MSL-Qt Documentation

Release 0.1.0.dev0

Measurement Standards Laboratory of New Zealand

May 31, 2020

Contents

This package provides custom Qt components that can be used for the graphical user interface.

CHAPTER 1

Contents

1.1 Install

To install **MSL-Qt** run:

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

Alternatively, using the MSL Package Manager run:

```
msl install qt
```

1.1.1 Dependencies

- Python 3.5+
- PyQt5 or Qt for Python – the Qt package that you want to use (you must install one, but the choice is yours)

You can automatically install one of these Qt packages when **MSL-Qt** is installed.

To also install PyQt5:

```
msl install qt[pyqt]
```

or, to also install Qt for Python:

```
msl install qt[pyside]
```

Optional Dependencies

- Python for .NET – only required if you want to load icons from DLL/EXE files on Windows

1.2 MSL-Qt API Documentation

The root package is

msl.qt	Custom Qt components for the graphical user interface.

which has the following modules

msl.qt.constants	Constants used by the **MSL-Qt** package.
msl.qt.convert	Functions to convert objects.
msl.qt.exceptions	Exception handling.
msl.qt.prompt	Convenience functions to prompt the user.
msl.qt.threading	Base classes for starting a process in a new QThread.
msl.qt.utils	General functions.

the following custom QWidget's

Button(*[, text, icon, icon_size, ...])	A QToolButton to display text, an icon and a menu.
LED(*[, parent, shape, on_color, off_color, ...])	An LED widget,
Logger(*[, level, fmt, datefmt, logger, parent])	A QWidget to display logging messages.
ToggleSwitch(*[, parent, height, on_color, ...])	Constructs a toggle switch,
DoubleSpinBox(*[, parent, value, minimum, ...])	A QDoubleSpinBox that emits editingFinished() after a stepBy() signal.
SpinBox(*[, parent, value, minimum, ...])	A QSpinBox that emits editingFinished() after a stepBy() signal.

and the following convenience classes

LoopUntilAbort(*[, loop_delay, ...])	Repeatedly perform a task until aborted by the user.
Sleep()	Sleep without freezing the graphical user interface.

1.2.1 Package Structure

msl.qt package

Custom Qt components for the graphical user interface.

msl.qt.**Signal**(**args*, ***kwargs*)

msl.qt.**Slot**(**args*, ***kwargs*)

msl.qt.**application**(**args*)
Returns the QtWidgets.QApplication instance (creating one if necessary).

Parameters **args** – A list of arguments to initialize the application. If None then uses sys.argv.

Returns QtWidgets.QApplication – The application instance.

msl.qt.**version_info = version_info(major=0, minor=1, micro=0, releaselevel='dev0**
Contains the version information as a (major, minor, micro, releaselevel) tuple.

Type namedtuple

msl.qt.constants module

Constants used by the **MSL-Qt** package.

class msl.qt.constants.**GREEK**
Bases: enum.Enum

Greek letters.

Alpha = 'A'

Beta = 'B'

Chi = 'X'

Delta = 'Δ'

Epsilon = 'E'

Eta = 'H'

Gamma = 'Γ'

Gammad = ''

Iota = 'I'

Kappa = 'K'

Lambda = 'Λ'

Mu = 'M'

Nu = 'N'

Omega = 'Ω'

Omicron = 'O'

Phi = 'Φ'

Pi = 'Π'

Psi = 'Ψ'

Rho = 'P'

Sigma = 'Σ'

Tau = 'T'

Theta = 'Θ'

```
Upsilon = 'Υ'
Xi = 'Ξ'
Zeta = 'Z'
alpha = 'α'
backepsilon = ''
beta = 'β'
chi = 'χ'
delta = 'δ'
epsilon = 'ϵ'
eta = 'η'
gamma = 'γ'
gammad = ''
iota = 'ι'
kappa = 'κ'
lambda_ = 'λ'
mu = 'μ'
nu = 'ν'
omega = 'ω'
omicron = 'o'
phi = 'ϕ'
pi = 'π'
piv = ''
psi = 'ψ'
rho = 'ρ'
sigma = 'σ'
sigmaf = ''
straightepsilon = ''
straightphi = ''
tau = 'τ'
theta = 'θ'
thetasym = ''
upsih = ''
upsilon = 'υ'
varkappa = ''
```

varrho = 'ϱ'

xi = 'ξ'

zeta = 'ζ'

msl.qt.constants.**HOME_DIR = '/home/docs/.msl'**
The default $HOME directory where all files used by **MSL-Qt** are located.

Type str

msl.qt.constants.**SI_PREFIX_MAP = {-8: 'y', -7: 'z', -6: 'a', -5: 'f', -4: '**
The SI prefixes used to form multiples of 10**(3*n), n=-8..8

Type dict

msl.qt.convert module

Functions to convert objects.

msl.qt.convert.**icon_to_base64**(*icon*, *, *fmt='png'*)
Convert an icon to a QtCore.QByteArray encoded as Base64.

This function is useful if you want to save icons in a database, use it in a data URI scheme, or if you want to use icons in your GUI and rather than loading icons from a file on the hard disk you define your icons in a Python module as Base64 variables. Loading the icons from the hard disk means that you must also distribute the icons with your Python code if you share your code.

Parameters

- **icon** – An icon with a data type that is handled by *to_qicon()*.
- **fmt** (str, optional) – The icon format to use when converting. The supported values are: BMP, JPG, JPEG and PNG.

Returns QtCore.QByteArray – The Base64 representation of the icon.

Raises

- IOError – If the icon file cannot be found.
- ValueError – If the icon format, *fmt*, to use for converting is not supported.

msl.qt.convert.**rescale_icon**(*icon*, *size*, *, *aspect_mode=1*)
Rescale an icon.

Parameters

- **icon** – Any object that is supported by *to_qicon()*.
- **size** (int, float, tuple of int or QtCore.QSize) – Rescale the icon to the specified *size*. If an int then set the width and the height to be the *size* value. If a float then a scaling factor. If a tuple then the (width, height) values.
- **aspect_mode** (QtCore.Qt.AspectRatioMode, optional) – How to maintain the aspect ratio if rescaling. The default mode is to keep the aspect ratio.

Returns QtGui.QPixmap – The rescaled icon.

msl.qt.convert.**number_to_si**(*number*)

Convert a number to be represented with an SI prefix.

