labscript-utils

Release 3.1.0.dev80+g4581953

labscript suite contributors

Nov 02, 2020

DOCUMENTATION

labscript-utils contains code shared between multiple programs in the labscript suite. This documentation is primarily for developers who might want to use this common code in their own custom features.

CHAPTER

ONE

API REFERENCE

1.1 Labscript Tools

1.1.1 labscript_utils

labscript_utils.**dedent**(*s*)
Remove leading spaces from the first line of a string, all common leading indentation (spaces only) from subsequent lines, strip trailing spaces from all lines and replace single newlines prior to lines with the common indentation with spaces. Lines with additional indentation are kept verbatim. Good for unwrapping error messages etc that are in code as multiline triple-quoted strings.

labscript_utils.**import_or_reload**(*modulename*)
Behaves like 'import modulename' would, excepts forces the imported script to be rerun

1.1.2 labscript_utils.connections

class labscript_utils.connections.**Connection**(*raw_row*)
A class to represent a row in the connection table, present the contents as instance attributes after deserialising their contents, and providing default values for backward compatibility with older HDF5 files. Contains links to Connection objects for child devices of each device

_defaults = {'BLACS_connection': '', 'properties': {}, 'unit conversion class': No

_deserialise(*name*, *value*)
deserialise one item of the row depending on what it is

_populate_relatives(*table*)
Populate child devices based on a list of other connection objects, and set self.parent to our parent device.

compare_to(*other_connection*)

diff(*other*)

find_by_name(*name*)

find_child(*parent_name*, *parent_port*)

print_details(*indent*)

property properties

property unit_conversion_params

class labscript_utils.connections.**ConnectionTable**(*h5file*, *logging_prefix=None*, *exceptions_in_thread=False*)

assert_superset(*other*)

compare_to(*other*)

find_by_name(*name*)

find_child(*parent_name*, *parent_port*)

get_attached_devices()
Finds out which devices in the connection table are connected to BLACS, based on whether their 'BLACS_connection' attribute is non-empty. Returns a dictionary of them in the form {device_instance_name: labscript_class_name}

print_details()

remove_device(*device_name*)
Removes a device from the ConnectionTable, but keeps it in the raw_table. This can help make comparissons of connection tables fail for tables with broken devices.

labscript_utils.connections.**_ensure_str**(*s*)
convert bytestrings and numpy strings to python strings

1.1.3 labscript_utils.dict_diff

labscript_utils.dict_diff.**dict_diff**(*dict1*, *dict2*)
Return the difference between two dictionaries as a dictionary of key: [val1, val2] pairs. Keys unique to either dictionary are included as key: [val1, '-'] or key: ['-', val2].

1.1.4 labscript_utils.labconfig

class labscript_utils.labconfig.**EnvInterpolation**
Interpolation which expands environment variables in values, by post-filtering BasicInterpolation.before_get()

before_get(**args*)

class labscript_utils.labconfig.**LabConfig**(*config_path=PosixPath('/home/docs/labscript-suite/labconfig/build-12240137-project-608578-philipstarkey-labscript-utils.ini')*, *required_params=None*, *defaults=None*)

exception NoOptionError(*option*, *section*)
A requested option was not found.

exception NoSectionError(*section*)
Raised when no section matches a requested option.

_abc_impl = <_abc_data object>

labscript_utils.labconfig.**load_appconfig**(*filename*)
Load an .ini file and return a dictionary of its contents. All values will be converted to Python objects with ast.literal_eval(). All keys will be lowercase regardless of the written contents on the .ini file.

labscript_utils.labconfig.**save_appconfig**(*filename*, *data*)
Save a dictionary as an ini file. The keys of the dictionary comprise the section names, and the values must themselves be dictionaries for the names and values within each section. All section values will be converted to strings with pprint.pformat().

1.1.5 labscript_utils.settings

class labscript_utils.settings.**Settings**(*storage='hdf5'*, *file=None*, *parent=None*, *page_classes=[]*)

add_settings_interface(*setting_class*)

close(**args*, ***kwargs*)

create_dialog(*goto_page=None*)

get_value(*settings_class*, *value_name*)

load(*name*)

on_cancel(**args*, ***kwargs*)

on_save(**args*, ***kwargs*)

register_callback(*callback*)

remove_callback(*callback*)

1.1.6 labscript_utils.testing_utils

class labscript_utils.testing_utils.**Any**(*types=<class 'object'>*)
A class whose instances equal any object of the given type or tuple of types. For use with mock.Mock.assert_called_with when you don't care what some of the arguments are

class labscript_utils.testing_utils.**ThreadTestCase**(**args*, ***kwargs*)
Test case that runs tests in a new thread whilst providing a mainloop that allows running scripts in the current thread. Those scripts can then be tested from the testing thread.

