slit width.

Parameters
um (`int`) – The slit width (in microns).

Return type
`int`

Returns
The actual slit width (in microns) after it has finished moving.

`set_front_exit_slit_width(um)`
Set the front exit slit width.

Parameters
um (`int`) – The slit width (in microns).

Return type
`int`

Returns
The actual slit width (in microns) after it has finished moving.

`set_grating_position(position)`
Set the grating position.

Parameters
position (`int`) – The grating position. Either 1, 2 or 3.

Return type
`int`

Returns
The actual grating position after it has finished moving.

set_wavelength(*nm*)

Set the wavelength.

Parameters
nm (float) – The wavelength (in nm).

Return type
float

Returns
The actual wavelength (in nm) after it has finished moving.

photons.equipment.idq_time_controller module

Time Controller from ID Quantique.

class photons.equipment.idq_time_controller.**Clock**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

INTERNAL = 'INTERNAL'

EXTERNAL = 'EXTERNAL'

INT = 'INTERNAL'

EXT = 'EXTERNAL'

class photons.equipment.idq_time_controller.**Coupling**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

AC = 'AC'

DC = 'DC'

class photons.equipment.idq_time_controller.**Edge**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

RISING = 'RISING'

FALLING = 'FALLING'

class photons.equipment.idq_time_controller.**Mode**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

ACCUMULATE = 'ACCUM'

ACCUM = 'ACCUM'

CYCLE = 'CYCLE'

NIM = 'NIM'

TTL = 'TTL'

HIGH_SPEED = 'LOWRES'

HIGH_RESOLUTION = 'HIRES'

LOW_SPEED = 'HIRES'

LOW_RESOLUTION = 'LOWRES'

FAST = 'LOWRES'

SLOW = 'HIRES'

class photons.equipment.idq_time_controller.**ResyncPolicy**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

AUTO = 'AUTO'

MANUAL = 'MANUAL'

class photons.equipment.idq_time_controller.**Select**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

LOOP = 'LOOP'

OUTPUT = 'OUTPUT'

SHAPED = 'SHAPED'

UNSHAPED = 'UNSHAPED'

class photons.equipment.idq_time_controller.**DelaySettings**(*address*, *value*)

Bases: object

address: str

value: float

to_json()

Return the settings as a JSON serializable object.

Return type

dict[str, str | float]

class photons.equipment.idq_time_controller.**DeviceSettings**(*clock*, *mode*)

Bases: object

clock: *Clock*

mode: *Mode*

to_json()

Return the settings as a JSON serializable object.

Return type
dict[str, str]

class photons.equipment.idq_time_controller.**Histogram**(*hist1*, *hist2*, *hist3*, *hist4*)

Bases: object

hist1: ndarray

hist2: ndarray

hist3: ndarray

hist4: ndarray

to_json()

Return the histogram data as a JSON serializable object.

Return type
dict[str, list]

class photons.equipment.idq_time_controller.**HistogramSettings**(*channel*, *ref*, *stop*, *enabler*, *minimum*, *maximum*, *bin_count*, *bin_width*)

Bases: object

channel: int

ref: str

stop: str

enabler: str

minimum: float

maximum: float

bin_count: int

bin_width: float

to_json()

Return the settings as a JSON serializable object.

Return type
dict[str, str | int | float]

class photons.equipment.idq_time_controller.**InputSettings**(*channel*, *coupling*, *edge*, *enabled*, *delay*, *duration*, *mode*, *resync_policy*, *select*, *threshold*)

Bases: object

channel: int

coupling: *Coupling*

edge: *Edge*

enabled: bool

delay: float

duration: float

mode: *Mode*

resync_policy: *ResyncPolicy*

select: *Select*

threshold: float

to_json()

Return the settings as a JSON serializable object.

Return type
dict[str, str | bool | int | float]

class photons.equipment.idq_time_controller.**StartSettings**(*coupling*, *edge*, *enabled*, *delay*, *duration*, *mode*, *select*, *threshold*)

Bases: object

coupling: *Coupling*

edge: *Edge*

enabled: bool

delay: float

duration: float

mode: *Mode*

select: *Select*

threshold: float

to_json()

Return the settings as a JSON serializable object.

Return type
dict[str, str | bool | float]

class photons.equipment.idq_time_controller.**IDQTimeController**(*record*, ***kwargs*)

Bases: *BaseEquipment*

Time Controller from ID Quantique.

Parameters

- **record** (EquipmentRecord) – The equipment record.
- ****kwargs** – Keyword arguments. Can be specified as attributes of an XML element in a configuration file (with the tag of the element equal to the alias of *record*).

class Clock(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

INTERNAL = 'INTERNAL'

EXTERNAL = 'EXTERNAL'

INT = 'INTERNAL'

EXT = 'EXTERNAL'

class Coupling(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

AC = 'AC'

DC = 'DC'

class Edge(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

RISING = 'RISING'

FALLING = 'FALLING'

class Mode(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

ACCUMULATE = 'ACCUM'

ACCUM = 'ACCUM'

CYCLE = 'CYCLE'

NIM = 'NIM'

TTL = 'TTL'

HIGH_SPEED = 'LOWRES'

HIGH_RESOLUTION = 'HIRES'

LOW_SPEED = 'HIRES'

LOW_RESOLUTION = 'LOWRES'

FAST = 'LOWRES'

SLOW = 'HIRES'

class ResyncPolicy(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

AUTO = 'AUTO'

MANUAL = 'MANUAL'

class Select(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

LOOP = 'LOOP'

OUTPUT = 'OUTPUT'

SHAPED = 'SHAPED'

UNSHAPED = 'UNSHAPED'

connection: ConnectionZeroMQ

counts_changed: SignalInstance

clear_high_resolution_error(*channel*)

Clear the high-resolution error for an input channel.

Parameters

channel (int) – The input channel number (1, 2, 3 or 4).

Return type

None

configure_delay(*, *block*, *address*, *value=0*)

Configure the settings for a DELAY block.

Parameters

- **block** (int) – The DELAY block number (1 through 8).
- **address** (UnionType[int, str, None]) – The address to link the DELAY to. Can be an INPUT channel number (e.g., 1 or 'INPUT1'), 'START' or None.
- **value** (float) – The delay value, in seconds.

Return type

DelaySettings

Returns

The DELAY settings that were read from the device after the settings were written.

configure_device(*, *clock*, *mode*)

Configure the DEVICE settings.

Parameters

- **clock** (*Clock* | str) – Use the internal or external clock.
- **mode** (*Mode* | str) – The resolution mode (high speed or high resolution).

Return type
DeviceSettings

Returns
The DEVICE settings that were read from the device after the settings were written.

configure_histogram(*, *channel*, *ref*, *stop=None*, *enabler='TSGE8'*, *minimum=0*, *maximum=1e-06*, *bin_count=None*, *bin_width=1e-10*)

Configure a HISTOGRAM channel.

Parameters

- **channel** (int) – The HISTOGRAM channel number (1, 2, 3 or 4).
- **ref** (UnionType[int, str, None]) – The INPUT channel number (0[START], 1, 2, 3 or 4) or the name of the reference channel (e.g., 'TSCO5')
- **stop** (UnionType[int, str, None]) – The INPUT channel number (0[START], 1, 2, 3 or 4) or the name of the stop channel (e.g., 'TSCO5'). If not specified, then the value of *ref* is used.
- **enabler** (str) – The timestamp-generator block that determines when data acquisition begins and ends.
- **minimum** (float) – Minimum time value, in seconds.
- **maximum** (float) – Maximum time value, in seconds.
- **bin_count** (Optional[int]) – The number of time bins. If specified, then *maximum* is ignored. Must be between 1 and 16384.
- **bin_width** (float) – The time-bin width, in seconds.

Return type
HistogramSettings

Returns
The HISTOGRAM settings that were read from the device after the settings were written.

configure_input(*, *channel*, *coupling=Coupling.DC*, *delay=0*, *duration=1*, *edge=Edge.RISING*, *enabled=False*, *mode=Mode.CYCLE*, *resync_policy=ResyncPolicy.AUTO*, *select=Select.UNSHAPED*, *threshold=1*)

Configure an INPUT channel.

Parameters

- **channel** (int) – The input channel number (1, 2, 3 or 4).
- **coupling** (*Coupling* | str) – Either AC or DC coupling.

- **delay** (float) – The delay, in seconds, to add to the timestamp when an edge is detected. Must be between 0 and 1 second.
- **duration** (float) – The number of seconds to count edges (integration time). Must be between 0.001 and 65.535 seconds.
- **edge** (*Edge* | str) – The discriminator edge, either RISING or FALLING.
- **enabled** (bool) – Whether the channel is enabled or disabled.
- **mode** (*Mode* | str) – The counter mode (either CYCLE or ACCUMULATE).
- **resync_policy** (*ResyncPolicy* | str) – The resync policy.
- **select** (*Select* | str) – Select what feeds the INPUT block.
- **threshold** (float) – The discriminator threshold value, in volts.

Return type
InputSettings

Returns
The INPUT settings that were read from the device after the settings were written.

configure_start(*, *coupling=Coupling.DC*, *delay=0*, *edge=Edge.RISING*, *enabled=False*, *duration=1*, *mode=Mode.CYCLE*, *select=Select.UNSHAPED*, *threshold=1*)

Configure the START channel.

Parameters

- **coupling** (*Coupling* | str) – Either AC or DC coupling.
- **delay** (float) – The delay, in seconds, to add to the timestamp when an edge is detected. Must be between 0 and 1 second.
- **edge** (*Edge* | str) – The discriminator edge, either RISING or FALLING.
- **enabled** (bool) – Whether the channel is enabled or disabled.
- **duration** (float) – The number of seconds to count edges (integration time). Must be between 0.001 and 65.535 seconds.
- **mode** (*Mode* | str) – The counter mode (either CYCLE or ACCUMULATE).
- **select** (*Select* | str) – Select what feeds the START block.
- **threshold** (float) – The discriminator threshold value, in volts.

Return type
StartSettings

Returns
The START settings that were read from the device after the settings were written.

count_edges(*, *channel*, *allow_zero=False*, *nsamples=1*)

Count the number of edges per second.

Parameters

- **channel** (int) – The input channel number (e.g., 0[START], 1, 2, 3 or 4).

- **allow_zero** (bool) – Whether to allow zero edges per second to be counted. Querying the COUNTER? value from the device immediately returns the value that is stored in the device's memory. After resetting the device (i.e., the value returned by COUNTER? is 0) Python sleeps for *duration* seconds (see *configure_input()*) before querying COUNTER?. Since the computer clock and the device clock are not synced it is possible that COUNTER? is queried before the device writes a value to memory. If zero edges are detected and *allow_zero* is False then this method will block until at least 1 edge is detected.
- **nsamples** (int) – The number of samples to acquire.

Return type
Samples

Returns
The number of edges per second.

has_high_resolution_error(*channel*)

Check if an input channel has a high-resolution error.

Parameters
channel (int) – The input channel number (1, 2, 3 or 4).

Return type
bool

Returns
Whether the specified channel has a high-resolution error.

load(*config*)

Load a pre-defined configuration.

Parameters
config (str) – The configuration to load ('INIT', 'HISTO', 'COUNT', or 'BLANK').

Return type
None

recalibrate()

Recalibrate the Time Controller.

Return type
None

settings_delay(*block*)

Get the settings of a DELAY block.

Parameters
block (int) – The DELAY block number (1 through 8).

Return type
DelaySettings

settings_device()

Get the DEVICE settings.