The hecto (h), deka (da), deci (d) and centi (c) prefixes are not used.

Parameters **number** (int or float) – The number to convert.

Returns

- float – The number rescaled.
- str – The SI prefix.

Examples

```
>>> number_to_si(0.0123)
(12.3, 'm')
>>> number_to_si(123456.789)
(123.456789, 'k')
>>> number_to_si(712.123e14)
(71.2123, 'P')
>>> number_to_si(1.23e-13)
(123.0, 'f')
```

msl.qt.convert.**print_base64**(*icon*, *, *size=None*, *name=''*, *line_width=80*, *file=None*)

Print the Base64 representation of an icon.

Parameters

- **icon** – Passed to *to_qicon()*.
- **size** – Passed to *to_qicon()*.
- **name** (str, optional) – The name of the icon.
- **line_width** (int, optional) – The maximum number of characters in a line.
- **file** (file-like object) – Where to print the output. Default is sys.stdout.

Examples

```
>>> print_base64(QtWidgets.QStyle.SP_MediaPlay, size=16)
b'iVBORw0KGgoAAAANSUhEUgAAABAAAAAQCAYAAAAf8/
 ↪9hAAAACXBIWXMAAAk6AAAJOgHwZJJKAAA' \
b
 ↪'AGXRFWHRTb2Z0d2FyZQB3d3cuaW5rc2NhcGUub3Jnm+48GgAAAJ9JREFUOI3djzEKwlAQRGcXO1
 ↪' \
b'ERrIQw/i7gCTyOvZfyTtaBjaCVEH9n8V3bID/GpBKnnJ157AA/
 ↪p6Io1kPy8m6QjCLyVFVWVXXvA' \
b
 ↪'2jOdPdFSqkheRoFaGlL8hFCOHQFshMAzDLZCKA0s+uQD9qaA7iQPI8FAEBjZpsxgCgiOzNbAkjt
 ↪' \
b'w6Sn6O5+rOt63xX4BLiZ2arvtdyEqaqW35T/RC/uTS/6P1rpJAAAAABJRU5ErkJggg==
 ↪'
```

```
>>> print_base64(QtWidgets.QStyle.SP_MediaPlay, name='my_play_icon')
my_play_icon = b
↪'iVBORw0KGgoAAAANSUhEUgAAACAAAAAgCAYAAABzenr0AAAACXBIWXMAAAk6' \
               b
↪'AAAJOgHwZJJKAAAAGXRFWHRTb2Z0d2FyZQB3d3cuaW5rc2NhcGUub3Jnm+48' \
               b
↪'GgAAAaVJREFUWIXtlLFuE0EURc8bO8gyiiMhUYC8M2vHldPhz+AD6LDSp0FU' \
               b
↪'QMsHUEJBAx+QjoKGKlIKhCihcDSzcYEsCylGXoTA+yhQUBSReLXebINPOzP3' \
               b
↪'Hr0nDaxZs4Qoim5fZb657LDf718zxhw7595ba3cqF0jT1ABz4I4x5sBa+7zb' \
               b
↪'7W5VJnAGUdUtERkuFoujOI53ASlD4NKQOI4bqjoBNs8dzYBj4H4I4cMqAnkn' \
               b'cJ4WsAO8s9a+arfbN6oW+CsiIvdqtdqo0+nsFckruoJ/
↪8Q2YqOowSZKDvAKr' \
               b
↪'TuAsm8C2iLxxzu07525VLXBKC7gLfHLOPR4MBhtVCwBsAC1VfTSdTo+iKNqu' \
               b'WgAAEVHglzEmu+hO/Yq6f/
↪LnB30aQngGLKoUOAHe1uv1vdFoNFl2uUyBmYh8' \
               b'AYbe+8O8j8oQ+AGkIvLEe/8CuHDfZQsoMFPV/
↪SzLHo7H469FQgoJiMgJEIBh' \
               b'COFjkYyiAt+BNMuyB0mSvF6l+JS8/
↪4CKyAx42Wg0OmWVw5IJNJvNbD6fXwcO' \
               b'RWTXe/+5rOLc9Hq9m5WXrlnzX/EbbYB/
↪8sxND3cAAAAASUVORK5CYII='
```

msl.qt.convert.**si_to_number**(*string*)

Convert a string with an SI prefix to a number.

Parameters **string** (str) – The string to convert.

Returns float – The number.

Examples

```
>>> si_to_number('12.3m')
0.0123
>>> si_to_number('123.456789k')
123456.789
>>> si_to_number('71.2123P')
7.12123e+16
>>> si_to_number('123f')
1.23e-13
```

msl.qt.convert.**to_qcolor**(**args*)

Convert the input argument(s) into a QtGui.QColor.

Parameters **args** – The argument(s) to convert to a QtGui.QColor.

- R, G, B, [A] → values can be int 0-255 or float 0.0-1.0
- (R, G, B, [A]) → tuple of int 0-255 or float 0.0-1.0
- int or QtCore.Qt.GlobalColor → a pre-defined enum value
- float → a greyscale value between 0.0-1.0
- QtGui.QColor → returns a copy

- `str` → see here for examples

Returns `QtGui.QColor` – The input argument(s) converted to a `QtGui.QColor`.

Examples

```
>>> color = to_qcolor(48, 127, 69)
>>> color = to_qcolor((48, 127, 69))
>>> color = to_qcolor(0.5)  # greyscale -> (127, 127, 127, 255)
>>> color = to_qcolor(0.2, 0.45, 0.3, 0.5)
>>> color = to_qcolor('red')
>>> color = to_qcolor(Qt.darkBlue)
>>> color = to_qcolor(15)  # 15 == Qt.darkBlue
```

`msl.qt.convert.`**`to_qfont`**(**args*)

Convert the input argument(s) into a `QtGui.QFont`.

Parameters **`args`** – The argument(s) to convert to a `QtGui.QFont`.

- If `int` or `float` then the point size.
- If `str` then the font family name.
- If `QtGui.QFont` then returns a copy.
- If multiple arguments then
  - family name, point size
  - family name, point size, weight
  - family name, point size, weight, is italic

Returns `QtGui.QFont` – The input argument(s) converted to a `QtGui.QFont`.