_mainloop()

_run(**args*, ***kwargs*)
Called in a thread to run the tests

quit_mainloop()

run(**args*, ***kwargs*)

run_script_as_main(*filepath*)

static wait_for(*condition_func*, *timeout=5*, *initial_poll_interval=0.005*, *max_poll_interval=0.5*)
Busy wait for a condition to be true. Uses exponential backoff so it's fast when things are fast and not a complete hog when they're not

class labscript_utils.testing_utils.**dotdict**
dot.notation access to dictionary attributes

class labscript_utils.testing_utils.**monkeypatch**(*obj*, *name*, *mocked_attr*)
Context manager to temporarily monkeypatch an object attribute with some mocked attribute

1.1.7 labscript_utils.properties

labscript_utils.properties.**_check_dicts**(*o*)

labscript_utils.properties.**_decode_bytestrings**(*o*)
Decode all base64-encoded values (not keys) to bytestrings

labscript_utils.properties.**_default**(*o*)

labscript_utils.properties.**_encode_bytestrings**(*o*)
Encode all bytestring values (not keys) to base64 with a prefix

labscript_utils.properties.**_get_con_table_properties**(*h5_file*, *device_name*)

labscript_utils.properties.**_get_device_properties**(*h5_file*, *device_name*)

labscript_utils.properties.**_get_unit_conversion_parameters**(*h5_file*, *device_name*)

labscript_utils.properties.**deserialise**(*value*)

labscript_utils.properties.**get**(*h5_file*, *device_name*, *location*)

labscript_utils.properties.**get_attribute**(*group*, *name*)
Return the attribute of the given name from the given HDF5 group, deserialising it if it has been encoded as JSON

labscript_utils.properties.**get_attributes**(*group*)
Return attributes of a HDF5 group as a dict, deserialising any that have been encoded as JSON

labscript_utils.properties.**is_json**(*value*)

labscript_utils.properties.**serialise**(*value*)

labscript_utils.properties.**set_attributes**(*group*, *attributes*)
Add attributes to a HDF5 group, serialising them to JSON if they do not map to native HDF5 datatypes

labscript_utils.properties.**set_device_properties**(*h5_file*, *device_name*, *properties*)

1.1.8 labscript_utils.unitconversions

labscript_utils.unitconversions.**get_unit_conversion_class**(*fullname*)
import and return the unit conversion class with the given name. Ideally this is a fully qualified class name with an absolute import path, i.e. path.to.some.module.ClassName. But if it is just a single name, we fall back to looking through all classes defined in submodules. This allows backward compatibility with old shot files that do not have the full name saved.

Basic Unit Conversion Classes

class labscript_utils.unitconversions.test.**test**(*calibration_parameters=None*)
Bases: *labscript_utils.unitconversions.UnitConversionBase.UnitConversion*

A_from_base(*volts*)

A_to_base(*amps*)

Gauss_from_base(*volts*)

Gauss_to_base(*gauss*)

base_unit = 'MHz'

derived_units = ['A', 'Gauss']

class labscript_utils.unitconversions.example.**example1**(*calibration_parameters=None*)
Bases: *labscript_utils.unitconversions.UnitConversionBase.UnitConversion*

A_from_base(*volts*)

A_to_base(*amps*)

Gauss_from_base(*volts*)

Gauss_to_base(*gauss*)

base_unit = 'V'

class labscript_utils.unitconversions.example.**example2**(*calibration_parameters=None*)
Bases: *labscript_utils.unitconversions.UnitConversionBase.UnitConversion*

base_unit = 'MHz'

detuned_MHz_from_base(*mhz*)

detuned_MHz_to_base(*d_mhz*)

class labscript_utils.unitconversions.example.**example3**(*calibration_parameters=None*)
Bases: *labscript_utils.unitconversions.UnitConversionBase.UnitConversion*

W_from_base(*vpp*)

W_to_base(*watts*)

base_unit = 'Vpp'

class labscript_utils.unitconversions.UnitConversionBase.**UnitConversion**(*params*)
Bases: object

unit_list = {'G': 1000000000.0, 'M': 1000000.0, 'T': 1000000000000.0, 'k': 1000.0, 'm

labscript_utils.unitconversions.UnitConversionBase.**vectorise**(*method*)

Unit Conversion Classes

class labscript_utils.unitconversions.aom.**SineAom**(*calibration_parameters=None*)
Bases: *labscript_utils.unitconversions.NovaTechDDS9m.NovaTechDDS9mAmpConversion*

AOM calibration P(A) is very close to a sine for dipole trap AOM!

Power_from_base(*amp*)

Power_to_base(*power*)

base_unit = 'Arb'

fraction_from_base(*amp*)

fraction_to_base(*fraction*)

class labscript_utils.unitconversions.detuning.**detuning**(*calibration_parameters=None*)
Bases: *labscript_utils.unitconversions.UnitConversionBase.UnitConversion*

MHz_from_base(*aom_frequency*)

MHz_to_base(*aom_frequency_MHz*)

base_unit = 'Hz'

d_MHz_from_base(*aom_frequency*)

d_MHz_to_base(*detuning_MHz*)

derived_units = ['MHz', 'd_MHz', 'linewidths']

linewidths_from_base(*aom_frequency*)

linewidths_to_base(*linewidths*)

class labscript_utils.unitconversions.linear_coil_driver.**BidirectionalCoilDriver**(*calibration_pa*
Bases: *labscript_utils.unitconversions.UnitConversionBase.UnitConversion*