Return type
DeviceSettings

`settings_histogram`(*channel*)

Get the settings of a HISTOGRAM channel.

Parameters
channel (int) – The HISTOGRAM channel number (1, 2, 3 or 4).

Return type
HistogramSettings

`settings_input`(*channel*)

Get the settings of an INPUT channel.

Parameters
channel (int) – The INPUT channel number (1, 2, 3 or 4).

Return type
InputSettings

`settings_start`()

Get the settings of the START channel.

Return type
StartSettings

`start_stop`(*, *clear=True*, *duration=30*, *enabler='TSGE8'*, *min_events=0*, *timeout=None*)

Acquire a start-stop histogram of the duration between two edges.

Parameters

- **clear** (bool) – Whether to clear the histogram data before acquiring data.
- **duration** (float) – The number of seconds to acquire data for.
- **enabler** (str) – The timestamp-generator block that determines when data acquisition begins and ends.
- **min_events** (Union[int, Sequence[int]]) – The minimum number of start-stop events, on each histogram channel, that must occur before returning to the calling program. If specified, then iteratively acquires data for *duration* seconds until the specified number of events has occurred.
- **timeout** (Optional[float]) – The maximum number of seconds to wait for *min_events* to occur. If a timeout occurs, the data that has been acquired is returned (an error is not raised).

Return type
Histogram

Returns
The histogram data.

photons.equipment.keithley_6430 module

Keithley 6430 sub-femtoAmp SourceMeter.

class photons.equipment.keithley_6430.**Keithley6430**(*record*, ***kwargs*)

Bases: *DMM*

Keithley 6430 sub-femtoAmp SourceMeter.

Parameters

- **record** (EquipmentRecord) – The equipment record.
- ****kwargs** – Keyword arguments. Can be specified as attributes of an XML element in a configuration file (with the tag of the element equal to the alias of *record*).

MODES: dict[str, str] = {'FIX': 'FIXED', 'FIXED': 'FIXED', 'LIST': 'LIST', 'SWE': 'SWEEP', 'SWEEP': 'SWEEP'}

TRIGGERS: dict[str, str] = {'BST': 'BSTEST', 'BSTEST': 'BSTEST', 'BUS': 'BUS', 'IMM': 'IMMEDIATE', 'IMMEDIATE': 'IMMEDIATE', 'MAN': 'MANUAL', 'MANUAL': 'MANUAL', 'NST': 'NSTEST', 'NSTEST': 'NSTEST', 'PST': 'PSTEST', 'PSTEST': 'PSTEST', 'TIM': 'TIMER', 'TIMER': 'TIMER', 'TLIN': 'TLINK', 'TLINK': 'TLINK'}

source_settings_changed: SignalInstance

check_errors()

Query the error queue of the SourceMeter.

If there is an error then raise an exception.

Return type
None

configure(*, *function='current'*, *range=1.05e-06*, *nsamples=10*, *nplc=10*, *auto_zero=True*, *trigger='bus'*, *edge=None*, *ntriggers=1*, *delay=0.0*, *cmpl=0.000105*, *enable=True*)

Configure the Sense subsystem.

Parameters

- **function** (str) – The function to measure. Can be any key in DMM.FUNCTIONS (case insensitive).
- **range** (float | str) – The range to use for the measurement. Can be any key in DMM.RANGES.
- **nsamples** (int) – The number of samples to acquire after a trigger event.
- **nplc** (int) – The number of power-line cycles.
- **auto_zero** (bool | int | str) – The auto-zero mode. Can be any key in DMM.AUTO.
- **trigger** (str) – The trigger mode. Can be any key in *Keithley6430.TRIGGERS* (case insensitive).
- **edge** (None) – Not supported and must be None (the edge to trigger on).

- **ntriggers** (int) – The number of triggers that are accepted by the SourceMeter before returning to the wait-for-trigger state.
- **delay** (float) – The trigger delay in seconds (auto delay is not supported).
- **cmpl** (float) – The protection (compliance) value.
- **enable** (bool) – Whether to enable the output.

Return type
dict[str, ...]

Returns
The result of *settings()* after applying the configuration.

configure_source(*, *function='current'*, *range=1e-06*, *nsamples=10*, *auto_zero=True*, *trigger='bus'*, *delay=0.0*, *mode='fixed'*, *cmpl=0.01*, *cmpl_range=0.21*)

Configure the Source subsystem.

Parameters

- **function** (str) – The output source. Can be any key in DMM.FUNCTIONS (case insensitive).
- **range** (float) – The range to use for the output level.
- **nsamples** (int) – The number of samples to acquire after a trigger event.
- **auto_zero** (bool | int | str) – The auto-zero mode. Can be any key in DMM.AUTO.
- **trigger** (str) – The trigger mode. Can be any key in *Keithley6430.TRIGGERS* (case insensitive).
- **delay** (float) – The trigger delay in seconds.
- **mode** (str) – The output mode. Can be any key in *Keithley6430.MODES* (case insensitive).
- **cmpl** (float) – The protection (compliance) value.
- **cmpl_range** (float) – The range to use for the protection (compliance).

Return type
dict[str, ...]

Returns
The result of *settings_source()* after applying the configuration.

disable_output()

Turn the output off.

Return type
None

enable_output()

Turn the output on.

Return type
None

get_output_level()

Returns the output level.

Return type
float

is_output_enabled()

Returns whether the source output is on or off.

Return type
bool

is_output_stable(*tol=0.001*)

Whether the output level has stabilized.

You must call *set_output_level()* once before calling this method, otherwise the level comparison is meaningless.

Parameters
tol (float) – The fractional tolerance for the output to be considered stable.

Return type
bool

set_output_level(*level*, *, *wait=False*, *tol=0.001*, *timeout=60*)

Set the level of the Source output.

Parameters

- **level** (float) – The value to set the output to.
- **wait** (bool) – Whether to wait for the output level to stabilize before returning.
- **tol** (float) – The fractional tolerance for the level to be considered stable.
- **timeout** (float) – The maximum number of seconds to wait.

Return type
None

settings()

Return type
dict[str, ...]

Returns the configuration settings of the Sense subsystem.

```
{
  'auto_range': str,
  'auto_zero': str,
  'cmpl': float,
  'function': str,
  'nplc': float,
  'nsamples': 1,
  'range': float,
  'trigger_count': int,
  'trigger_delay': float,
```

(continues on next page)

(continued from previous page)

```
  'trigger_delay_auto': False,
  'trigger_edge': 'N/A',
  'trigger_mode': str
}
```

settings_source()

Return type
dict[str, ...]

Returns the configuration settings of the Source subsystem.

```
{
  'auto_zero': str,
  'cmpl': float,
  'cmpl_range': float,
  'function': str,
  'mode': str,
  'nsamples': int,
  'range': float,
  'trigger_delay': float,
  'trigger_mode': str
}
```

disconnect_equipment()

Turn the output off and disconnect.

Return type
None

photons.equipment.kinesis module

Base class for equipment that use the Kinesis SDK from Thorlabs.

class photons.equipment.kinesis.**KinesisBase**(*record*, ***kwargs*)

Bases: *BaseEquipment*

Base class for equipment that use the Kinesis SDK from Thorlabs.

Parameters

- **record** (EquipmentRecord) – The equipment record.
- ****kwargs** – Keyword arguments. Can be specified as attributes of an XML element in a configuration file (with the tag of the element equal to the alias of *record*).

connection: MotionControl

MOVING_CLOCKWISE = 16

MOVING_COUNTERCLOCKWISE = 32

JOGGING_CLOCKWISE = 64

JOGGING_COUNTERCLOCKWISE = 128

HOMING = 512

HOMED = 1024

MOVING = 752

static build_device_list()

Builds the Thorlabs device list.

Only builds the device list once per application instance. This function can be called multiple times.

Return type
None

get_position()

Get the current position of the device.

Return type
int | float

info()

Return information about the device.

The subclass must populate the dict.

Return type
dict

is_moving(*delay=0.2*)

Returns whether the device moving.

Parameters
delay (float) – The number of seconds to wait before checking whether the device is moving. Starting from rest, the motors take time to start moving the device and if this method is called too soon after requesting the device to move to a new position then the callback that checks the moving status might indicate that the motors are not currently moving.

Return type
bool

set_position(*position*)

Set the current position of the device.

Return type
None

status_bits()

Returns the device status bits.

This method gets called automatically in the registered callback.

Return type
int

wait(*timeout=None*)

Wait for the device to stop moving.

Parameters
timeout (float) – The maximum number of seconds to wait. Default is to wait forever.

Return type
None

class photons.equipment.kinesis.**Signaler**(*kinesis*)

Bases: QObject

Qt Signaler for callbacks that are received from the DLL.

position_changed: SignalInstance

photons.equipment.kinesis.**callback**(*signaler*)

Create a callback for the *signaler*.

photons.equipment.laser_superk module

SuperK Fianium laser from NKT Photonics.

class photons.equipment.laser_superk.**ID60**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: IntEnum

Register IDs for "SK Fianium" (Module type 0x0060).

INLET_TEMPERATURE = 17

EMISSION = 48

MODE = 49

INTERLOCK = 50

PULSE_PICKER_RATIO = 52

WATCHDOG_INTERVAL = 54

POWER_LEVEL = 55

CURRENT_LEVEL = 56

NIM_DELAY = 57

SERIAL_NUMBER = 101

STATUS_BITS = 102

SYSTEM_TYPE = 107

USER_TEXT = 108

class photons.equipment.laser_superk.**ID88**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: IntEnum

Register IDs for "SuperK G3 Mainboard" (Module type 0x0088).

INLET_TEMPERATURE = 17

EMISSION = 48

MODE = 49

INTERLOCK = 50

DATETIME = 51

PULSE_PICKER_RATIO = 52

WATCHDOG_INTERVAL = 54

CURRENT_LEVEL = 55

PULSE_PICKER_NIM_DELAY = 57

MAINBOARD_NIM_DELAY = 58

USER_CONFIG = 59

MAX_PULSE_PICKER_RATIO = 61

STATUS_BITS = 102

ERROR_CODE = 103

USER_TEXT = 141

class photons.equipment.laser_superk.**ID61**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: IntEnum

Register IDs for "SuperK Front panel" (Module type 0x61).

PANEL_LOCK = 61

DISPLAY_TEXT = 114

ERROR_FLASH = 141

class photons.equipment.laser_superk.**ID89**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: IntEnum

Register IDs for "SuperK G3 Front Panel" (Module type 0x0089).

According to the NKT engineers, there are no front-panel registers available. This means that the PANEL_LOCK, DISPLAY_TEXT and ERROR_FLASH do not exist.

class photons.equipment.laser_superk.**OperatingModes**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: IntEnum

The operating modes for a SuperK Fianium laser.

CONSTANT_CURRENT = 0

CONSTANT_POWER = 1

MODULATED_CURRENT = 2

MODULATED_POWER = 3

POWER_LOCK = 4

class photons.equipment.laser_superk.**SuperK**(*record*, ***kwargs*)

Bases: *BaseEquipment*

SuperK Fianium laser from NKT Photonics.

Parameters

- **record** (EquipmentRecord) – The equipment record.
- ****kwargs** – Keyword arguments. Can be specified as attributes of an XML element in a configuration file (with the tag of the element equal to the alias of *record*).

connection: NKT

class **OperatingModes**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: IntEnum

The operating modes for a SuperK Fianium laser.