Examples

```
>>> font = to_qfont(48)
>>> font = to_qfont(23.4)
>>> font = to_qfont('Papyrus')
>>> font = to_qfont('Ariel', 16)
>>> font = to_qfont('Ariel', 16, QtGui.QFont.Bold)
>>> font = to_qfont('Ariel', 16, 50, True)
```

`msl.qt.convert.`**`to_qicon`**(*obj*, *, *size=None*, *aspect_mode=1*)

Convert the input object to a `QtGui.QIcon`.

Parameters

- **`obj`** – The object to be converted to a `QtGui.QIcon`. The data type of *obj* can be one of:
  - `QtGui.QIcon`
  - `QtGui.QPixmap`
  - `QtGui.QImage`

– QtWidgets.QStyle.StandardPixmap: One of the built-in Qt pixmaps. Example:

```
to_qicon(QtWidgets.QStyle.SP_TitleBarMenuButton)
to_qicon(14)  # the QtWidgets.QStyle.SP_TrashIcon␣
↪enum value
```

– QtCore.QByteArray: A Base64 representation of an encoded icon.

See *icon_to_base64()*.

– str: The path to an icon file or an icon embedded in a DLL or EXE file.

If *obj* is a path to an icon file and only the filename is specified then the directories in sys.path and os.environ['PATH'] are also used to search for the icon file. If *obj* refers to an icon in a Windows DLL/EXE file then *obj* is the path to the DLL/EXE file and the icon index separated by the | character.

The following examples illustrate the various ways to request an icon by passing in a str argument:

```
# provide the full path to the icon file
to_qicon('D:/code/resources/icons/msl.png')
to_qicon('D:/code/resources/icons/photon.png')

# insert the folder where the icons are located in to␣
↪sys.path
sys.path.insert(0, 'D:/code/resources/icons/')
# so now only the filename needs to be specified to␣
↪load the icon
to_qicon('msl.png')
to_qicon('photon.png')

# load icon 23 from the Windows shell32.dll file
to_qicon('C:/Windows/System32/shell32.dll|23')

# load icon 0 from the Windows explorer.exe file
to_qicon('C:/Windows/explorer.exe|0')

# it is assumed that the DLL/EXE file is located in a␣
↪default directory:
#   - a DLL file in C:/Windows/System32/
#   - an EXE file in C:/Windows/
# so the following is a simplified way to load an␣
↪icon in a DLL file
to_qicon('shell32|23')
to_qicon('imageres|1')
to_qicon('compstui|51')
# and the following is a simplified way to load an␣
↪icon in an EXE file
to_qicon('explorer|0')
```

- **size** (int, float, tuple of int or QtCore.QSize, optional) – Rescale the icon to the specified *size*. If the value is None then do not rescale the icon. If an int then set the width and the height to be the *size* value. If a float then a scaling factor. If a tuple then the (width, height) values.

- **aspect_mode** (QtCore.Qt.AspectRatioMode, optional) – How to maintain the aspect ratio if rescaling. The default mode is to keep the aspect ratio.

Returns QtGui.QIcon – The input object converted to a QtGui.QIcon.

Raises

- IOError – If the icon cannot be found.
- TypeError – If the data type of *obj* or *size* is not supported.

Example

To view the standard icons that come with Qt and that come with Windows run:

```
>>> from msl.examples.qt import ShowStandardIcons
>>> ShowStandardIcons()
```

msl.qt.exceptions module

Exception handling.

msl.qt.exceptions.**excepthook**(*exctype*, *value*, *traceback*)
Displays unhandled exceptions in a QtWidgets.QMessageBox.

See sys.excepthook() for more details.

To implement the *excepthook()* in your own application include the following:

```
import sys
from msl import qt

sys.excepthook = qt.excepthook
```

msl.qt.loop_until_abort module

Repeatedly perform a task until aborted by the user.

class msl.qt.loop_until_abort.**LoopUntilAbort**(*, *loop_delay=0*, *max_iterations=None*, *single_shot=False*, *title=None*, *bg_color='#DFDFDF'*, *text_color='#20548B'*, *font_family='Helvetica'*, *font_size=14*)

Bases: object

Repeatedly perform a task until aborted by the user.

This class provides an interface to show the status of a task (e.g., read a sensor value and write the value to a file) that you want to perform for an unknown period of time (e.g., during lunch or overnight) and you want to stop the task whenever you return. It can be regarded as a way to tell your program to *"get as much data as possible until I get back"*.

The following example illustrates how to repeatedly write data to a file in a loop:

```
"""
Example script to repeatedly write data to a file until aborted by␣
↪the user.
"""
import tempfile

from msl.qt import (
    prompt,
    LoopUntilAbort
)


class LoopExample(LoopUntilAbort):

    def __init__(self):
        """Initialize the LoopUntilAbort class and create a file to␣
↪write data to.

        Use a 250 ms delay between successive calls to the `loop`␣
↪method.
        """
        super(LoopExample, self).__init__(loop_delay=250)

        self.output_path = tempfile.gettempdir() + '/msl-qt-loop-
↪until-abort.txt'
        self.f = open(self.output_path, 'w')
        self.f.write('Started at {}\n'.format(self.current_time))

    def loop(self):
        """Overrides LoopUntilAbort.loop()

        This method gets called repeatedly in a loop (every `loop_
↪delay` ms).
        """
        self.f.write('Iteration: {}\n'.format(self.iteration))
        self.f.write('Elapsed time: {}\n'.format(self.elapsed_time))
        self.set_label_text('The current time is\n' + str(self.
↪current_time))

    def cleanup(self):
        """Overrides LoopUntilAbort.cleanup()

        This method gets called when the LoopExample window is␣
↪closing.
        """
        self.f.write('Stopped at {}\n'.format(self.current_time))
        self.f.close()
        msg = 'The data was save to\n{}\n\n... in case you want to␣
↪look at it'
        prompt.information(msg.format(self.output_path))


def main():
    loop = LoopExample()
    loop.start()
```

(continues on next page)

(continued from previous page)

```
if __name__ == '__main__':
    main()
```

Examples

To run the above example enter the following:

```
>>> from msl.examples.qt import LoopExample
>>> loop = LoopExample()
>>> loop.start()
```

Another example which uses *single-shot* mode:

```
>>> from msl.examples.qt import LoopExampleSleep
>>> loop = LoopExampleSleep()
>>> loop.start()
```