A_from_base(*volts*)

A_to_base(*amps*)

base_unit = 'V'

derived_units = ['A']

class labscript_utils.unitconversions.linear_coil_driver.**UnidirectionalCoilDriver**(*calibration_*
Bases: *labscript_utils.unitconversions.linear_coil_driver.BidirectionalCoilDriver*

A_from_base(*volts*)

A_to_base(*amps*)

class labscript_utils.unitconversions.NovaTechDDS9m.**NovaTechDDS9mAmpConversion**(*calibration_para*
Bases: *labscript_utils.unitconversions.UnitConversionBase.UnitConversion*

base_unit = 'Arb'

hardware_from_base(*arb*)

hardware_to_base(*hardware*)

class labscript_utils.unitconversions.NovaTechDDS9m.**NovaTechDDS9mFreqConversion**(*calibration_par*
Bases: *labscript_utils.unitconversions.UnitConversionBase.UnitConversion*

MHz_from_base(*Hz*)

MHz_to_base(*MHz*)

base_unit = 'Hz'

class labscript_utils.unitconversions.optotunelens.**OptotuneLens**(*calibration_parameters=None*)
Bases: *labscript_utils.unitconversions.UnitConversionBase.UnitConversion*

I_from_base(*volts*)

I_to_base(*current*)

base_unit = 'V'

derived_units = ['distance', 'I']

distance_from_base(*volts*)

distance_to_base(*percentage*)

class labscript_utils.unitconversions.quad_driver.**quad_driver**(*calibration_parameters={'A_min': -0.09, 'A_offset': -0.642724, 'A_per_V': 19.9757, 'Gcm_per_A': 1.88679}*)
Bases: *labscript_utils.unitconversions.UnitConversionBase.UnitConversion*

A_from_base(*volts*)

A_to_base(*arg*)

Gcm_from_base(*volts*)

Gcm_to_base(*gauss_per_cm*)

base_unit = 'V'

derived_units = ['A', 'Gcm']

class labscript_utils.unitconversions.quad_monitor.**quad_monitor**(*calibration_parameters={'A_offset': -0.043200000000000016, 'A_per_V': 20.032, 'Gcm_per_A': 1.88679}*)

Bases: *labscript_utils.unitconversions.UnitConversionBase.UnitConversion*

A_from_base(*volts*)

A_to_base(*amps*)

Gcm_from_base(*volts*)

Gcm_to_base(*gauss_per_cm*)

base_unit = 'V'

derived_units = ['A', 'Gcm']

1.2 Communications

1.2.1 labscript_utils.h5_lock

class labscript_utils.h5_lock.**File**(*name*, *mode=None*, *driver=None*, *libver=None*, ***kwds*)

_abc_impl = <_abc_data object>

close()

Close the file. All open objects become invalid

labscript_utils.h5_lock.**hack_locks_onto_h5py**()

1.2.2 labscript_utils.ls_zprocess

class labscript_utils.ls_zprocess.**Context**(*io_threads=1*, *shared_secret=None*)

Subclass of zprocess.security.SecureContext configured with settings from labconfig, substitutable for a zmq.Context. Can be instantiated to get a unique context, or call the .instance() classmethod to possibly get an already-existing one. Only use the latter if you do not indent to terminate the context.

classmethod instance()

Returns a shared instance with the same shared secret, if there is one, otherwise creates it. If an instance already exists, io_threads will be ignored, otherwise it will be used in the new instance. Takes into account subclasses such that a subclass calling this method will always get back an instance of its own class

socket(**args*, ***kwargs*)
Create a Socket associated with this Context.

Parameters

- **socket_type** (*int*) – The socket type, which can be any of the 0MQ socket types: REQ, REP, PUB, SUB, PAIR, DEALER, ROUTER, PULL, PUSH, etc.
- **kwargs** – will be passed to the __init__ method of the socket class.

labscript_utils.ls_zprocess.**Event**(**args*, ***kwargs*)

labscript_utils.ls_zprocess.**Handler**(**args*, ***kwargs*)

labscript_utils.ls_zprocess.**Lock**(**args*, ***kwargs*)

class labscript_utils.ls_zprocess.**ProcessTree**(*shared_secret=None*, *allow_insecure=False*, *zlock_host=None*, *zlock_port=7339*, *zlog_host=None*, *zlog_port=7340*)
A singleton zprocess.ProcessTree configured with settings from labconfig for security, zlock and zlog. Being a singleton is not enforced - the class can still be instantiated as normal - but calling the .instance() classmethod will give the singleton.

_instance = <labscript_utils.ls_zprocess.ProcessTree object>

classmethod instance()

labscript_utils.ls_zprocess.**RemoteProcessClient**(*host*, *port=None*)

class labscript_utils.ls_zprocess.**ZMQClient**
A singleton zprocess.ZMQClient configured with settings from labconfig for security. Being a singleton is not enforced - the class can still be instantiated as normal - but calling the .instance() classmethod will give the singleton.