CONSTANT_CURRENT = 0

CONSTANT_POWER = 1

MODULATED_CURRENT = 2

MODULATED_POWER = 3

POWER_LOCK = 4

DEVICE_ID = 15

FRONT_PANEL_ID = 1

MODULE_TYPE_0x60 = 96

MODULE_TYPE_0x88 = 136

level_changed: SignalInstance

emission_changed: SignalInstance

mode_changed: SignalInstance

emission(*enable*)

Turn the laser emission on or off.

Parameters
enable (bool) – Whether to turn the laser emission on or off.

Return type
None

enable_constant_current_mode()

Set the laser to be in constant current mode.

Return type
None

enable_constant_power_mode()

Set the laser to be in constant power mode.

Return type
None

enable_modulated_current_mode()

Set the laser to be in modulated current mode.

Return type
None

enable_modulated_power_mode()

Set the laser to be in modulated power mode.

Return type
None

enable_power_lock_mode()

Set the laser to be power lock (external feedback) mode.

Return type
None

ensure_interlock_ok()

Make sure that the interlock is okay.

Raises an exception if it is not okay, and it cannot be reset.

Return type
bool

get_current_level()

Returns the constant/modulated current level of the laser.

Return type
float

get_feedback_level()

Get the power lock (external feedback) level of the laser.

Return type
float

get_operating_mode()

Returns the operating mode of the laser.

Return type
OperatingModes

get_operating_modes()

Get all supported operating modes of the laser.

Return type
dict[str, *OperatingModes*]

get_power_level()

Returns the constant/modulated power level of the laser.

Return type
float

get_temperature()

Returns the temperature of the laser.

Return type
float

get_user_text()

Returns the custom user-text value.

Return type
str

is_constant_current_mode()

Whether the laser in constant current mode.

Return type
bool

is_constant_power_mode()

Whether the laser in constant power mode.

Return type
bool

is_emission_on()

Check if the laser emission is on or off.

Return type
bool

is_modulated_current_mode()

Whether the laser in modulated current mode.

Return type
bool

`is_modulated_power_mode()`

Whether the laser in modulated power mode.

Return type
bool

`is_power_lock_mode()`

Whether the laser in power lock (external feedback) mode.

Return type
bool

`lock_front_panel(lock)`

Lock the front panel so that the level cannot be changed manually.

Parameters
lock (bool) – Whether to lock (True) or unlock (False) the front panel.

Return type
bool

Returns
Whether the request to (un)lock the front panel was successful. A laser with a module type 0x88 does not permit the front panel to be (un)locked and therefore this method will always return False for this laser.

`set_current_level(percentage)`

Set the constant/modulated current level of the laser.

Parameters
percentage (float) – The current level as a percentage 0 - 100 (resolution 0.1).

Return type
float

Returns
The actual current level that the laser is at.

`set_feedback_level(percentage)`

Set the power-lock (external feedback) level of the laser.

Parameters
percentage (float) – The power-lock level as a percentage 0 - 100 (resolution 0.1).

Return type
float

Returns
The power-lock level that the laser is at.

`set_operating_mode(mode)`

Set the operating mode of the laser.

Parameters
mode (int | str | *OperatingModes*) – The operating mode. Can be an *OperatingModes* value or member name.

Return type
None

set_power_level(*percentage*)

Set the constant/modulated power level of the laser.

Parameters
percentage (float) – The power level as a percentage 0 - 100 (resolution 0.1).

Return type
float

Returns
The actual power level that the laser is at.

set_user_text(*text*)

Set the custom user-text value.

Parameters
text (str) – The text to write to the laser's firmware. Only ASCII characters are allowed. The maximum number of characters is 20 for the laser with module type 0x60 and 240 characters for module type 0x88. The laser with module type 0x60 can display the text on the front panel (if selected from the menu option).

Return type
str

Returns
The text that was actually stored in the laser's firmware.

disconnect_equipment()

Unlock the front panel, set the user text to an empty string and close the port.

class photons.equipment.laser_superk.**Signaler**(*device*)

Bases: QObject

Qt Signaler for callbacks that are received from the DLL.

device_status_changed: SignalInstance

register_status_changed: SignalInstance

port_status_changed: SignalInstance

maybe_emit_notification(**args*, ***kwargs*)

Notify all linked Clients.

Return type
None

photons.equipment.laser_superk.**register_callbacks**(*superk*)

Register the callbacks from the DLL.

Return type
Signaler

photons.equipment.nidaq module

DAQ from National Instruments.

class photons.equipment.nidaq.**Timing**(***kwargs*)

Bases: object

Do not instantiate this class directly. Use *NIDAQ.timing()*.

add_to(*task*)

Add the timing configuration to a task.

Parameters
task (Task) – The task to add the timing configuration to.

Return type
None

property rate: float

Returns the sample rate (in Hz).

property sample_mode: AcquisitionType

Returns the sample mode.

property samples_per_channel: int

Returns the number of samples per channel to acquire or generate.

class photons.equipment.nidaq.**Trigger**(***kwargs*)

Bases: object

Do not instantiate this class directly. Use *NIDAQ.trigger()*.

add_to(*task*)

Add this trigger to a task.

Parameters
task (Task) – The task to add the trigger event to.

Return type
None

class photons.equipment.nidaq.**NIDAQ**(*record*, ***kwargs*)

Bases: *BaseEquipment*

DAQ from National Instruments.

Parameters

- **record** (EquipmentRecord) – The equipment record.
- ****kwargs** – Keyword arguments. Can be specified as attributes of an XML element in a configuration file (with the tag of the element equal to the alias of *record*).

connection: ConnectionNIDAQ

counts_changed: SignalInstance

WAIT_INFINITELY = -1.0

analog_in(*channel*, *, *config='DIFF'*, *duration=None*, *maximum=10*, *minimum=-10*, *nsamples=1*, *timeout=10*, *timing=None*, *trigger=None*, *wait=True*)

Read the voltage(s) of the analog-input channel(s).

Parameters

- **channel** (int | str) – The analog-input channel number(s), e.g., channel=0, channel='0:7'.
- **config** (int | str) – Specifies the input terminal configuration for the channel, see TerminalConfiguration.
- **duration** (float) – The number of seconds to read voltages for. If specified then this value is used instead of *nsamples*.
- **maximum** (float) – The maximum voltage that is expected to be measured.
- **minimum** (float) – The minimum voltage that is expected to be measured.
- **nsamples** (int) – The number of samples per channel to read. If a *duration* is also specified then that value is used instead of *nsamples*.
- **timeout** (float) – The maximum number of seconds to wait for the task to finish. Set to -1 to wait forever.
- **timing** (*Timing*) – The timing settings to use. See *timing()*.
- **trigger** (*Trigger*) – The trigger settings to use. See *trigger()*.
- **wait** (bool) – Whether to wait for the task to finish. If enabled then also closes the task when it is finished.

Return type

tuple[ndarray | Task, float]

Returns

If *wait* is True then the voltage(s) of the requested analog-input channel(s) and the time interval between samples (i.e., dt) are returned. Otherwise, the analog-input task, which has *not* been started yet and the time interval between samples are returned. Not starting the task allows one to register a callback before starting the task.

Examples

Read the value of a single analog-input channel

```
>>> daq.analog_in(0)
(array([-0.48746041]), 0.001)
>>> daq.analog_in(6, nsamples=5)
(array([-0.44944232, -0.45040888, -0.45137544, -0.45556387, -0.
↪45298637]), 0.001)
```

Read the values of multiple analog-input channels

```
>>> daq.analog_in('0:3', nsamples=4)
(array([[ 0.03512726,  0.03770475,  0.03867132,  0.03512726],
       [-0.1675285 ,  0.17527869, -0.17171693,  0.17237901],
       [ 0.08248878,  0.12243999,  0.00741916,  0.07991128],
       [ 0.08861033,  0.09859814,  0.05832474,  0.06831254]]), 0.001)
```

analog_out(*channel*, *voltage*, *, *auto_start=True*, *timeout=10*, *timing=None*, *trigger=None*, *wait=True*)

Write the voltage(s) to the analog-output channel(s).

Parameters

- **channel** (int | str) – The analog-output channel number(s), e.g., channel=0, channel='0:1'.
- **voltage** (float | list[float] | list[list[float]] | ndarray) – The voltage(s) to output.
- **auto_start** (bool) – Whether to automatically start the task.
- **timeout** (float) – The maximum number of seconds to wait for the task to finish. Set to -1 to wait forever.
- **timing** (*Timing*) – The timing settings to use. See *timing()*.
- **trigger** (*Trigger*) – The trigger settings to use. See *trigger()*.
- **wait** (bool) – Whether to wait for the task to finish. If enabled then also closes the task when it is finished.

Return type
Task

Returns
The analog-output task.

Examples

Write to a single analog-output channel

```
>>> daq.analog_out(0, 1.123)
```

Write to multiple analog-output channels

```
>>> daq.analog_out('0:1', [0.2, -1.2])
>>> daq.analog_out('0:1', [[0.2, 0.1, 0.], [-0.1, 0., 0.1]])
```

analog_out_read(*channel*, ***kwargs*)

Read the output voltage(s) from the analog-output channel(s).

Parameters

- **channel** (int | str) – The analog-output channel number(s), e.g., channel=0, channel='0:1'.
- ****kwargs** – All keyword arguments are passed to *analog_in()*.

Return type
tuple[ndarray | Task, float]

Returns
If *wait* is True then the voltage(s) of the requested analog-output channel(s) and the time interval between samples (i.e., dt) are returned. Otherwise, the analog-output task, which has *not* been started yet and the time interval between samples are returned. Not starting the task allows one to register a callback before starting the task.

Examples

Read a single value from an analog-output channel

```
>>> daq.analog_out_read(0)
(array([-1.09800537]), 0.001)
```

Read multiple values from multiple analog-output channels

```
>>> daq.analog_out_read('0:1', nsamples=4)
(array([[-1.09832756, -1.09736099, -1.09800537, -1.09736099],
       [ 0.21168585,  0.21233022,  0.21200803,  0.21168585]]), 0.001)
```

close_all_tasks()
Close all tasks.

Return type
None

count_edges(*pfi*, *duration*, *, *nsamples=1*, *rising=True*)
Count the number of edges per second.

Parameters

- **pfi** (int) – The PFI terminal number.
- **duration** (float) – The number of seconds to count edges for.
- **nsamples** (int) – The number of times to count edges for *duration* seconds.
- **rising** (bool) – Whether to count rising edges, otherwise count falling edges.

Return type
Samples

Returns
The number of edges per second.

digital_in(*lines*, *, *port=1*)
Read the state of the digital-input channel(s).

Parameters

- **lines** (int | str) – The line number(s) (e.g., line=1, line='0:7', line='/Dev1/port0/line0:7,/Dev1/port1/line0:3').
- **port** (int) – The port number.

Return type
bool | list[bool]

Returns
Whether the requested digital input channel(s) are HIGH or LOW.

Examples

Read the state of a single digital-input channel (P1.0)

```
>>> daq.digital_in(0)
False
```

Read the state of a single digital-input channel (P0.2)

```
>>> daq.digital_in(2, port=0)
True
```

Read the state of multiple digital-input channels (P1.0-7)

```
>>> daq.digital_in('0:7')
[False, False, True, False, False, False, False, True]
```

digital_out(*lines*, *state*, *, *auto_start=True*, *port=1*, *timeout=10*, *timing=None*, *trigger=None*, *wait=True*)

Write the state of digital-output channels(s).