Parameters

- **loop_delay** (int, optional) – The delay time, in milliseconds, to wait between successive calls to the *loop()* method. For example, if *loop_delay* = 0 then there is no time delay between successive calls to the *loop()* method; if *loop_delay* = 1000 then wait 1 second between successive calls to the *loop()* method.
- **max_iterations** (int, optional) – The maximum number of times to call the *loop()* method. The default value is None, which means to loop until the user aborts the program.
- **single_shot** (bool, optional) – Whether to call the *loop()* method once (and only once). If you specify the value to be True then you must call the *loop_once()* method in the subclass whenever you want to run the *loop()* one more time. This is useful if the *loop()* depends on external factors (e.g., waiting for an oscilloscope to download a trace after a trigger event) and the amount of time that the *loop()* requires to complete is not known.
- **title** (str, optional) – The text to display in the title bar of the QtWidgets.QMainWindow. If None then uses the name of the subclass as the title.
- **bg_color** (QtGui.QColor, optional) – The background color of the QtWidgets.QMainWindow. Can be any data type and value that the constructor of a QtGui.QColor accepts.
- **text_color** (QtGui.QColor, optional) – The color of the **Elapsed time** and **Iteration** text.Can be any data type and value that the constructor of a QtGui.QColor accepts.
- **font_family** (QtGui.QFont, optional) – The font family to use for the text. Can be any value that the constructor of a QtGui.QFont accepts.

- **font_size** (int, optional) – The font size of the text.

cleanup()
This method gets called when the QtWidgets.QMainWindow is closing.

You can override this method to properly cleanup any tasks. For example, to close a file that is open.

current_time
The current time.

Type datetime.datetime

elapsed_time
The elapsed time since the *loop()* started.

Type datetime.datetime

iteration
The number of times that the *loop()* method has been called.

Type int

loop()
The task to perform in a loop.

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

loop_delay
The time delay, in milliseconds, between successive calls to the *loop()*.

Type int

loop_once()
Run the *loop()* method once.

This method should be called if the *LoopUntilAbort* class was initialized with *single_shot* = True, in order to run the *loop()* method one more time.

loop_timer
The reference to the *loop()*'s timer.

Type QtCore.QTimer

main_window
The reference to the main window.

Type QtWidgets.QMainWindow

max_iterations
The maximum number of times that the *loop()* will be called.

Type int or None

set_label_text(*text*)
Update the text of the label that the user has access to.

Parameters **text** (str) – The text to display in the user-accessible label.

See also:

user_label() For the reference to the `QtWidgets.QLabel` object.

set_status_bar_text(*text*)
Set the text to display in the status bar of the `QtWidgets.QMainWindow`.

Parameters **text** (`str`) – The text to display in the status bar of the `QtWidgets.QMainWindow`.

start()
Show the `QtWidgets.QMainWindow` and start looping.

start_time
The time when the *loop()* started.

Type `datetime.datetime`

user_label
The reference to the label object that the user can modify the text of.

See also:

set_label_text() To set the text of the `QtWidgets.QLabel`.

Type `QtWidgets.QLabel`

msl.qt.prompt module

Convenience functions to prompt the user.

The following functions create a dialog window to either notify the user of an event that happened or to request information from the user.

msl.qt.prompt.**comments**(*, *path=None*, *title=None*, *even_row_color='#FFFFFF'*, *odd_row_color='#EAF2F8'*)
Ask the user to enter comments.

Opens a `QtWidgets.QDialog` to allow for a user to enter comments about a task that they are performing. The dialog provides a table of all the previous comments that have been used. Comments that are in the table can be deleted by selecting the desired row(s) and pressing the `Delete` key or the comment in a row can be selected by double-clicking on a row.

This function is useful when acquiring data and you want to include comments about how the data was acquired. Using a prompt to enter comments forces you to enter a new comment (or use a previous comment) every time you acquire data rather than having the comments in a, for example `QtWidgets.QPlainTextEdit`, which you might forget to update before acquiring the next data set.

Parameters

- **path** (`str`, optional) – The path to a JSON file that contains the history of the comments that have been used. If `None` then the default file is used. The file will automatically be created if it does not exist.
- **title** (`str`, optional) – The text to display in the title bar of the dialog window.

- **even_row_color** – The background color of the even-numbered rows in the history table. See *to_qcolor()* for details about the different data types that are supported.
- **odd_row_color** – The background color of the odd-numbered rows in the history table. See *to_qcolor()* for details about the different data types that are supported.

Returns str – The comment that was entered.

msl.qt.prompt.**critical**(*message*, *, *title=None*, *font=None*)

Display a *critical* message in a dialog window.

Parameters

- **message** (str or Exception) – The message to display. The message can use HTML/CSS markup, for example, '<html>A critical error occurred!</html>'
- **title** (str, optional) – The text to display in the title bar of the dialog window. If None then uses the text in the title bar of the active window.
- **font** (int, str, tuple or QtGui.QFont, optional) – The font to use. If an int then the point size, if a str then the family name, if a tuple then the (family name, point size).

msl.qt.prompt.**double**(*message*, *, *title=None*, *font=None*, *value=0*, *minimum=-2147483647*, *maximum=2147483647*, *step=1*, *decimals=2*)

Request a double-precision value (a Python float data type).

Parameters

- **message** (str) – The message that is shown to the user to describe what the value represents. The message can use HTML/CSS markup, for example, '<html>Enter a mass, in μg</html>'
- **title** (str, optional) – The text to display in the title bar of the dialog window. If None then uses the text in the title bar of the active window.
- **font** (int, str, tuple or QtGui.QFont, optional) – The font to use. If an int then the point size, if a str then the family name, if a tuple then the (family name, point size).
- **value** (float, optional) – The initial value.
- **minimum** (float, optional) – The minimum value that the user can enter.
- **maximum** (float, optional) – The maximum value that the user can enter.
- **step** (float, optional) – The step size by which the value is increased and decreased.
- **decimals** (int, optional) – The number of digits that are displayed after the decimal point.

Returns float or None – The value or None if the user cancelled the request.

msl.qt.prompt.**filename**(*, *title='Select File'*, *directory=None*, *filters=None*, *multiple=False*)

Request to select the file(s) to open.