_instance = None

classmethod instance()

class labscript_utils.ls_zprocess.**ZMQServer**(*port=None*, *dtype='pyobj'*, *pull_only=False*, *bind_address='tcp://0.0.0.0'*, *timeout_interval=None*, ***kwargs*)
A ZMQServer configured with security settings from labconfig

labscript_utils.ls_zprocess.**connect_to_zlock_server**()

labscript_utils.ls_zprocess.**ensure_connected_to_zlog**()
Ensure we are connected to a zlog server. If one is not running and we are the top-level process, start a zlog server configured according to LabConfig.

labscript_utils.ls_zprocess.**get_config**()
Get relevant options from LabConfig, substituting defaults where appropriate and return as a dict

labscript_utils.ls_zprocess.**zmq_get**(**args*, ***kwargs*)

labscript_utils.ls_zprocess.**zmq_get_multipart**(**args*, ***kwargs*)

labscript_utils.ls_zprocess.**zmq_get_raw**(**args*, ***kwargs*)

labscript_utils.ls_zprocess.**zmq_get_string**(**args*, ***kwargs*)

labscript_utils.ls_zprocess.**zmq_push**(**args*, ***kwargs*)

labscript_utils.ls_zprocess.**zmq_push_multipart**(**args*, ***kwargs*)

labscript_utils.ls_zprocess.**zmq_push_raw**(**args*, ***kwargs*)

labscript_utils.ls_zprocess.**zmq_push_string**(**args*, ***kwargs*)

1.2.3 labscript_utils.remote

Script to run a zprocess.remote server configured according to LabConfig. Run with:

```
python -m labscript_utils.remote [--daemon] [--no-tui]
```

If –daemon is specified, the server will be started in the background. If –no-tui is specified, the server will run with ordinary terminal output instead of with the interactive text-based user interface (TUI).

labscript_utils.remote.**main**()

1.2.4 labscript_utils.shared_drive

labscript_utils.shared_drive.**path_to_agnostic**(*path*)

labscript_utils.shared_drive.**path_to_local**(*path*)

1.2.5 labscript_utils.zlock

Script to run a zlock server configured according to LabConfig. Run with:

```
python -m labscript_utils.zlock [--daemon]
```

If –daemon is specified, the zlock server will be started in the background.

labscript_utils.zlock.**main**()

1.2.6 labscript_utils.zlog

Script to run a zlog server configured according to LabConfig. Run with:

```
python -m labscript_utils.zlog [--daemon]
```

If –daemon is specified, the zlog server will be started in the background.

labscript_utils.zlog.**main**()

1.3 GUI

1.3.1 labscript_utils.splash

class labscript_utils.splash.**Splash**(*imagepath*)

BG = '#ffffff'

alpha = 0.875

h = 230

icon_frac = 0.65

imheight = 150

imwidth = 150

paintEvent(*self*, *QPaintEvent*)

show(*self*)

update_text(*text*)

w = 250

1.3.2 labscript_utils.qtwidgets

Helper Widgets

class labscript_utils.qtwidgets.dragdroptab.**DragDropTabBar**(*parent*, *group_id*)

Bases: labscript_utils.qtwidgets.dragdroptab._BaseDragDropTabBar

FLUSH_GAP = 5

SCROLL_BUTTON_GAP = 2

SCROLL_BUTTON_WIDTH = 15

property drag_in_progress

property dragged_tab_index

property dragged_tab_parent

ensure_visible(*index*, *prefer_left=True*)

insertion_index_at(*pos*)

Compute at which index the tab with given upper left corner position in global coordinates should be inserted into the tabBar.

is_dragged_tab(*index*)

Return whether the tab at the given index is currently being dragged

limbo = None

minimumSizeHint(*self*) → QSize

mouseMoveEvent(*event*)

Update the parent of the tab to be the DragDropTabWidget under the mouse, if any, otherwise update it to the limbo object. Update the position of the tab in the widget it's in.

mousePressEvent(*event*)

Take note of the tab that was clicked so it can be dragged on mouseMoveEvents

mouseReleaseEvent(*event*)

Same as mouseMove event - update the DragDropTabWidget and position of the tab to the current mouse position. Unless the mouse position is outside of any widgets at the time of mouse release, in which case move the tab to its last known parent and position.

on_scroll_button_clicked(*button*)

paintEvent(*self*, *QPaintEvent*)

paint_tab(*index*, *painter*, *option*)

setUsesScrollButtons(*self*, *bool*)

set_tab_parent(*dest*, *index=None*, *pos=None*)
Move the tab to the given parent DragDropTabBar if it's not already there. index=None will determined the insertion index from the given mouse position.

sizeHint(*self*) → QSize

tabAt(*self*, *QPoint*) → int

tabInserted(*self*, *int*)

tabLayoutChange(*self*)

tabRect(*self*, *int*) → QRect

tabRemoved(*self*, *int*)

tab_widgets = {}

update(*self*)
update(self, QRect) update(self, QRegion) update(self, int, int, int, int)

update_dragged_tab_animation_pos(*pos*)

update_scroll_button_state()

update_tab_index(*index*, *pos*)
Check if the tab at the given index, being dragged by the mouse at the given position, needs to be moved. Move it and return the new index.