Parameters

- **lines** (int | str) – The line number(s) (e.g., line=1, line='0:7', line='/Dev1/port0/line0:7,/Dev1/port1/line0:3').
- **state** (bool | list[bool] | list[list[bool]]) – Whether to set the specified line(s) to HIGH or LOW.
- **auto_start** (bool) – Whether to automatically start the task.
- **port** (int) – The port number.
- **timeout** (float) – The maximum number of seconds to wait for the task to finish. Set to -1 to wait forever.
- **timing** (*Timing*) – The timing settings to use. See *timing()*.
- **trigger** (*Trigger*) – The trigger settings to use. See *trigger()*.
- **wait** (bool) – Whether to wait for the task to finish. If enabled then also closes the task when it is finished.

Return type
Task

Returns
The digital-output task.

Examples

Set the state of a single digital-output channel (P1.0)

```
>>> daq.digital_out(0, True)
```

Set multiple digital-output channels to be in the same state (P2.0-7)

```
>>> daq.digital_out('0:7', False, port=2)
```

Set the state of multiple digital-output channels (P1.2-4)

```
>>> daq.digital_out('2:4', [False, True, True])
```

digital_out_read(*lines*, *, *port=1*)

Read the state of digital-output channel(s).

Parameters

- **lines** (int | str) – The line number(s) (e.g., line=1, line='0:7', line='/Dev1/port0/line0:7,/Dev1/port1/line0:3').
- **port** (int) – The port number.

Return type
bool | list[bool]

Returns
Whether the requested digital-output channel(s) are HIGH or LOW.

Examples

Read the state of a single digital-output channel (P1.0)

```
>>> daq.digital_out_read(0)
True
```

Read the state of a single digital-output channel (P0.5)

```
>>> daq.digital_out_read(5, port=0)
False
```

Read the state of multiple digital-output channels (P1.0-7)

```
>>> daq.digital_out_read('0:7')
[False, True, True, False, True, False, False, False]
```

edge_separation(*start*, *stop*, *, *maximum=1.0*, *minimum=1e-07*, *nsamples=10*, *start_edge='RISING'*, *stop_edge='FALLING'*, *timeout=10*)

Get the duration, in seconds, between two edges.

Parameters

- **start** (int) – The PFI terminal number to use for the start time, t=0.

- **stop** (int) – The PFI terminal number to use for the stop time, t=dt. Can be same as *start* provided that *start_edge* and *stop_edge* are different values.
- **maximum** (float) – The maximum time, in seconds, between the start-stop edges that is expected.
- **minimum** (float) – The minimum time, in seconds, between the start-stop edges that is expected.
- **nsamples** (int) – The number of start-stop samples to acquire.
- **start_edge** (int | str) – Specifies on which edge to start each measurement. See Edge for allowed values.
- **stop_edge** (int | str) – Specifies on which edge to stop each measurement. See Edge for allowed values.
- **timeout** (float) – The maximum number of seconds to wait for the task to finish. Set to -1 to wait forever.

Return type
Samples

Returns
The duration(s), in seconds, between the start-stop edges.

function_generator(*channel*, *, *amplitude=1*, *duty=0.5*, *frequency=1000*, *offset=0*, *nsamples=1000*, *phase=0*, *preview=False*, *symmetry=1.0*, *trigger=None*, *waveform='sine'*)

Generate a waveform.

Parameters

- **channel** (int | str) – The analog-output channel number(s), e.g., channel=0, channel='0:1'.
- **amplitude** (float) – The zero-to-peak amplitude of the waveform to generate in volts. Zero and negative values are valid.
- **duty** (float) – The duty cycle of the square wave. Must be in the interval [0, 1]. Only used if *waveform* is square.
- **frequency** (float) – The frequency of the waveform to generate, in Hz.
- **offset** (float) – The voltage offset of the waveform to generate.
- **nsamples** (int) – The number of voltage samples per waveform period.
- **phase** (float) – The phase of the waveform, in degrees.
- **preview** (bool) – Whether to return a ndarray of a single period of the waveform voltages.
- **symmetry** (float) – The symmetry of the ramp. Corresponds to the ratio of the rising portion of the ramp to the ramp period. For example, a symmetry of 0.5 corresponds to a triangle wave. Must be in the interval [0, 1]. Only used if *waveform* is ramp.
- **trigger** (*Trigger*) – The trigger settings to use. See *trigger()*.

- **waveform** (str) – Specifies the kind of waveform to generate. Can be: sine, square, ramp, triangle, sawtooth.

Return type
Task | ndarray

Returns
The analog-output task or a single period of the waveform if *preview* is True.

info()

Returns the driver information about the NIDAQ board.

Return type
dict[str, int]

pulse(*pfi*, *duration*, *, *ctr=1*, *delay=0*, *npulses=1*, *state=True*, *timeout=10*, *wait=True*)

Generate one (or more) digital pulse(s).

If *state* is True then the *pfi* terminal will output 0V for *delay* seconds, generate *npulses* +5V pulse(s) (each with a width of *duration* seconds) and then remain at 0V when the task is done.

If *state* is False then the *pfi* terminal will output +5V for *delay* seconds, generate *npulses* 0V pulse(s) (each with a width of *duration* seconds) and then remain at +5V when the task is done.

Parameters

- **pfi** (int) – The PFI terminal number to output the pulse(s) from.
- **duration** (float) – The duration (width) of each pulse, in seconds.
- **ctr** (int) – The counter terminal number to use for timing.
- **delay** (float) – The number of seconds to wait before generating the first pulse.
- **npulses** (int) – The number of pulses to generate.
- **state** (bool) – Whether to generate HIGH or LOW pulse(s).
- **timeout** (float) – The maximum number of seconds to wait for the task to finish. Set to -1 to wait forever.
- **wait** (bool) – Whether to wait for the task to finish. If enabled then also closes the task when it is finished.

Return type
Task

Returns
The task.

Examples

Generate a single HIGH pulse for 0.1 seconds from PFI2

```
>>> daq.pulse(2, 0.1)
```

storm(*camera*, *sequence*)

Create a task for STORM/PALM acquisition.

For example, for a 4-frame sequence controlling two lasers:

```
sequence = {
  'port0/line0': [True, False, False, False],
  'port0/line1': [False, True, True, True]
}
```

Parameters

- **camera** (int) – The PFI terminal number that the camera's Fire signal is connect to.
- **sequence** (dict) – The keys are the digital-output terminals that turn the laser pulses on/off and the values represent the state of the lasers in each frame.

Return type
Task

Returns
The task.

timing(*, *finite=True*, *pfi=None*, *rate=1000*, *rising=True*)

Configure and return the sample clock to add to a task.

Parameters

- **finite** (bool) – Whether to acquire/generate a continuous or a finite number of samples.
- **pfi** (int) – The PFI terminal number to use as the external sample clock. If not specified then uses the default onboard clock of the device.
- **rate** (float) – The sampling rate in Hz. If you specify an external sample clock (i.e., a value for *pfi*) then set the *rate* to be the maximum expected rate of the external clock.
- **rising** (bool) – Whether to acquire/generate samples on the rising or falling edge of the sample clock.

Return type
Timing

Returns
The timing instance.

trigger(*source*, *, *delay=0*, *hysteresis=0*, *level=None*, *retriggerable=False*, *rising=True*)

Configure and return a trigger to add to a task.

Parameters

- **source** (int | str) – Either a PFI or an AI channel number or a terminal name to use as the trigger source.
- **delay** (float) – The time (in seconds) between the trigger event and when to acquire/generate samples. Can be < 0 to acquire/generate samples before the trigger event (only if the NIDAQ task supports it).
- **hysteresis** (float) – A hysteresis level (in volts). Only applicable for an analog trigger.
- **level** (float) – The voltage level to use for the trigger signal. Whether this value is set decides whether the trigger source is from a digital or an analog channel. If None then *channel* refers to a PFI channel (a digital trigger), otherwise, *channel* refers to an AI channel (an analog trigger).
- **retriggerable** (bool) – Whether the task can be retriggered.
- **rising** (bool) – Whether to use the rising or falling edge(slope) of the digital(analog) trigger signal.

Return type
Trigger

Returns
The trigger instance.

staticMetaObject = PySide6.QtCore.QMetaObject("NIDAQ" inherits "BaseEquipment": Methods: #5 type=Signal, signature=counts_changed(PyObject), parameters=PyObject)

static time_array(*n*, *dt*)

Create an array based on a sampling time.

Parameters

- **n** (int | ndarray) – The number of samples. If an array of voltage samples is passed in, then the returned time array will have the appropriate size.
- **dt** (float) – The sampling time.

Return type
ndarray

Returns
The array (e.g., [0, dt, 2*dt, 3*dt, ..., (n-1)*dt]).

static wait_until_done(**tasks*, *timeout=10.0*)

Wait until all tasks are done and then close each task.

Parameters

- **tasks** (Task) – The task(s) to wait for.
- **timeout** (float) – The number of seconds to wait for each task to finish. Set to -1 to wait forever.

Return type
None

photons.equipment.oscilloscope module

Base class for an oscilloscope.

class photons.equipment.oscilloscope.**Oscilloscope**(*record*, ***kwargs*)

Bases: *BaseEquipment*

Base class for an oscilloscope.

Parameters

- **record** (EquipmentRecord) – The equipment record.
- ****kwargs** – Keyword arguments. Can be specified as attributes of an XML element in a configuration file (with the tag of the element equal to the alias of *record*).

connection: ConnectionMessageBased

configure_channel(*channel*, ***kwargs*)

Configure a channel.

Return type
None

configure_timebase(***kwargs*)

Configure the timebase.

Return type
None

configure_trigger(***kwargs*)

Configure the trigger.

Return type
None

run()

Start acquiring waveform data.

Return type
None

single()

Capture and display a single acquisition.

Return type
None

trigger()

Send a software trigger.

Return type
None

stop()

Stop acquiring waveform data.

Return type
None

waveform(**channels*, ***kwargs*)
Get the waveform data.

Return type
ndarray

photons.equipment.oscilloscope_rigol module

An oscilloscope from Rigol.

class photons.equipment.oscilloscope_rigol.**RigolOscilloscope**(*record*, ***kwargs*)
Bases: *Oscilloscope*

An oscilloscope from Rigol.

Base class for an oscilloscope.

Parameters

- **record** (EquipmentRecord) – The equipment record.
- ****kwargs** – Keyword arguments. Can be specified as attributes of an XML element in a configuration file (with the tag of the element equal to the alias of *record*).

clear()
Clears the event registers and the error queue.

Return type
None

configure_channel(*channel*, ***, *bw_limit=False*, *coupling='DC'*, *enable=True*, *invert=False*, *offset=0*, *probe=1*, *scale=1*)
Configure a channel.

Parameters

- **channel** (int) – The channel number.
- **bw_limit** (bool) – Whether to enable or disable the bandwidth limit.
- **coupling** (Literal['DC', 'AC', 'GND']) – The coupling mode (either DC, AC or GND).
- **enable** (bool) – Whether to enable or disable the channel.
- **invert** (bool) – Whether to invert the waveform.
- **offset** (float) – The vertical offset [Volts].
- **probe** (float) – The probe ratio. Only discrete values are allowed (see manual).
- **scale** (float) – The vertical scale [Volts/div].

Return type
None

configure_timebase(*, *mode='MAIN'*, *offset=0*, *scale=1e-06*)

Configure the timebase.

Parameters

- **mode** (Literal['MAIN', 'XY', 'ROLL']) – The timebase mode (either MAIN, XY or ROLL).
- **offset** (float) – The horizontal offset [seconds].
- **scale** (float) – The horizontal scale [seconds/div].