Parameters

- **title** (str, optional) – The text to display in the title bar of the dialog window.
- **directory** (str, optional) – The initial directory to start in.
- **filters** (str, list of str or dict, optional) – Only filenames that match the specified *filters* are shown.

  Examples:

```
'Images (*.png *.xpm *.jpg)'
'Images (*.png *.xpm *.jpg);;Text files (*.txt);;XML␣
↪files (*.xml)'
['Images (*.png *.xpm *.jpg)', 'Text files (*.txt)',
↪'XML files (*.xml)']
{'Images': ('*.png', '*.xpm', '*.jpg'), 'Text files':
↪'*.txt'}
```

- **multiple** (bool, optional) – Whether multiple files can be selected.

Returns str, list of str or None – The name(s) of the file(s) to open or None if the user cancelled the request.

msl.qt.prompt.**folder**(*, *title='Select Folder'*, *directory=None*)

Request to select an existing folder or to create a new folder.

Parameters

- **title** (str, optional) – The text to display in the title bar of the dialog window.
- **directory** (str, optional) – The initial directory to start in.

Returns str or None – The name of the selected folder or None if the user cancelled the request.

msl.qt.prompt.**information**(*message*, *, *title=None*, *font=None*)

Display an *information* message in a dialog window.

Parameters

- **message** (str or Exception) – The message to display. The message can use HTML/CSS markup, for example, '<html>The temperature is 21.3 °C</html>'
- **title** (str, optional) – The text to display in the title bar of the dialog window. If None then uses the text in the title bar of the active window.
- **font** (int, str, tuple or QtGui.QFont, optional) – The font to use. If an int then the point size, if a str then the family name, if a tuple then the (family name, point size).

msl.qt.prompt.**integer**(*message*, *, *title=None*, *font=None*, *value=0*, *minimum=-2147483647*, *maximum=2147483647*, *step=1*)

Request an integer value.

Parameters

- **message** (str) – The message that is shown to the user to describe what the value represents. The message can use HTML/CSS markup, for example, '<html>Enter a mass, in μg</html>'
- **title** (str, optional) – The text to display in the title bar of the dialog window. If None then uses the text in the title bar of the active window.
- **font** (int, str, tuple or QtGui.QFont, optional) – The font to use. If an int then the point size, if a str then the family name, if a tuple then the (family name, point size).
- **value** (int, optional) – The initial value.
- **minimum** (int, optional) – The minimum value that the user can enter.
- **maximum** (int, optional) – The maximum value that the user can enter.
- **step** (int, optional) – The step size by which the value is increased and decreased.

Returns int or None – The value or None if the user cancelled the request.

msl.qt.prompt.**item**(*message*, *items*, *, *title=None*, *font=None*, *index=0*)
Request an item from a list of items.

Parameters

- **message** (str) – The message that is shown to the user to describe what the list of items represent. The message can use HTML/CSS markup.
- **items** (list or tuple) – The list of items to choose from. The items can be of any data type.
- **title** (str, optional) – The text to display in the title bar of the dialog window. If None then uses the text in the title bar of the active window.
- **font** (int, str, tuple or QtGui.QFont, optional) – The font to use. If an int then the point size, if a str then the family name, if a tuple then the (family name, point size).
- **index** (int, optional) – The index of the initial item that is selected.

Returns The selected item or None if the user cancelled the request.

Notes

The data type of the selected item is preserved. For example, if *items* = [1, 2.0, 2+3j, 'hello', b'world', True, QtWidgets.QPushButton] and the user selects the 2+3j item then a complex data type is returned.

msl.qt.prompt.**ok_cancel**(*message*, *, *title=None*, *font=None*, *default=True*)
Ask for a response to a message where the logical options are Ok or Cancel.

Parameters

- **message** (str) – The message to ask the user. The message can use HTML/CSS markup.
- **title** (str, optional) – The text to display in the title bar of the dialog window. If None then uses the text in the title bar of the active window.

- **font** (int, str, tuple or QtGui.QFont, optional) – The font to use. If an int then the point size, if a str then the family name, if a tuple then the (family name, point size).
- **default** (bool, optional) – The answer to be selected by default. If True then Ok is the default answer, otherwise Cancel is the default answer.

Returns bool or None – True if the user answered Ok, None if the user answered Cancel.

msl.qt.prompt.**password**(*message*, *, *title=None*, *font=None*)
Request a password.

Parameters

- **message** (str) – The message that is shown to the user to describe what the list of items represent. The message can use HTML/CSS markup.
- **title** (str, optional) – The text to display in the title bar of the dialog window. If None then uses the text in the title bar of the active window.
- **font** (int, str, tuple or QtGui.QFont, optional) – The font to use. If an int then the point size, if a str then the family name, if a tuple then the (family name, point size).

Returns str or None – The password or None if the user cancelled the request.

msl.qt.prompt.**save**(*, *title='Save As'*, *directory=None*, *filters=None*, *options=None*)
Request to enter the name of a file to save.

Parameters

- **title** (str, optional) – The text to display in the title bar of the dialog window.
- **directory** (str, optional) – The initial directory to start in.
- **filters** (str, list of str or dict, optional) – Only filenames that match the specified *filters* are shown.

  Examples:

```
'Images (*.png *.xpm *.jpg)'
'Images (*.png *.xpm *.jpg);;Text files (*.txt);;XML␣
↪files (*.xml)'
['Images (*.png *.xpm *.jpg)', 'Text files (*.txt)',
↪'XML files (*.xml)']
{'Images': ('*.png', '*.xpm', '*.jpg'), 'Text files':
↪'*.txt'}
```

- **options** (QtWidgets.QFileDialog.Option, optional) – Specify additional options on how to run the dialog.

Returns str or None – The name of the file to save or None if the user cancelled the request.

msl.qt.prompt.**text**(*message*, *, *title=None*, *font=None*, *value=''*, *multi_line=False*, *echo=0*)
Request text.

Parameters

- **message** (str) – The message that is shown to the user to describe what the list of items represent. The message can use HTML/CSS markup.
- **title** (str, optional) – The text to display in the title bar of the dialog window. If None then uses the text in the title bar of the active window.
- **font** (int, str, tuple or QtGui.QFont, optional) – The font to use. If an int then the point size, if a str then the family name, if a tuple then the (family name, point size).
- **value** (str, optional) – The initial value.
- **multi_line** (bool, optional) – Whether the entered text can span multiple lines.
- **echo** (int or QLineEdit.EchoMode, optional) – The echo mode for the text value. Useful if requesting a password.