widgetAt(*pos*)
If the given position is over a DragDropTabBar belonging to the current group, return the DragDropTabBar. If it is over a TabWidget in the same group that has no tabs, or the dragged tab as its only tab, return its DragDropTabBar. Otherwise return the limbo object.

class labscript_utils.qtwidgets.dragdroptab.**DragDropTabWidget**(*group_id=None*, *accept_drops_bar_only=False*)
Bases: PyQt5.QtWidgets.QTabWidget

A tab widget that supports dragging and dropping of tabs between tab widgets that share a group_id. a group_id of None indicates that tab dragging is disabled.

setElideMode(*self*, *Qt.TextElideMode*)

setUsesScrollButtons(*self*, *bool*)

class labscript_utils.qtwidgets.dragdroptab.**Tab**(*widget*, *text*, *data*, *text_color*, *tooltip*, *whats_this*, *button_left*, *button_right*, *icon*)
Bases: tuple

property button_left
Alias for field number 6

property button_right
Alias for field number 7

property data
Alias for field number 2

property icon
Alias for field number 8

property text
Alias for field number 1

property text_color
Alias for field number 3

property tooltip
Alias for field number 4

property whats_this
Alias for field number 5

property widget
Alias for field number 0

class labscript_utils.qtwidgets.dragdroptab.**TabAnimation**(*parent*)
Bases: PyQt5.QtCore.QAbstractAnimation

animate_limbo(*limbo*, *index*)
If the floating tab in limbo is being sucked back into one of our tabs, then we can animate that by hiding the relevant tab rect off to the side somwhere whilst the floating tab swoops in.

duration(*self*) → int

ensure_running()

on_tab_moved(*source_index*, *dest_index*)

tabInserted(*index*)

tabRemoved(*index*)

target(*i*)
Return the target position we are animating toward for a tab

tau = 60.0

updateCurrentTime(*self*, *int*)

class labscript_utils.qtwidgets.dragdroptab.**debug**
Bases: object

DEBUG = False

depth = 0

classmethod trace(*f*)
decorator to print function entries and exits

class labscript_utils.qtwidgets.elide_label.**ElideScrollArea**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QScrollArea

A ScrollArea for containing a label that we want to elide. The elision is attained by just letting the text we don't want to see be scrolled off to the side with the scrollbars hidden.

event(*self*, *QEvent*) → bool

minimumSizeHint(*self*) → QSize

setElideMode(*elideMode*)

setWidget(*self*, *QWidget*)

sizeHint(*self*) → QSize

class labscript_utils.qtwidgets.elide_label.**ElidedLabelContainer**(*label*)
Bases: PyQt5.QtWidgets.QWidget

A QWidget to contain a QLabel with a single line of (possibly rich) text that we want to elide. The elision is obtained by putting the QLabel in a QScrollArea and having the QScrollarea only show the part of the text we want to see. An extra label with the elision indication "…" is also inserted next to the QScrollArea.

elideMode()

event(*self*, *QEvent*) → bool

minimumSizeHint(*self*) → QSize

setElideMode(*elideMode*)

sizeHint(*self*) → QSize

update_elide_widget()

labscript_utils.qtwidgets.elide_label.**elide_label**(*label*, *layout*, *elide_mode*)

Take an existing label that is in a layout, and wrap it in our widgets that elide the text, and insert it back into the layout. This is a hack that allows us to elide a QLabel with a single line of (possibly rich) text, a task that seems pretty much impossible to do in any kosher way.

This function is for modifying an existing label already in a layout, but if you are programatically creating a label, then you can wrap it in ElidedLabelContainer(label) before inserting it into a layout or other container widget, which is more flexible than this function which only works if the label is in a QBoxLayout

class labscript_utils.qtwidgets.fingertab.**FingerTabBarWidget**(*parent=None*, **args*, ***kwargs*)

Bases: PyQt5.QtWidgets.QTabBar

paintEvent(*self*, *QPaintEvent*)

tabSizeHint(*self*, *int*) → QSize

class labscript_utils.qtwidgets.fingertab.**FingerTabWidget**(*parent*, **args*)

Bases: PyQt5.QtWidgets.QTabWidget

A QTabWidget equivalent which uses our FingerTabBarWidget

class labscript_utils.qtwidgets.headerview_with_widgets.**HorizontalHeaderViewWithWidgets**(*mod*, *par*, *ent*=

Bases: PyQt5.QtWidgets.QHeaderView

A QHeaderView that supports inserting arbitrary widgets into sections. Use setWidget(logical_index, widget) to set and setWidget(logical_index, None) to unset. Decorations, checkboxes or anything other than text in the headers containing widgets is unsupported, and may result in garbled output

do_update_widget_positions()

eventFilter(*target*, *event*)