Return type
None

configure_trigger(*, *channel=1*, *coupling='DC'*, *holdoff=1.6e-08*, *level=0*, *noise_reject=True*, *slope='POS'*, *sweep='AUTO'*)

Configure the Edge trigger.

Parameters

- **channel** (int | str) – The channel to use as the trigger source.
- **coupling** (Literal['AC', 'DC', 'LFReject', 'HFReject']) – The trigger coupling type (either AC, DC, LFReject or HFReject).
- **holdoff** (float) – The trigger holdoff time [seconds].
- **level** (float) – The voltage level to trigger at.
- **noise_reject** (bool) – Whether to enable or disable noise rejection.
- **slope** (Literal['POS', 'NEG', 'RFAL']) – The slope edge to trigger on (either POS, NEG or RFAL).
- **sweep** (Literal['AUTO', 'NORMAL', 'SINGLE']) – The sweep mode (either AUTO, NORMAL or SINGLE).

Return type
None

run()

Start acquiring waveform data.

Return type
None

single()

Capture and display a single acquisition.

Return type
None

trigger()

Send a software trigger.

Return type
None

stop()

Stop acquiring waveform data.

Return type
None

waveform(**channels*, *displayed=True*)

Get the waveform data.

Parameters

- ***channels** (int | str) – The channel(s) to read (e.g., 1, 'CHAN1', 'channel1', 'D6').
- **displayed** (bool) – Whether to read the waveform data displayed on the screen or from internal memory. If reading from internal memory then *stop()* is called prior to reading the data.

Return type
ndarray

Returns
The waveform data.

photons.equipment.shot702_controller module

OptoSigma SHOT-702 controller.

class photons.equipment.shot702_controller.**OptoSigmaSHOT702**(*record*, ***kwargs*)

Bases: *BaseEquipment*

OptoSigma SHOT-702 controller.

Parameters

- **record** (EquipmentRecord) – The equipment record.
- ****kwargs** – Keyword arguments. Can be specified as attributes of an XML element in a configuration file (with the tag of the element equal to the alias of *record*).

connection: SHOT702

NUM_PULSES_PER_360_DEGREES = 144000

angle_changed: SignalInstance

property degrees_per_pulse: float

Returns the number of degrees per pulse.

degrees_to_position(*degrees*)

Convert an angle, in degrees, to an encoder position.

Parameters
degrees (float) – An angle, in degrees.

Return type
int

Returns
The corresponding encoder position.

`get_angle()`

Returns the angle (in degrees) of the filter wheel.

Return type
`float`

`get_speed()`

Get speed that the stage moves to a new angle.

Return type
`tuple[int, int, int]`

Returns

The minimum speed (in number of pulses per second),
the maximum speed (in number of pulses per second) and the acceleration/deceleration time in ms.

`get_speed_home()`

Get speed that the stage moves home.

Return type
`tuple[int, int, int]`

Returns

The minimum speed (in number of pulses per second),
the maximum speed (in number of pulses per second), and the acceleration/deceleration time in ms.

`home(*, wait=True, timeout=300)`

Home the continuously-variable filter wheel.

Parameters

- **wait** (`bool`) – Whether to wait for the wheel to stop moving.
- **timeout** (`float`) – The maximum number of seconds to wait for the wheel to stop moving.

Return type
`None`

`is_moving()`

Returns whether the filter wheel is moving.

Return type
`bool`

`position_to_degrees(position, *, bound=False)`

Convert an encoder position to an angle in degrees.

Parameters

- **position** (`int`) – The encoder position.
- **bound** (`bool`) – Whether to bound the angle to be between [0, 360) degrees.

Return type
float

Returns
The angle in degrees.

set_angle(*degrees*, *, *wait=True*, *timeout=300*)

Set the angle of the continuously-variable filter wheel.

Parameters

- **degrees** (float) – The angle, in degrees.
- **wait** (bool) – Whether to wait for the wheel to stop moving.
- **timeout** (float) – The maximum number of seconds to wait for the wheel to stop moving.

Return type
None

set_speed(*minimum*, *maximum*, *acceleration*)

Set speed that the stage moves to a new angle.

According to the manual:

```
Max. Driving Speed: 500000 pps -> 1250 deg/s
Min. Driving Speed: 1 pps -> 0.0025 deg/s
Acceleration/Deceleration Time: 1 - 1000ms
```

Parameters

- **minimum** (int) – The minimum speed (in number of pulses per second).
- **maximum** (int) – The maximum speed (in number of pulses per second).
- **acceleration** (int) – The acceleration/deceleration time in ms.

Return type
None

set_speed_home(*minimum*, *maximum*, *acceleration*)

Set speed that the stage moves home.

According to the manual:

```
Max. Driving Speed: 500000 pps -> 1250 deg/s
Min. Driving Speed: 1 pps -> 0.0025 deg/s
Acceleration/Deceleration Time: 1 - 1000ms
```

Parameters

- **minimum** (int) – The minimum speed (in number of pulses per second).
- **maximum** (int) – The maximum speed (in number of pulses per second).
- **acceleration** (int) – The acceleration/deceleration time in ms.

Return type
None

`status()`

Get the status of the continuously-variable filter wheel.

Return type
tuple[int, bool]

Returns
The position of the encoder and whether the stage is moving.

`stop_slowly()`

Slowly bring the stage to a stop.

Return type
None

photons.equipment.shutter module

Base class for a shutter.

`class photons.equipment.shutter.Shutter(record, **kwargs)`

Bases: *BaseEquipment*

Base class for a shutter.

Parameters

- **record** (EquipmentRecord) – The equipment record.
- ****kwargs** – Keyword arguments. Can be specified as attributes of an XML element in a configuration file (with the tag of the element equal to the alias of *record*).

`state_changed: SignalInstance`

`is_open()`

Query whether the shutter is open (True) or closed (False).

Return type
bool

`open()`

Open the shutter.

Return type
None

`close()`

Close the shutter.

Return type
None

photons.equipment.shutter_ksc101 module

Communicate with a Thorlabs KSC101 controller to control a shutter.

class photons.equipment.shutter_ksc101.**KSC101Shutter**(*record*, ***kwargs*)

Bases: *Shutter*

Communicate with a Thorlabs KSC101 controller to control a shutter.

Parameters

- **record** (EquipmentRecord) – The equipment record.
- ****kwargs** – Keyword arguments. Can be specified as attributes of an XML element in a configuration file (with the tag of the element equal to the alias of *record*).

connection: KCubeSolenoid

is_open()

Query whether the shutter is open (True) or closed (False).

Return type
bool

open()

Open the shutter.

Return type
None

close()

Close the shutter.

Return type
None

photons.equipment.shutter_s25120a module

Control an electronic shutter controller from Melles Griot.

class photons.equipment.shutter_s25120a.**S25120AShutter**(*record*, ***kwargs*)

Bases: *Shutter*

Control an electronic shutter controller from Melles Griot.

Uses a NI-DAQ board to output a 0 or 5 volt digital signal. The *ConnectionRecord.properties* attribute must contain the NI-DAQ port and line value for the digital output signal (e.g., port=1; line=1)

Parameters

- **record** (EquipmentRecord) – The equipment record.
- ****kwargs** – Keyword arguments. Can be specified as attributes of an XML element in a configuration file (with the tag of the element equal to the alias of *record*).

is_open()

Query whether the shutter is open (True) or closed (False).

Return type
bool

open()

Open the shutter.

Return type
None

close()

Close the shutter.

Return type
None

photons.equipment.sia_cmi module

Switched Integrator Amplifier from CMI.

class photons.equipment.sia_cmi.**SIA3CMI**(*record*, ***kwargs*)

Bases: *BaseEquipment*

Switched Integrator Amplifier from CMI.

Parameters

- **record** (EquipmentRecord) – The equipment record.
- ****kwargs** – Keyword arguments. Can be specified as attributes of an XML element in a configuration file (with the tag of the element equal to the alias of *record*).

connection: SIA3

Integration

alias of IntegrationTime

integration_time_changed: SignalInstance

get_integration_time(*as_enum=False*)

Get the integration time.

Parameters
as_enum (bool) – Whether to return the value as an IntegrationTime enum value or as a numeric value in seconds.

Return type
IntegrationTime | float

Returns
The integration time.

set_integration_time(*time*)

Set the integration time (i.e., the gain).

For example:

```
time=sia.Integration.TIME_100u
time='100u'
time='100us'
time=6
```

are all equivalent.

Parameters
time (int | str) – The integration time.

Return type
None

photons.equipment.thorlabs_flipper module

Thorlabs filter flipper (MFF101 or MFF102).

class photons.equipment.thorlabs_flipper.**ThorlabsFlipper**(*record*, ***kwargs*)

Bases: *KinesisBase*

Thorlabs filter flipper (MFF101 or MFF102).

The optical component that is installed in each position can be passed in as kwargs (e.g., position_1='ND-4.0', position_2='Empty').

Parameters

- **record** (EquipmentRecord) – The equipment record.
- ****kwargs** – Keyword arguments. Can be specified as attributes of an XML element in a configuration file (with the tag of the element equal to the alias of *record*).

connection: FilterFlipper

info()

Returns the information about what is installed in each position.

For example:

```
{1: 'ND-4.0', 2: 'Empty'}
```

Return type
dict[int, str]

get_position()

Returns the current position of the flipper.

The position is either 1 or 2 (but can be 0 during a move).

Return type
int

set_position(*position*, *wait=True*)

Set the flipper to the specified position.

Parameters

- **position** (int) – The position to move to. Must be either 1 or 2.
- **wait** (bool) – Whether to wait for the flipper to finish moving before returning to the calling program.

Return type
None

photons.equipment.thorlabs_stage module

Communicate with a Thorlabs translation/rotation stage.

class photons.equipment.thorlabs_stage.**ThorlabsStage**(*record*, ***kwargs*)

Bases: *KinesisBase*

Communicate with a Thorlabs translation/rotation stage.

The *ConnectionRecord.properties* attribute must contain the "encoder_factor" to convert the encoder position to real-world units.

Parameters

- **record** (EquipmentRecord) – The equipment record.
- ****kwargs** – Keyword arguments. Can be specified as attributes of an XML element in a configuration file (with the tag of the element equal to the alias of *record*).

connection: IntegratedStepperMotors | BenchtopStepperMotor

info()

Returns the information about the stage.

For example:

```
{
  'unit': str, either ' mm' or the unicode value of the degree symbol
  'minimum': float, the minimum position that the stage can be set to
  'maximum': float, the maximum position that the stage can be set to
}
```

Return type
dict[str, float | str]

home(*wait=True*)

Home the stage.

Parameters
wait (bool) – Whether to wait for the stage to finish homing before returning to the calling program.

Return type
None

stop()
Stop moving immediately.

Return type
None

get_encoder()
Returns the value of the encoder.

Return type
int

get_position()
Returns the position of the stage (in mm or degrees).

Return type
float

set_position(*position*, *wait=True*)
Set the position of the stage (in mm or degrees).

Parameters
- **position** (float) – The position, in mm or degrees.
- **wait** (bool) – Whether to wait for the stage to finish moving before returning to the calling program.

Return type
None

to_human(*encoder*, *ndigits=4*)
Convert the encoder value to human units.

Parameters
- **encoder** (int) – The value of the encoder.
- **ndigits** (int) – Round the value to *ndigits* precision (see round() for more details).

Return type
float

Returns
The position (in mm or degrees).

to_encoder(*position*)
Convert the position (in mm or degrees) to an encoder value.

Parameters
position (float) – The position of the stage (in mm or degrees).

Return type
int

Returns
The encoder value.

photons.plugins package

Custom Plugins.