Returns str or None – The text that the user entered or None if the user cancelled the request.

msl.qt.prompt.**warning**(*message*, *, *title=None*, *font=None*)
Display a *warning* message in a dialog window.

Parameters

- **message** (str or Exception) – The message to display. The message can use HTML/CSS markup, for example, '<html>A warning ...</html>'
- **title** (str, optional) – The text to display in the title bar of the dialog window. If None then uses the text in the title bar of the active window.
- **font** (int, str, tuple or QtGui.QFont, optional) – The font to use. If an int then the point size, if a str then the family name, if a tuple then the (family name, point size).

msl.qt.prompt.**yes_no**(*message*, *, *title=None*, *font=None*, *default=True*)
Ask for a response to a message where the logical options are Yes or No.

Parameters

- **message** (str) – The message to ask the user. The message can use HTML/CSS markup.
- **title** (str, optional) – The text to display in the title bar of the dialog window. If None then uses the text in the title bar of the active window.
- **font** (int, str, tuple or QtGui.QFont, optional) – The font to use. If an int then the point size, if a str then the family name, if a tuple then the (family name, point size).
- **default** (bool, optional) – The answer to be selected by default. If True then Yes is the default answer, otherwise No is the default answer.

Returns bool – True if the user answered Yes, False if the user answered No.

msl.qt.prompt.**yes_no_cancel**(*message*, *, *title=None*, *font=None*, *default=True*)
Ask for a response to a message where the logical options are Yes, No or Cancel.

Parameters

- **message** (str) – The message to ask the user. The message can use HTML/CSS markup.
- **title** (str, optional) – The text to display in the title bar of the dialog window. If None then uses the text in the title bar of the active window.
- **font** (int, str, tuple or QtGui.QFont, optional) – The font to use. If an int then the point size, if a str then the family name, if a tuple then the (family name, point size).
- **default** (bool, optional) – The answer to be selected by default. If True then Yes is the default answer, if False then No is the default answer, else if None then Cancel is the default answer.

Returns bool or None – True if the user answered Yes, False if the user answered No, or None if the user answered Cancel.

msl.qt.sleep module

Sleep without freezing the graphical user interface.

class msl.qt.sleep.**Sleep**

Bases: *msl.qt.threading.Thread*

Sleep without freezing the graphical user interface.

The following example illustrates how one can use *Sleep* and keep the GUI active:

```
"""
Illustrates the use of the :class:`~msl.qt.loop_until_abort.
↪LoopUntilAbort` class
in *single shot* mode using a random :class:`~msl.qt.sleep.Sleep`␣
↪time.
"""
import random

from msl.qt import (
    LoopUntilAbort,
    Sleep
)


class LoopExampleSleep(LoopUntilAbort):

    def __init__(self):
        """Initialize the LoopUntilAbort class in single-shot mode."""
        super(LoopExampleSleep, self).__init__(single_shot=True)

        # create the Sleep thread which uses SleepWorker as the␣
↪worker class
        self.sleep = Sleep()

        # call self.update_status_bar_text when sleeping is finished
        self.sleep.finished.connect(self.update_status_bar_text)

        # just for fun, add another slot to be called when sleeping␣
↪is finished
```

(continues on next page)

(continued from previous page)

```
        self.sleep.finished.connect(self.countdown)

    def countdown(self):
        if self.iteration == 5:
            # stop being notified of the countdown
            self.sleep.finished.disconnect(self.countdown)
        print('countdown: ' + str(5 - self.iteration))

    def update_status_bar_text(self):
        print('finished loop #: ' + str(self.iteration))

        # you can access attributes from the SleepWorker class, i.e.,␣
↪self.sleep.seconds
        self.set_status_bar_text('Slept for {:.3f} seconds'.
↪format(self.sleep.seconds))

        # run the loop one more time (only if the loop's QTimer still␣
↪exists)
        # NOTE: do not call self.loop() directly, you should call␣
↪self.loop_once()
        if self.loop_timer is not None:
            self.loop_once()

    def loop(self):
        """Overrides LoopUntilAbort.loop()"""

        print('started loop #: ' + str(self.iteration))

        # pick a random number of seconds to sleep for
        sleep_for_this_many_seconds = random.randint(100, 2000) * 1e-3

        # the `sleep_for_this_many_seconds` parameter gets passed to␣
↪the
        # constructor of the SleepWorker class and the Sleep thread␣
↪starts
        self.sleep.start(sleep_for_this_many_seconds)

        # the self.sleep.start() call does not block (if it did the␣
↪GUI would freeze)
        # so any code that is after it will be executed immediately --
↪ this is why
        # the text of the "label" gets updated immediately
        self.set_label_text('Sleeping for {:.3f} seconds'.
↪format(sleep_for_this_many_seconds))

    def cleanup(self):
        """Overrides LoopUntilAbort.cleanup()"""

        # let's be nice and stop the Sleep thread before exiting the␣
↪program
        # waiting for the thread to stop is a blocking call so the␣
↪GUI will be frozen while it is closing
        self.sleep.stop()


def main():
```

(continues on next page)

(continued from previous page)

```
    loop = LoopExampleSleep()
    loop.start()


if __name__ == '__main__':
    main()
```

Example

To run this example enter the following:

```
>>> from msl.examples.qt import LoopExampleSleep
>>> loop = LoopExampleSleep()
>>> loop.start()
```

class msl.qt.sleep.**SleepWorker**(*seconds*)

Bases: *msl.qt.threading.Worker*

The *Worker* class for *Sleep*.

Delays execution for a certain number of seconds without freezing the graphical user interface.

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

process()

Calls time.sleep().

msl.qt.threading module

Base classes for starting a process in a new QThread.

msl.qt.threading.**Signal**(**args*, ***kwargs*)

class msl.qt.threading.**Worker**

Bases: QObject

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*.

Example

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

process()

The expensive or blocking operation to process.

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

class msl.qt.threading.**Thread**(*worker*)

Bases: QObject

Moves the *worker* to a new QtCore.QThread.

Parameters **worker** – A *Worker* subclass that has **NOT** been instantiated.