Ensure we don't leave the curor set as a resize handle when the mouse moves onto a child widget:

hideSection(*self*, *int*)

on_columnsInserted(*parent*, *logical_first*, *logical_last*)

on_columnsRemoved(*parent*, *logical_first*, *logical_last*)

sectionSizeFromContents(*self*, *int*) → QSize

setSectionHidden(*self*, *int*, *bool*)

setStyleSheet(*self*, *str*)

setWidget(*logical_index*, *widget=None*)

showEvent(*self*, *QShowEvent*)

showSection(*self*, *int*)

stylesheet = '\n QHeaderView::section {\n /* Will be set dynamically: */\n padding-to

thinspace = '\u2009'

update_indents()

update_widget_positions()

viewportEvent(*self*, *QEvent*) → bool

class labscript_utils.qtwidgets.InputPlotWindow.**PlotWindow**(**args*, ***kwargs*)

Bases: zprocess.process_tree.Process

run(*connection_name*, *hardware_name*, *device_name*)

The method that gets called in the subprocess. To be overridden by subclasses

setTopLevelWindow()

update_plot(*new_data*)

class labscript_utils.qtwidgets.outputbox.**OutputBox**(*container*, *scrollback_lines=1000*)

Bases: qtutils.outputbox.OutputBox

A subclass of qtutils.outputbox.OutputBox configured with security from labconfig.

class labscript_utils.qtwidgets.toolpalette.**ToolPalette**(*parent*, *name*, **args*, ***kwargs*)

Bases: PyQt5.QtWidgets.QScrollArea

addWidget(*widget*, *force_relayout=True*)

insertWidget(*index*, *widget*, *force_relayout=True*)

minimumSize(*self*) → QSize

minimumSizeHint(*self*) → QSize

resizeEvent(*self*, *QResizeEvent*)

sizeHint(*self*) → QSize

updateMinimumSize()

class labscript_utils.qtwidgets.toolpalette.**ToolPaletteGroup**(**args*, ***kwargs*)

Bases: PyQt5.QtWidgets.QVBoxLayout

add_to_linked_width_group(*width_group_name*, *name*)

append_new_palette(*name*, **args*, ***kwargs*)

create_linked_width_group(*width_group_name*, *names*)

get_index_from_name(*name*)

get_name_from_index(*index*)

get_palette(*name*)

get_palette_by_index(*index*)

has_palette(*name*)

hide_palette(*name*)

hide_palette_by_index(*index*)

insert_new_palette(*index*, *name*, **args*, ***kwargs*)

remove(*name*)

remove_by_index(*index*)

remove_from_linked_width_group(*width_group_name*, *name*)

reorder_palette(*name*, *new_index*)

reorder_palette_by_index(*old_index*, *new_index*)

show_palette(*name*)

show_palette_by_index(*index*)

property widths_linked

Input/Output Widgets

class labscript_utils.qtwidgets.analoginput.**AnalogInput**(*device_name*, *hardware_name*, *connection_name='-'*, *display_name=None*, *horizontal_alignment=False*, *parent=None*)

Bases: PyQt5.QtWidgets.QWidget

get_AI()

open_plot_window()

set_AI(*AI*, *notify_old_AI=True*, *notify_new_AI=True*)

set_value(*value*)

class labscript_utils.qtwidgets.analogoutput.**AnalogOutput**(*hardware_name*, *connection_name='-'*, *display_name=None*, *horizontal_alignment=False*, *parent=None*)

Bases: PyQt5.QtWidgets.QWidget

block_combobox_signals()

block_spinbox_signals()

connect_value_change(*func*)

disconnect_value_change()

eventFilter(*self*, *QObject*, *QEvent*) → bool

get_AO()

lock(*notify_ao=True*)

property selected_unit

set_AO(*AO*, *notify_old_AO=True*, *notify_new_AO=True*)

set_combobox_model(*model*)

set_limits(*lower*, *upper*)

set_num_decimals(*decimals*)

set_selected_unit(*unit*)

set_spinbox_value(*value*, *unit*)

set_step_size(*step*)

unblock_combobox_signals()

unblock_spinbox_signals()

unlock(*notify_ao=True*)

class labscript_utils.qtwidgets.analogoutput.**NoStealFocusDoubleSpinBox**(**args*, ***kwargs*)

Bases: PyQt5.QtWidgets.QDoubleSpinBox

A QDoubleSpinBox that doesn't steal focus as you scroll over it with a mouse wheel.

focusInEvent(*self*, *QFocusEvent*)

focusOutEvent(*self*, *QFocusEvent*)

wheelEvent(*self*, *QWheelEvent*)

class labscript_utils.qtwidgets.ddsoutput.**DDSOutput**(*hardware_name*, *connection_name='-'*, *parent=None*)

Bases: PyQt5.QtWidgets.QWidget

get_sub_widget(*subchnl*)

hide_sub_widget(*subchnl*)

show_sub_widget(*subchnl*)

class labscript_utils.qtwidgets.digitaloutput.**DigitalOutput**(**args*, ***kwargs*)