Submodules

photons.plugins.base module

Base class for Plugins.

class photons.plugins.base.**BasePlugin**(*main*, ***kwargs*)

Bases: QWidget

Base class for all Plugins.

Parameters

- **main** (*MainWindow*) – The main window.
- ****kwargs** – All keyword arguments are passed to super().

closing: SignalInstance

after_show()

Override in subclass. Called immediately after show().

Return type
None

closeEvent(*event*)

Overrides QtWidgets.QWidget.closeEvent().

Return type
None

class photons.plugins.base.**PluginInfo**(*cls*, *name*, *description*)

Bases: object

cls: type[*BasePlugin*]

name: str

description: str

photons.plugins.base.**plugin**(*, *name*, *description*)

A decorator to register a Plugin.

Parameters

- **name** (str) – A name to associate with the Plugin.

- **description** (str) – A short description about the Plugin.

photons.plugins.black_screen module

Plugin that is just a widget with a black background.

class photons.plugins.black_screen.**BlackScreen**(*parent*, ***kwargs*)

Bases: *BasePlugin*

Make the desktop screen black.

Parameters

- **parent** (*MainWindow*) – The main window.
- ****kwargs** – All keyword arguments are passed to super().

toggle()

Toggle between full screen and normal display.

Return type
None

mousePressEvent(*event*)

Overrides QtWidgets.QWidget.mousePressEvent().

Return type
None

keyPressEvent(*event*)

Overrides QtWidgets.QWidget.keyPressEvent().

Return type
None

photons.plugins.browse_icons module

Plugin that helps to select an icon.

class photons.plugins.browse_icons.**BrowseIcons**(*parent*, ***kwargs*)

Bases: *BasePlugin*

Find an icon to use.

Parameters

- **parent** (*MainWindow*) – The main window.
- ****kwargs** – All keyword arguments are passed to super().

photons.plugins.spatial_scan module

Perform a 3D spatial scan of a photodetector.

class photons.plugins.spatial_scan.**SpatialScan**(*parent*, ***kwargs*)

Bases: *BasePlugin*

Perform a 3D spatial scan of a photodetector.

Parameters

- **parent** (*MainWindow*) – The main window.
- ****kwargs** – All keyword arguments are passed to super().

closeEvent(*event*)

Overrides QtWidgets.QWidget.closeEvent().

Return type
None

keyReleaseEvent(*event*)

Overrides QtWidgets.QWidget.keyReleaseEvent().

Return type
None

on_worker_abort()

Abort the worker thread.

Return type
None

on_worker_finished()

Called when the worker thread finishes.

Return type
None

on_worker_start()

Start acquiring data in the worker thread.

Return type
None

stop_monitor_and_detector_timer_and_thread()

Stop the timers/threads for the monitor/detector connection.

Return type
None

class photons.plugins.spatial_scan.**SpatialScanWorker**(*parent*)

Bases: Worker

Process an expensive or blocking operation in a thread that is separate from the main thread.

You can access to the attributes of the Worker as though they are attributes of the Thread.

The *args and **kwargs parameters are passed to the constructor of the Worker when the start() method is called.

Example

See `SleepWorker` for an example of a `Worker`.

acquire()

Acquire the monitor and detector samples.

Return type
`tuple`[*Samples*, *Samples*]

Returns
(monitor samples, detector samples)

acquire_dark()

Take a dark measurement.

Return type
`dict`[`str`, `float`]

process()

Run the spatial scan.

Return type
`None`

read_dut()

Read the samples for the device-under-test.

Return type
Samples

photons.plugins.tia_gain module

Plugin to determine the gain of a transimpedance amplifier.

Setup

1. A Keithley 6430 sub-femto SourceMeter produces a current.
2. The current (from the IN/OUT HIGH port of the pre-amp) is the input signal to transimpedance amplifier.
3. The output of the transimpedance amplifier is connected to a digital multimeter.

class photons.plugins.tia_gain.**TIAGain**(*parent*, ***kwargs*)

Bases: *BasePlugin*

Plugin to determine the gain of a transimpedance amplifier.

Parameters

- **parent** (*MainWindow*) – The main window.
- ****kwargs** – All keyword arguments are passed to super().

on_tia_changed(*text*)

Slot for the TIA changed.

Return type
None

on_worker_abort()

Abort the worker thread.

Return type
None

on_worker_start()

Start acquiring data in the worker thread.

Return type
None

on_worker_finished()

Called when the worker thread finishes.

Return type
None

class photons.plugins.tia_gain.**TIAGainWorker**(*parent*)

Bases: Worker

Process an expensive or blocking operation in a thread that is separate from the main thread.

You can access to the attributes of the Worker as though they are attributes of the Thread.

The *args and **kwargs parameters are passed to the constructor of the Worker when the start() method is called.

Example

See SleepWorker for an example of a Worker.

process()

The expensive or blocking operation to process.

Attention: You must override this method.

photons.services package

Custom Services.

Submodules

photons.services.base module

Base class for Services.

photons.services.base.**service**(*, *name=None*, *description=None*)

A decorator to register a Service.

Parameters

- **name** (str) – The name of the registered Service. If not specified then uses the class name. If you specify a name in the decorator then you should also specify the same name in the call to super().
- **description** (str) – A short description about the Service.

class photons.services.base.**ServiceInfo**(*cls*, *name*, *description*)

Bases: object

cls: type[Service]

name: str

description: str

Submodules

photons.app module

Main application entry point and GUI.

class photons.app.**App**(*config=None*)

Bases: QObject

Main application entry point.

Parameters

config (str) – The path to a configuration file. If not specified then uses ~/photons.xml.

added_connection: SignalInstance

removed_connection: SignalInstance

add_lab_logging_metadata(*writer*)

Add the current temperature and humidity of an OMEGA iServer to the writer.

All parameters are read from the configuration file.

Return type

None

property config: Config

The configuration object.

connect_equipment(**args*)

Connect to equipment.

The connection to each equipment is attempted in the following order:

1. If a Link can be established then that gets precedence
2. If a BaseEquipment exists then use it
3. Use EquipmentRecord.connect()

Parameters
***args** (str) – The alias(es) of the EquipmentRecord(s) or the name(s) of Services to Link with to establish the connection.

Return type
Link | *BaseEquipment* | Connection | tuple[Link | *BaseEquipment* | Connection, ...]

Returns
The connection(s).

connect_manager(***kwargs*)

Connect to a Manager.

All keyword arguments are passed to connect().

Return type
None

property connections

The connections to equipment.

create_writer(*prefix*, *, *root=None*, *suffix=None*, *use_timestamp=True*, *zero_padding=3*)

Create a new PhotonWriter to save data to.

The file path has the following structure:

<root>/<year>/<month>/<day>/<prefix>_<suffix | timestamp | run_number>.json

Parameters

- **prefix** (str) – The prefix of the filename.
- **root** (str) – The root directory where the data is saved. If not specified then the value is determined from the <data_root> element in the configuration file.
- **suffix** (str) – If specified, use this value as the suffix. The *use_timestamp* and *zero_padding* parameters are ignored.
- **use_timestamp** (bool) – If True and *suffix* is not specified then use the current time as the suffix.
- **zero_padding** (int) – If *use_timestamp* is False and *suffix* is not specified then use an auto-incremented run number as the suffix. The *zero_padding* value specifies how many leading zeros should be padded to the run number.

Return type
PhotonWriter

Returns
The writer object.

property database: Database

The database object.

disconnect_equipment(**args*)

Disconnect from equipment.

Also handles if the connection to the equipment was established via a Link.

Parameters
***args** (str) – The alias(es) of the EquipmentRecord(s) or the name(s) of Services to disconnect from. If not specified then disconnect from all connections.

Return type
None

disconnect_managers()

Disconnect from all Managers.

Return type
None

property equipment: dict[str, EquipmentRecord]

The equipment record`s that were specified in the configuration file.

link(**names*, *timeout=10*, *strict=True*)

Create links with Services.

You must have first connected to a Manager (see *connect_manager()*).

A Link is established with the first Service that was found with the appropriate name.

Parameters

- **names** (str) – The name(s) of the Service(s) to link with.
- **timeout** (float) – The maximum number of seconds to wait when sending a request to a Manager.
- **strict** (bool) – Whether to raise an error if the Service does not exist.

Return type
Link | tuple[Link, ...]

Returns
The link(s).

property links: dict[str, Link]

The Links that have been made to Services.

property logger

The application logger.

static play_sound(*wav=None*, *wait=True*)

Play a WAV file or theme.

Parameters

- **wav** (str | *Theme*) – The file or *Theme* to play. If not specified then play a random *Theme*.
- **wait** (bool) – Whether to wait for the WAV file to finish playing before returning. Only used if *wav* is a file. Specifying a *Theme* will always wait, since the audio data is stored in memory.

Return type
None

static plot(*data=None*, *block=True*, ***kwargs*)

Show the *Plot* widget.

Parameters

- **data** (str | Root | Dataset | ndarray | list | tuple) – The data to initially plot. If a string then a file path. If not specified then an emtpy *Plot* is returned.
- **block** (bool) – Whether to block until all plots are closed.
- ****kwargs** – If *data* is a filename then all keyword arguments are passed to read(). Otherwise, ignored.

Return type
QApplication

Returns
The application instance.

property prompt

Prompt the user (see msl.qt.prompt).

records(**aliases*, ***kwargs*)

Returns the equipment records.

Parameters

- ***aliases** (str) – The alias(es) of the equipment records that are specified in the configuration file.
- ****kwargs** – Find all equipment records that match the specified search criteria (e.g., manufacturer, model, description). See msl.equipment.database.Database.records() for more details.

Return type
list[EquipmentRecord]

Examples

```
>>> app.records('dmm-3458a', 'shutter', manufacturer='Keithley')
```

run(*show=True*)

Run the main application.

To override the default style, font and palette theme that is used for the QApplication you can create an <app> XML element in the configuration file with the following (optional) attributes:

<app style="windows" font_family="arial" font_size="12" theme="dark"/>

For possible themes, see *MainWindow.create_palette()*.

Parameters
show (bool) – After the application is created, either return immediately or instantiate and show the GUI (which blocks until the GUI is closed).

Return type
QApplication

Returns
The application instance.

send_email(**to*, *subject=None*, *body=None*)

Send an email.

Requires a <smtp> element in the XML configuration file with a <settings> sub-element which is the path to an SMTP configuration file, a <from> sub-element which is the email address of the person who is sending the email and can contain multiple <to> sub-elements for the email addresses that should be emailed. For example,

```
<smtp>
    <settings>path/to/SMTP/setting.txt</settings>
    <from>max.planck</from>
    <!-- The following are optional -->
    <to>neils.bohr</to>
    <to>marie.curie</to>
</smtp>
```

Parameters

- **to** (str) – Who to send the email to. If not specified then uses the <to> elements in the configuration file.
- **subject** (str) – The text to include in the subject field.
- **body** (str) – The text to include in the body of the email. The text can be enclosed in <html></html> tags to use HTML elements to format the message.

Return type
None

set_logging_level(*level*, **names*)

Set the logging level for the specified loggers.

Parameters

- **level** (int | str) – The logging level.
- **names** (str) – The names of the loggers to set to *level*. If none are specified then defaults to the logger for this package.

Return type
None

static sleep(*duration*)

Suspend execution of the calling thread for the given number of seconds.

Parameters
duration (float) – The number of seconds to sleep.

Return type
None

start_equipment_service(*alias*, ***kwargs*)

Start a Service that interfaces with equipment.