Example

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

__getattr__(*item*)
All other attributes are assumed to be those of the *Worker*.

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.

Parameters

- **exception** (BaseException) – The exception instance
- **traceback** (traceback) – A traceback object.

finished = None
This Signal is emitted when the thread finishes (i.e., when *Worker.process()* finishes).

is_finished()
Whether the thread successfully finished.

Returns bool – True if the thread finished otherwise False.

is_running()
Whether the thread is running.

Returns bool – True if the thread is running otherwise False.

quit()
Tells the thread's event loop to exit.

start(**args*, ***kwargs*)
Start processing the *Worker* in a QtCore.QThread.

The *args and **kwargs are passed to the constructor of the *Worker* class.

stop(*milliseconds=None*)
Calls *quit()* then *wait()*.

wait(*milliseconds=None*)
Wait for the thread to either finish or timeout.

Parameters **milliseconds** (int, optional) – The number of milliseconds to wait before a timeout occurs. If None then this method will never timeout and the thread must return from its *run* method.

Returns bool or None – True if the thread finished, False if the thread timed out or None if the thread is not running.

msl.qt.utils module

General functions.

msl.qt.utils.**drag_drop_paths**(*event*, *, *pattern=None*)
Returns the list of file paths from a drag-enter or drop event.

Parameters

- **event** (QtGui.QDragEnterEvent or QtGui.QDropEvent) – A drag-enter or drop event.
- **pattern** (str, optional) – Include only the file paths that match the *pattern*. For example, to only include JPEG or JPG image files use '*.jp*g'.

  See fnmatch.fnmatch().

Returns list of str – The list of file paths.

msl.qt.utils.**screen_geometry**(*widget=None*)
Get the geometry of a desktop screen.

Parameters **widget** (QtWidgets.QWidget, optional) – Get the geometry of the screen that contains this widget.

Returns QtCore.QRect – If a *widget* is specified then the geometry of the screen that contains the *widget* otherwise returns the geometry of the primary screen (i.e., the screen where the main widget resides).

msl.qt.widgets package

Custom QtWidgets.QWidget.

Submodules

msl.qt.widgets.button module

A QToolButton to display text, an icon and a menu.

class msl.qt.widgets.button.**Button**(*, *text=None*, *icon=None*, *icon_size=None*, *left_click=None*, *right_click=None*, *is_text_under_icon=True*, *tooltip=None*, *parent=None*)
Bases: QToolButton

A QToolButton to display text, an icon and a menu.

Parameters

- **text** (str, optional) – The text to display on the button.
- **icon** (object, optional) – Any icon object that is supported by *to_qicon()*.
- **icon_size** (int, float, tuple of int or QtCore.QSize, optional) – Rescale the icon to the specified *size*. If the value is None then do not rescale the icon. If an int then set the width and the height to be the *size* value. If a float then a scaling factor. If a tuple then the (width, height) values.
- **left_click** (callable, optional) – The function to be called for a mouse left-click event.

- **right_click** (callable, optional) – The function to be called for a mouse right-click event.
- **is_text_under_icon** (bool, optional) – If displaying an icon and text then whether to place the text under, True, or beside, False, the icon.
- **tooltip** (str, optional) – The tooltip to display for the button.
- **parent** (QtWidgets.QWidget, optional) – The parent widget.

add_menu_item(*, *text=None*, *triggered=None*, *icon=None*, *shortcut=None*, *tooltip=None*)
Add a new item to the action menu.

Parameters

- **text** (str, optional) – The text to display for this item.
- **triggered** (callable, optional) – The function to be called when this item is selected. If None then the item is displayed but it is disabled.
- **icon** (object, optional) – Any icon object that is supported by *to_qicon()*.
- **shortcut** (str, optional) – The keyboard shortcut to use to select this item, e.g., 'CTRL+A'
- **tooltip** (str, optional) – The tooltip to display for this item.

add_menu_separator()
Insert a separator between menu items.

set_left_click(*fcn*)
The function to be called for a mouse left-click event.

Parameters **fcn** (callable) – The function to be called for a mouse left-click event.

set_right_click(*fcn*)
The function to be called for a mouse right-click event.

Parameters **fcn** (callable) – The function to be called for a mouse right-click event.

class msl.qt.widgets.button.**ButtonMenu**
Bases: QMenu

Display the QtWidgets.QMenu underneath the *Button*.

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

msl.qt.widgets.led module

An *LED* widget.

class msl.qt.widgets.led.**LED**(*, *parent=None*, *shape=<Shapes.Circle: 0>*, *on_color='#0F6900'*, *off_color='grey'*, *clickable=False*, *tooltip=None*)
Bases: QWidget

An LED widget,

This class is based off of the QLed *project*.

Parameters

- **parent** (QtWidgets.QWidget) – The parent widget.
- **shape** (int, str or *Shapes*) – The shape to draw the *LED*. If a str then the name of one of the *Shapes*, e.g. 0, 'circle' and *Shapes.Circle* are equivalent.
- **on_color** – The color when the *LED* is on. See *to_qcolor()* for details about the different data types that are supported.
- **off_color** – The color when the *LED* is off. See *to_qcolor()* for details about the different data types that are supported.
- **clickable** (bool) – Whether the state of the *LED* can be changed by clicking on it.
- **tooltip** (str, optional) – The tooltip to use for the *LED*.

Example

To view an example with the *LED* run:

```
>>> from msl.examples.qt import led
>>> led.show()
```

Circle = 0

Rounded = 1

Square = 2

Triangle = 3

clickable()
Get if the on/off state of the *LED* can be changed by clicking on it.

Returns bool – Whether the state of the *LED* can be changed by clicking on it.

clicked = None
Emitted when the LED is clicked (even if it is not *clickable*).

is_on
Whether the *LED* is on or off.

Type bool

off_color()
Get the color of the *LED* when it is off.

Returns QtGui.QColor – The off color.

on_color()
Get the color of the *LED* when it is on.

Returns QtGui.QColor – The on color.

set_clickable(*clickable*)
Set if the on/off state of the *LED* can be changed by clicking on it.

Parameters **clickable** (bool) – Whether the state of the *LED* can be changed by clicking on it.

set_off_color(*color*)
Set the color of the *LED* when it is off.