Bases: PyQt5.QtWidgets.QPushButton

eventFilter(*self*, *QObject*, *QEvent*) → bool

get_DO()

lock(*notify_do=True*)

set_DO(*DO*, *notify_old_DO=True*, *notify_new_DO=True*)

property state

unlock(*notify_do=True*)

class labscript_utils.qtwidgets.digitaloutput.**InvertedDigitalOutput**(**args*, ***kwargs*)

Bases: *labscript_utils.qtwidgets.digitaloutput.DigitalOutput*

property state

class labscript_utils.qtwidgets.enumoutput.**EnumOutput**(*hardware_name*, *connection_name='-'*, *display_name=None*, *horizontal_alignment=False*, *parent=None*)

Bases: PyQt5.QtWidgets.QWidget

block_combobox_signals()

connect_value_change(*func*)

disconnect_value_change()

eventFilter(*self*, *QObject*, *QEvent*) → bool

get_EO()

lock(*notify_eo=True*)

property selected_index

property selected_option

set_EO(*EO*, *notify_old_EO=True*, *notify_new_EO=True*)

set_combobox_model(*model*)

unblock_combobox_signals()

unlock(*notify_eo=True*)

class labscript_utils.qtwidgets.imageoutput.**BrowseButton**(*image_output*, **args*, ***kwargs*)

Bases: PyQt5.QtWidgets.QPushButton

browse()

eventFilter(*self*, *QObject*, *QEvent*) → bool

class labscript_utils.qtwidgets.imageoutput.**ImageOutput**(*name*, *width*, *height*, **args*, ***kwargs*)

Bases: PyQt5.QtWidgets.QWidget

eventFilter(*self*, *QObject*, *QEvent*) → bool

get_Image()

imageUpdated

lock(*notify_Image=True*)

set_Image(*Image*, *notify_old_Image=True*, *notify_new_Image=True*)

unlock(*notify_Image=True*)

property value

class labscript_utils.qtwidgets.imageoutput.**ImageView**(**args*, ***kwargs*)

Bases: PyQt5.QtWidgets.QGraphicsView

contextMenuEvent(*self*, *QContextMenuEvent*)

1.4 Logging and Profiling

1.4.1 labscript_utils.impprof

class labscript_utils.impprof.**_ProfilingImporter**

disable()

enable(*threshold=0.1*)

profiling_import(*name*, **args*, ***kwargs*)

labscript_utils.impprof.**disable**()

labscript_utils.impprof.**enable**(*threshold=0.1*)

1.4.2 labscript_utils.memprof

class labscript_utils.memprof.**MemoryProfiler**
Class to count number instances of each type in the interpreter in order to detect Python memory leaks

check()

count_types()

start(*filepath='memprof.txt'*)

write_to_file(*types*)

labscript_utils.memprof.**check**()

labscript_utils.memprof.**start**(*filepath='memprof.txt'*)

1.4.3 labscript_utils.setup_logging

class labscript_utils.setup_logging.**LessThanFilter**(*less_than*)

filter(*record*)
Determine if the specified record is to be logged.

Is the specified record to be logged? Returns 0 for no, nonzero for yes. If deemed appropriate, the record may be modified in-place.

labscript_utils.setup_logging.**setup_logging**(*program_name*, *log_level=10*, *terminal_level=20*, *maxBytes=52428800*, *backupCount=1*)

1.4.4 labscript_utils.tracelog

labscript_utils.tracelog.**log**(*log_path=None*, *module_names=*, *sub=False*, *all=False*, *mode='w'*)
Trace and log Python execution.

output includes the time, thread name, containing function name, line number and source line. Indentation before the thread name represents stack depth, indentation before source line is as in the source line itself.

log_path: the path of the desired output file to write to, or None for stdout (default=None) module_names: list of module names that tracing is desired for (default=()) sub: whether submodules of the above modules should be traced (default=False) all: whether all modules should be traced, in which case module_names is ignored (default=False) mode: mode to open the output file in, if log_path is not None (default='w')

1.5 Module and File Tools

1.5.1 labscript_utils.double_import_denier

class labscript_utils.double_import_denier.**DoubleImportDenier**
A module finder that tracks what's been imported and disallows multiple imports of the same module under different names, raising an exception upon detecting that this has occured

_format_tb(*tb*)
Take a formatted traceback as returned by traceback.format_stack() and remove lines that are solely about us and the Python machinery, leaving only lines pertaining to the user's code

_raise_error(*path*, *name*, *tb*, *other_name*, *other_tb*)

_restore_tracebacklimit_after_exception()
Record the current value of sys.tracebacklimit, if any, and set a temporary sys.excepthook to restore it to that value (or delete it) after the next exception.

find_spec(*fullname*, *path=None*, *target=None*)

labscript_utils.double_import_denier.**disable**()

labscript_utils.double_import_denier.**enable**()

1.5.2 labscript_utils.filewatcher

class labscript_utils.filewatcher.**FileWatcher**(*callback*, *files=None*, *folders=None*, *clean_modified_info=None*, *hashable_types=None*, *interval=1*, ***kwargs*)

_modified_info_of_file(*name*)

add_file(*path*)

add_files(*files*, *clean_modified_info=None*)

add_folder(*folder*)

add_folders(*folders*, *clean_modified_info=None*)

check(*trigger_callback=True*)

get_clean_modified_info()

get_modified_info()

get_modified_times()

mainloop()

stop()

update_files(*folders=None*, *trigger_callback=True*, *recursive=True*)
Refresh the watchlist of files (FileWatcher.files) by checking the folders kwarg or Filewatcher.folders if this is not specified.