This is a blocking call. It is meant to be invoked by the console script.

Parameters

- **alias** (str) – The alias of an EquipmentRecord.
- ****kwargs** – Keyword arguments.

Return type
None

static start_service(*name*, ***kwargs*)

Start a registered Service.

This is a blocking call. It is meant to be invoked by the console script.

Parameters

- **name** (str) – The name of the Service to start.
- ****kwargs** – Keyword arguments.

Return type
None

unlink(**names*)

Unlink from Services.

Parameters
names (str) – The name(s) of the Services to unlink with. If not specified then unlink from all Services.

Return type
None

class photons.app.MainWindow(*app*, ***kwargs*)

Bases: QMainWindow

Main application window.

Parameters

- **app** (*App*) – The application instance.
- ****kwargs** – All keyword argument are passed to super().

hide_progress_bar: SignalInstance

Hide the progress bar.

show_indeterminate_progress_bar: SignalInstance

Show an indeterminate progress bar.

status_bar_message: SignalInstance

The message (text) to display in the status bar.

update_progress_bar: SignalInstance

Update the progress bar with a value in the range [0, 100].

closeEvent(*event*)

Overrides QtWidgets.QWidget.closeEvent().

Return type
None

static create_palette(*name*)

Create and return a QPalette based on a colour theme.

Parameters
name (str) – The name of the theme. Currently only supports "dark".

Return type
QPalette

dragEnterEvent(*event*)

Overrides QtWidgets.QWidget.dragEnterEvent().

Return type
None

dropEvent(*event*)

Overrides QtWidgets.QWidget.dropEvent().

Return type
None

static find_widget(*connection*, *, *parent=None*, ***kwargs*)

Returns the widget that is used for the equipment.

Parameters

- **connection** (Link | *BaseEquipment* | Connection) – The connection to the equipment.
- **parent** (QWidget) – The parent widget to use for the BaseEquipmentWidget.
- ****kwargs** – All additional keyword arguments are passed to super() for the BaseEquipmentWidget.

Raises
RuntimeError – If a widget does not exist for the *connection*.

Return type
BaseEquipmentWidget

on_added_connection(*alias*)

Add a checkmark to a QAction in the Connections QMenu.

Return type
None

on_connections_triggered(*action*)

A QAction in the Connections QMenu was triggered.

Return type
None

on_dock_top_level_changed(*widget*, *is_floating*)

Show the Minimum, Maximum and Close buttons when a docked widget becomes floating.

Return type
None

on_hide_progress_bar()

Hide the progress bar.

Return type
None

on_plugin_closed(*action*, *plugin*)

Called when a Plugin closes.

Return type
None

on_plugins_triggered(*action*)

A QAction in the Plugins QMenu was triggered.

Return type
None

on_removed_connection(*alias*)

Remove a checkmark from a QAction in the Connections QMenu.

Return type
None

on_show_indeterminate_progress_bar()

Show an indeterminate progress bar.

Call this method if a process completion rate is unknown or if it is not necessary to indicate how long the process will take.

Return type
None

on_status_bar_message(*message*)

Display a message in the QStatusBar.

Return type
None

on_update_progress_bar(*percentage*)

Update the value of the progress bar.

Call this method if a process completion rate can be determined. Automatically shows the progress bar if it is hidden.

Parameters
percentage (int | float) – A value in the range [0, 100]. The value gets rounded to the nearest integer.

Return type
None

on_widget_closed(*action*, *widget*)

Called when a docked widget closes.

Return type
None

on_widgets_triggered(*action*)

A QAction in the Widgets QMenu was triggered.

Return type
None

photons.audio module

Play WAV tones.

class photons.audio.**Theme**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

Short theme clips to play as a WAV file.

MARIO = 'mario'

Mario completes a level.

TETRIS = 'tetris'

The Tetris theme.

CONTRA = 'contra'

The Jungle theme from Contra.

class photons.audio.**Song**(*sample_rate=44100*, *tempo=100*)

Bases: object

Create a song.

Parameters

- **sample_rate** (float) – The sample rate, in Hz.
- **tempo** (float) – The tempo (beats per minute). This parameter creates class-instance constants that can be useful when adding notes and rests.

add(*duration*, *left=None*, *right=None*, *volume=0.5*)

Add a rest, note or chord to the song.

Parameters

- **duration** (float) – The number of seconds the note will play for, or, the number of seconds to rest.

- **left** (str | list[str]) – The name(s) of the note(s) to add to the left channel (e.g., 'A4' or ['C4', 'G3', 'E3'] for a chord). Do not specify a value to add a rest.
- **right** (str | list[str]) – The name(s) of the note(s) to add to the right channel (e.g., 'C3'). If not specified then equals the left channel.
- **volume** (float) – The amplitude of the sine wave. Should be between 0 and 1.

Return type
None

play()

Play the song.

Return type
None

photons.audio.**play**(*wav*, *wait=True*)

Play a WAV file or theme.

Parameters

- **wav** (str | *Theme*) – A file or theme to play.
- **wait** (bool) – Whether to wait for the WAV file to finish playing before returning. Only used if *wav* is a file. Specifying a *Theme* will always wait, since the audio data is stored in memory.

Return type
None

photons.audio.**random**()

Play a random *Theme*.

Return type
None

photons.io module

Custom file writer.

class photons.io.**PhotonWriter**(*file*, *, *log_size=None*)

Bases: JSONWriter

A custom file writer.

Parameters

- **file** (str) – The path of the file to create.
- **log_size** (int) – The initial size of DatasetLogging. If None or 0 then do not include log messages in the writer.

add_equipment(**records*)

Add equipment records to an 'equipment' Group.

Automatically creates the 'equipment' Group if it does not already exist.

Return type
None

append(**data*, *name=None*)

Append data to a dataset.

Parameters

- **data** – The data to append. The timestamp when this method is called is automatically added to the dataset.
- **name** (str) – The name of the dataset to append the data to. If not specified then appends to the latest dataset that was initialized.

Return type
None

data(*name=None*)

Return the current data of a dataset.

Parameters
name (str) – The name of the dataset. If not specified then uses the name of the latest dataset that was initialized.

Return type
ndarray

initialize(**header*, *microseconds=False*, *name='dataset'*, *size=100*, *types=None*, ***metadata*)

Initialize a dataset.

Parameters

- ***header** (str) – The names of the header fields. The first field name is `timestamp` and it is automatically created.
- **microseconds** (bool) – Whether to include microseconds in the `timestamp`.
- **name** (str) – The name of the dataset to initialize. Can contain `/` to specify a subgroup (relative to the root Group).
- **size** (int) – The initial size of the dataset. The dataset will automatically increase in size when it needs to.
- **types** (list[type]) – The data types of each header field. If not specified then uses *float* for each field.
- ****metadata** – The metadata to associate with the dataset.

Return type
None

meta(*name=None*)

Return the current metadata of a dataset.

Parameters
name (str) – The name of the dataset. If not specified then uses the name of the latest dataset that was initialized.

Return type
dict

static now_iso(*ignore_microsecond=True*)

Get the current time in ISO-8601 format.

Parameters
ignore_microsecond (bool) – Whether to ignore the microsecond part.

Return type
str

update_metadata(*name=None*, ***metadata*)

Update the metadata for a dataset.

Parameters

- **name** (str) – The name of the dataset to append the data to. If not specified then appends to the latest dataset that was initialized.
- ****metadata** – The metadata to associate with the dataset.

Return type
None

write(***kwargs*)

Overrides the write() method.

Return type
None

photons.log module

Logging configuration.

photons.log.**env_level**(*key='PHOTONS_LOG_LEVEL'*)

Read the logging level from an environment variable.

Return type
int

photons.log.**set_block**(**names*)

Block all messages from the specified loggers.

Return type
None

photons.log.**set_debug**(**names*)

Show DEBUG (and above) messages from the specified loggers.

Return type
None

photons.log.**set_errors**(**names*)

Show ERROR (and above) messages from the specified loggers.

Return type
None

photons.log.**set_info**(**names*)

Show INFO (and above) messages from the specified loggers.

Return type
None

photons.log.**set_level**(*name*, *level*)

Set the logging level for a particular logger.

Return type
None

photons.log.**set_warnings**(**names*)

Show WARNING (and above) messages from the specified loggers.

Return type
None

photons.network module

QDialogs for starting a Network Manager or a Service, or, connecting to a Manager as a Client.

class photons.network.**ClientServiceDialog**(*parent*, *widget*)

Bases: QDialog

Connect to a Manager as either a Client or as a Service.

Parameters

- **parent** (*MainWindow*) – The parent widget.
- **widget** (QWidget) – The widget to add to the first row of the QFormLayout.

check_hostname()

Check the hostname of the Manager.

Return type
bool

on_connect_clicked()

Connect to a Manager.

Return type
None

prompt_authentication()

Prompt the user for the authentication credentials (if required).

Return type
tuple[Optional[str], Optional[str], Optional[str]]

Returns
The username, the password for username, the password for the Manager.

class photons.network.**CreateClient**(*parent*)

Bases: *ClientServiceDialog*

Connect to a Manager as a Client.

Parameters
parent (*MainWindow*) – The parent widget.

on_connect_clicked()

Connect to a Manager.

Return type
None

class photons.network.**StartEquipmentService**(*parent*)

Bases: *ClientServiceDialog*

Start a Service that interfaces with equipment.

The Service will be running on *localhost*, but the Manager can be running on a remote computer.

Parameters
parent (*MainWindow*) – The parent widget.

on_connect_clicked()

Connect to a Manager.

Return type
None

class photons.network.**StartManager**(*parent*)

Bases: QDialog

Start a Network Manager on *localhost*.

Parameters
parent (*MainWindow*) – The parent widget.

on_start_clicked()

Start a Manager.

Return type
None

class photons.network.**StartService**(*parent*)

Bases: *ClientServiceDialog*

Start a Service.

The Service will be running on *localhost*, but the Manager can be running on a remote computer.

Parameters
parent (*MainWindow*) – The parent widget.

on_connect_clicked()

Connect to a Manager.

Return type
None

photons.plotting module

Plot widget.

class photons.plotting.**BaseTable**(*rows=0*, *header=None*, *parent=None*, *tooltip=None*)

Bases: QTableWidget

Custom table that has text-selectable, read-only cells.

resize_row_column(*factor=1.5*)

Resize the width of the row-indicator column (the first column).

Return type
None

class photons.plotting.**BaseTableDelegate**(*parent*)

Bases: QItemDelegate

Allows for a table cell to be selectable and read only.

createEditor(*parent*, *option*, *index*)

Overrides QtWidgets.QAbstractItemDelegate.createEditor().

Return type
QLineEdit

class photons.plotting.**LogTable**(*log_dataset*, *title*)

Bases: *BaseTable*

Display the logging records in a table.

COLORS: dict[str, QColor] = {'CRITICAL': PySide6.QtGui.QColor.fromRgbF(1.000000, 0.000000, 0.000000, 1.000000), 'DEBUG': PySide6.QtGui.QColor.fromRgbF(0.466667, 0.533333, 0.600000, 1.000000), 'ERROR': PySide6.QtGui.QColor.fromRgbF(0.545098, 0.000000, 0.000000, 1.000000), 'INFO': PySide6.QtGui.QColor.fromRgbF(0.000000, 0.000000, 0.000000, 1.000000), 'NOTSET': PySide6.QtGui.QColor.fromRgbF(1.000000, 1.000000, 1.000000, 1.000000), 'WARNING': PySide6.QtGui.QColor.fromRgbF(0.721569, 0.525490, 0.043137, 1.000000)}

class photons.plotting.**MetadataTable**(*tooltip=None*)

Bases: *BaseTable*

Display metadata in a table.

add(*metadata*)

Add a row to the table.

Return type
None

class photons.plotting.**Plot**(*root=None*, *parent=None*, ***kwargs*)

Bases: QWidget

A widget for plotting 2D data and displaying metadata.

Parameters

- **root** (Root) – A Root object.
- **parent** (QWidget) – The parent widget.
- ****kwargs** – All keyword arguments are passed to super().

dragEnterEvent(*event*)

Overrides QtWidgets.QWidget.dragEnterEvent().

Return type
None

dropEvent(*event*)

Overrides QtWidgets.QWidget.dropEvent().

Return type
None

new_root()

A new Root has been dropped.

Return type
None

on_clear_plot()

Clear the scatter plot.

Return type
None

on_dataset_changed(*name*)

A new dataset was selected.

Return type
None

on_replot()

Replot the currently-selected data.

Return type
None

on_screenshot()

Save a screenshot of the Plot widget.

Return type
None

should_clear_plot()

Whether the plots should be cleared.

Return type
bool

class photons.plotting.**ScatterPlot**(*parent*)

Bases: QWidget

A scatter-plot for a 2D dataset.

clear()

Clear all plots.

Return type
None

static hovering(*x_text*)

Callback function for a mouse hover.

Return type
Callable[[float, float, Optional[float]], str]

on_y_range_checkbox_clicked(*state*)

Enable or disable the min-max widgets for the Y axis.

Return type
None

redraw(*ignore=None*)

Redraw the plot.

Return type
None

set_dataset(*dataset*)

A different dataset was selected in the combobox.

Return type
None

class photons.plotting.**RealTimePlot**(*, *error_options=None*, *plot_options=None*, *signaler=None*, *size=10000*, *title=None*)

Bases: QWidget

Plot data in real time.

Parameters

- **error_options** (dict) – Options passed to ErrorBarItem. If not specified, default options are used.
- **plot_options** (dict) – Options passed to PlotItem. If not specified, default options are used.
- **signaler** (SignalInstance) – The Qt signal that emits *Samples*.
- **size** (int) – The maximum number of data points that can be shown. When the number of data points exceeds this value the latest data point is shown and the oldest data point is removed.
- **title** (str) – The text to display in the titlebar of the widget.

closing: SignalInstance

Emitted when the widget closes.

clear()

Clear the plot.

Return type
None

update(*samples*)

Update the plot.

If a *signaler* is specified when this class is instantiated, this method *(slot)* is called automatically when the *signaler* emits the *samples*.

Parameters
samples (*Samples*) – The data to add to the plot. The standard deviation of the mean is used as the error bar.

Return type
None

closeEvent(*event*)

Overrides QtWidgets.QWidget.closeEvent().

Disconnect from the *signaler* (if one was specified), removes items from the plot and emits the *closing* signal.

Return type
None

photons.samples module

A formatting-friendly convenience class for 1-D sample data.

photons.samples.**order_of_magnitude**(*value*)

Returns the order of magnitude of *value*.

Return type
int

photons.samples.**parse**(*format_spec*)

Parse a format specification into its grammar fields.

Return type
dict[str, str]

photons.samples.**si_prefix_factor**(*exponent*)

Returns the SI prefix and scaling factor.

Parameters
exponent (int) – The exponent, e.g., 10 ** exponent

Return type
tuple[str, float]

class photons.samples.**Rounded**(*value*, *precision*, *type*, *exponent*, *suffix*)

Bases: object

Represents a rounded value.

value: float

precision: int

type: str

exponent: int

suffix: str

class photons.samples.**Format**(**kwargs*)

Bases: object

Format specification.

result(*text*)

Format *text* using the fill, align, zero and width fields.

Return type
str

uncertainty(*uncertainty*, *, *hash=None*, *type='f'*, *precision=None*)

Format *uncertainty* using the hash, grouping, precision and type fields.

Parameters

- **uncertainty** (float) – The uncertainty to format.
- **hash** (str) – Can be either # or '' (an empty string)
- **type** (Optional[str]) – Can be one of: e, E, f, F, g, G, n
- **precision** (int) – Indicates how many digits should be displayed after the decimal point for presentation types f and F, or before and after the decimal point for presentation types g or G.

Return type
str

Returns
The *uncertainty* formatted.

update(*std*)

Update the *precision* and *u_exponent* attributes.

Parameters
std (float) – The standard uncertainty of the samples.

Return type
None

value(*value*, *, *hash=None*, *type=None*, *sign=None*, *precision=None*)

Format *value* using the sign, hash, grouping, precision and type fields.

Parameters

- **value** (float) – The value to format.
- **hash** (str) – Can be either # or '' (an empty string)
- **type** (str) – Can be one of: e, E, f, F, g, G, n
- **sign** (str) – Can be one of: +, -, ' ' (a space)
- **precision** (int) – Indicates how many digits should be displayed after the decimal point for presentation types f and F, or before and after the decimal point for presentation types g or G.

Return type
str

Returns
The *value* formatted.

class photons.samples.**Samples**(*samples=None*, *, *mean=None*, *stdev=None*, *size=None*, *overload=1e+30*)

Bases: object

Convenience class for a 1-D array of data samples.

Calculates the mean, standard deviation, variance, relative standard deviation and standard deviation of the mean of the samples.

Parameters

- **samples** (Union[str, Sequence[str | int | float], ndarray]) – The samples. If a string then in CSV format.
- **mean** (float) – If specified, then it is not calculated from the *samples*.
- **stdev** (float) – If specified, then it is not calculated from the *samples*.
- **size** (int) – If specified, then it is not determined from the *samples*.
- **overload** (Optional[float]) – For some devices, like a DMM, if the input signal is greater than the present range can measure, the device returns a large value (e.g., 9.9E+37) to indicate a measurement overload. If the absolute value of the mean is greater than *overload* then the mean and standard deviation become NaN. Setting *overload* to None disables this check.

__getattr__(*item*)

Pass all other attributes to the ndarray.

property mean: float

Returns the mean.

property overload: float | None

Returns the overload value.

property relative_stdev: float

Returns the relative standard deviation.

property relative_stdom: float

Returns the relative standard deviation of the mean.

property samples: ndarray

Returns the samples.

property size: int

Returns the number of samples.

property stdev: float

Returns the sample standard deviation.

property stdom: float

Returns the standard deviation of the mean.

to_json()

Allows for this class to be JSON serializable with msl-network.

Return type
dict[str, float]

to_ureal(*, *label=None*, *delta=None*, *truncated=False*)

Convert to an uncertain-real number.

Parameters

- **label** (str) – The label to associate with the uncertain number.
- **delta** (float) – The digitization step size (only valid if the samples are digitized).
- **truncated** (bool) – Whether the digitized samples were truncated or rounded. Only used if *delta* is not None.

Return type
UncertainReal

Returns
The samples as an uncertain-real number.

property variance: float

Returns the sample variance.

photons.utils module

General classes and functions.

photons.utils.**array_central**(*centre*, *width*, *step*, *, *randomize=False*, *decimals=None*)

Get an array about a central value, for a given full width and step size.

Parameters

- **centre** (float) – The central value in the array.
- **width** (float) – The full width about *centre*. Must be an integer multiple of *step*.
- **step** (float) – The step size. Must be a positive number.
- **randomize** (bool) – Whether to randomize the values in the array.
- **decimals** (int) – The number of decimals to use in numpy.around(). If not specified then uses the value of *step* to determine the number of decimals.

Return type
ndarray

Returns
The requested array.

photons.utils.**array_evenly**(*start*, *stop*, *step*, *, *randomize=False*, *decimals=None*)

Return evenly-spaced values within a given interval.

The values are generated within the interval [start, stop].

Parameters

- **start** (float) – Start value (the interval includes this value).
- **stop** (float) – Stop value (the interval includes this value).
- **step** (float) – The step size.
- **randomize** (bool) – Whether to randomize the values in the array.
- **decimals** (int) – The number of decimals to use in numpy.around(). If not specified then uses the value of *step* to determine the number of decimals.

Return type
ndarray

Returns
The requested array.

photons.utils.**array_merge**(**vectors*)

Merge 1-D arrays, field by field, to create a new structured N-D array.

Return type
ndarray

photons.utils.**array_photodiode_centre**(*centre*, ***, *width=10*, *step=0.1*, *randomize=False*, *decimals=None*)

Useful when trying to find the centre position of a photodiode.

The returned array has values at the two edges of the photodiode and in the central region of the photodiode (in one-dimension only).

Parameters

- **centre** (float) – The position of a translation stage where the centre of the photodiode is believed to be.
- **width** (float) – The width of the photodiode or the diameter of the aperture. Must have the same unit as *step*.
- **step** (float) – The step size to move the translation stage within each of the three regions (not between regions).
- **randomize** (bool) – Whether to randomize the values in the array.
- **decimals** (int) – The number of decimals to use in numpy.around(). If not specified then uses the values of *centre* and *step* to determine the number of decimals.

Return type
ndarray

Returns
The requested array.

photons.utils.**ave_std**(*data*, ***, *axis=None*)

Calculate the average and standard deviation.

Parameters

- **data** (ndarray) – The values to compute the average and standard deviation of.

- **axis** (int | tuple[int]) – Axis or axes along which the average and standard deviation is computed. The default is to compute the values of the flattened array.

Return type
tuple[float | ndarray, float | ndarray]

Returns
The average value and the standard deviation.

photons.utils.**get_decimals**(*value*)

Get the number of digits after the decimal point.

This function returns a sensible result only if *value* was explicitly defined for a parameter (for example a value from a QSpinbox). If *value* is the result from a calculation then there will be floating-point issues, and will most likely return nonsense (e.g., a number > 15).

Return type
int

photons.utils.**lab_logging**(*root_url*, **aliases*, *corrected=True*, *strict=True*, *timeout=10*)

Read the current temperature, humidity and dewpoint of (an) OMEGA iServer(s).

Parameters

- **root_url** (str) – The root url of the webapp, e.g., 'http://hostname:port/'
- **aliases** (str) – The iServer alias(es) to retrieve the data from. If not specified then retrieves the data from all iServers.
- **corrected** (bool) – Whether to return corrected (True) or uncorrected (False) values.
- **strict** (bool) – Whether to raise an exception if the connection to the webapp cannot be established.
- **timeout** (float) – The maximum number of seconds to wait for a reply from the webapp.

Return type
dict

Returns
The temperature, humidity and dewpoint from the iServer(s).

photons.utils.**std_relative**(*array*, ***, *axis=None*)

Calculate the relative standard deviation.

Parameters

- **array** (ndarray) – The values to compute the relative standard deviation of.
- **axis** (int | tuple[int]) – Axis or axes along which the relative standard deviation is computed. The default is to compute the value of the flattened array.

Return type
float | ndarray

Returns
The relative standard deviation.

photons.utils.**hhmmss**(*seconds*)

Convert seconds to a hh:mm:ss representation.

Return type
str

photons.utils.**mean_max_n**(*array*, *n*)

Return the mean of the maximum *n* values in *array*.

Return type
float

photons.utils.**mean_min_n**(*array*, *n*)

Return the mean of the minimum *n* values in *array*.

Return type
float

1.2 License

1.3 Developers

- Joseph Borbely <joseph.borbely@measurement.govt.nz>

1.4 Changelog

1.4.1 Version 0.1.0 (in development)

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

Symbols

A

E

F

G

H

I

N

P

R

S

T

Z