1.5.3 labscript_utils.modulewatcher

class labscript_utils.modulewatcher.**ModuleWatcher**(*debug=False*)
A watcher that reloads modules that have been modified on disk

Only reloads modules imported after instantiation. Does not reload C extensions.

Parameters **debug** (*bool, optional*) – When `True`, prints debugging information when reloading modules.

check()

mainloop()

unload()

1.5.4 labscript_utils.versions

exception labscript_utils.versions.**BrokenInstall**

class labscript_utils.versions.**NoVersionInfo**

class labscript_utils.versions.**NotFound**

exception labscript_utils.versions.**VersionException**

labscript_utils.versions.**_get_literal_version**(*filename*)
Tokenize a source file and return any `__version__ = <version>` literal defined in it.

Parameters **filename** (*str*) – The path to the file to tokenize.

Returns Any version literal found matching the above criteria, or None.

labscript_utils.versions.**_get_metadata_version**(*project_name*, *import_path*)
Gets the package metadata version.

Parameters

- **project_name** (*str*) – The package name (e.g. the name used when pip installing the package).
- **import_path** (*str*) – The path to the folder containing the installed package.

Raises ***BrokenInstall*** – Raised if the package installation is corrupted (multiple packages matching the given arguments were found). May occur if (un)installation for a particular package version only partially completed.

Returns The metadata version for a package with the given project name located at the given import path, or None if there is no such package.

labscript_utils.versions.**check_version**(*module_name*, *at_least*, *less_than*, *version=None*, *project_name=None*)
Checks if a module version is within specified bounds.

Checks that the version of the given module is at least and less than the given version strings. This function uses *get_version()* to determine version numbers without importing modules. In order to do this, `project_name` must be provided if it differs from `module_name`. For example, pyserial is imported as 'serial', but the project name, as passed to a 'pip install' command, is 'pyserial'. Therefore to check the version of pyserial, pass in `module_name='serial'` and `project_name='pyserial'`. You can also pass in a version string yourself, in which case no inspection of packages will take place.

Parameters

- **module_name** (*str*) – The name of the module to check.
- **at_least** (*str*) – The minimum acceptable module version.
- **less_than** (*str*) – The minimum unacceptable module version. Usually this would be the next major version if the package follows semver.
- **version** (*str, optional*) – The current version of the installed package. Useful when the package version is stored in a non-standard location.
- **project_name** (*str, optional*) – The package name (e.g. the name used when pip installing the package). This must be specified if it does not match the module name.

Raises ***VersionException*** – if the module was not found or its version could not be determined.

labscript_utils.versions.**get_import_path**(*import_name*)
Get which entry in sys.path a module would be imported from, without importing it.

Parameters **import_name** (*str*) – The module name.

Raises

- **ModuleNotFoundError** – Raised if the module is not installed.
- **NotImplementedError** – Raised if the module is a "namespace package". Support for namepsace packages is not currently availabled.

Returns The path to the folder containing the module.

Return type str

labscript_utils.versions.**get_version**(*import_name*, *project_name=None*, *import_path=None*)

Try very hard to get the version of a package without importing it.

If import_path is not given, first find where it would be imported from, without importing it. Then look for metadata in the same import path with the given project name (note: this is not always the same as the import name, it is the name for example you would ask pip to install). If that is found, return the version info from it. Otherwise look for a `__version__.py` file in the package directory, or a `__version__ = <version>` literal defined in the package source (without executing it).

Parameters

- **import_name** (*str*) – The module name.
- **project_name** (*str, optional*) – The package name (e.g. the name used when pip installing the package). This must be specified if it does not match the module name.
- **import_path** (*str, optional*) – The path to the folder containing the installed package.

Raises **NotImplementedError** – Raised if the module name contains a period. Only top-level packages are supported at this time.

Returns The version literal of the package. If the package cannot be found, *NotFound* is returned. If the version cannot be obtained in the above way, or if the version was found but was `None`, *NoVersionInfo* is returned.

CHAPTER

TWO

LABSCRIPT SUITE COMPONENTS

The *labscript suite* is modular by design, and is comprised of:

Table 1: Python libraries

	labscript — Expressive composition of hardware-timed experiments
	labscript-devices — Plugin architecture for controlling experiment hardware
	labscript-utils — Shared modules used by the *labscript suite*

Table 2: Graphical applications

	runmanager — Graphical and remote interface to parameterized experiments
	blacs — Graphical interface to scientific instruments and experiment supervision
	lyse — Online analysis of live experiment data
	runviewer — Visualize hardware-timed experiment instructions

PYTHON MODULE INDEX

INDEX

Symbols

A

B

C

D

H

I

L

M

N

O

P

Q

R

S

U