Parameters **color** – The color when the *LED* is off. See *to_qcolor()* for details about the different data types that are supported.

set_on_color(*color*)
Set the color of the *LED* when it is on.

Parameters **color** – The color when the *LED* is on. See *to_qcolor()* for details about the different data types that are supported.

set_shape(*shape*)
Set the shape of the *LED*.

Parameters **shape** (int, str or *Shapes*) – The shape to draw the *LED*. If a str then the name of one of the *Shapes*, e.g. 0, 'circle' and *Shapes.Circle* are equivalent.

shape()
Get the shape of the *LED*.

Returns *Shapes* – The shape of the *LED*.

toggle()
Toggle the on/off state of the *LED*.

toggled = None
Emitted when the LED turns on → off or off → on.

turn_off()
Turn the *LED* off.

turn_on()
Turn the *LED* on.

class msl.qt.widgets.led.**Shapes**
Bases: enum.IntEnum

Shapes that are available to draw the *LED*.

Circle = 0

Rounded = 1

Square = 2

Triangle = 3

msl.qt.widgets.led.**Signal**(**args*, ***kwargs*)

msl.qt.widgets.logger module

A QWidget to display logging messages.

class msl.qt.widgets.logger.**Logger**(*, *level=20*, *fmt='%(asctime)s [%(levelname)s] – %(name)s – %(message)s'*, *datefmt=None*, *logger=None*, *parent=None*)

Bases: logging.Handler, QWidget

A QWidget to display logging messages.

Parameters

- **level** (int or str, optional) – The Logging Level to use to display the LogRecord.
- **fmt** (str, optional) – The LogRecord attributes to use to display the LogRecord.
- **datefmt** (str or None, optional) – The strftime format to use for the time stamp. If None then the ISO8601 date format is used, YYYY-mm-dd HH:MM:SS,sss.
- **logger** (logging.Logger, optional) – The logging.Logger to add this logging.Handler to. If None the uses the root logging.Logger.
- **parent** (QtWidgets.QWidget, optional) – The parent widget.

Example

To view an example of the *Logger* widget run:

```
>>> from msl.examples.qt import logger
>>> logger.show()
```

records
The history of all the log records.

Type list of LogRecord

save(*path*, *, *level=20*)
Save log records to a file.

Parameters

- **path** (str) – The path to save the log records to. Appends the records to the file if the file already exists, otherwise creates a new log file. It is recommended that the file extension be .log, but not mandatory.
- **level** (int, optional) – All LogRecord's with a Logging Level >= *level* will be saved.

msl.qt.widgets.comments module

A QtWidgets.QDialog to prompt the user to enter comments.

class msl.qt.widgets.comments.**Comments**(*json_path*, *title*, *even_row_color*, *odd_row_color*)

Bases: QDialog

A QtWidgets.QDialog to prompt the user to enter comments.

Do not instantiate directly. Use `msl.qt.prompt.comments()` instead.

append_to_history_table(*timestamp*, *comment*)

apply_filter()

clear_filter()

clear_history()

create_widgets()

load_json()

prepend_and_close()

save_json()

table_double_click(*row*, *col*)

table_key_press(*event*)

text()
str: The text in the comment editor

update_table_row_colors_and_resize()

msl.qt.widgets.spinboxes module

Custom `QtWidgets.QDoubleSpinBox` and `QtWidgets.QSpinBox` classes.

In situations where the value of the spinbox is used to represent the position/value of equipment in the laboratory one typically does not want to connect the `valueChanged()` signal of the spinbox to change the value of the equipment because each numeric key press would be sent to the equipment. For example, if you wanted to set the position/value of the equipment to 1234 then typing the value 1234 in the spinbox would send:

- 1 → set the value of the equipment to be 1
- 12 → set the value of the equipment to be 12
- 123 → set the value of the equipment to be 123
- 1234 → set the value of the equipment to be 1234

These custom `QAbstractSpinBox` subclasses will emit the `editingFinished()` signal when any of the following events occur:

- the spinbox loses focus
- the Enter key is pressed
- the Up, Down, PageUp or PageDown keys are pressed
- the Increment and Decrement buttons are clicked

class msl.qt.widgets.spinboxes.**DoubleSpinBox**(*, *parent=None*, *value=0*, *minimum=0*, *maximum=100*, *step=1*, *decimals=2*, *use_si_prefix=False*, *unit=''*, *tooltip=''*)

Bases: `QDoubleSpinBox`

A QDoubleSpinBox that emits editingFinished() after a stepBy() signal.

Parameters

- **parent** (QtWidgets.QWidget, optional) – The parent widget.
- **value** (float, optional) – The initial value.
- **minimum** (float, optional) – The minimum value.
- **maximum** (float, optional) – The maximum value.
- **step** (float, optional) – The step-by size.
- **decimals** (int, optional) – The number of digits after the decimal place to use to show the value.
- **use_si_prefix** (bool, optional) – Whether to use an SI prefix to represent the number, e.g. a value of 1.2e-9 would be represented as '1.2 n'
- **unit** (str or enum.Enum, optional) – The text to display after the value.
- **tooltip** (str, optional) – The tooltip to use for the *DoubleSpinBox*.

fixup(*text*)
Overrides QtWidgets.QAbstractSpinBox.fixup().

setValue(*value*)
Overrides QtWidgets.QDoubleSpinBox.setValue().

textFromValue(*value*)
Overrides QtWidgets.QDoubleSpinBox.textFromValue().

validate(*text*, *position*)
Overrides QtWidgets.QAbstractSpinBox.validate().

valueFromText(*text*)
Overrides QtWidgets.QDoubleSpinBox.valueFromText().

class msl.qt.widgets.spinboxes.**SpinBox**(*, *parent=None*, *value=0*, *minimum=0*, *maximum=100*, *step=1*, *unit=''*, *tooltip=''*)

Bases: QSpinBox

A QSpinBox that emits editingFinished() after a stepBy() signal.

Parameters

- **parent** (QtWidgets.QWidget, optional) – The parent widget.
- **value** (int, optional) – The initial value.
- **minimum** (int, optional) – The minimum value.
- **maximum** (int, optional) – The maximum value.
- **step** (int, optional) – The step-by size.
- **unit** (str or enum.Enum, optional) – The text to display after the value.
- **tooltip** (str, optional) – The tooltip to use for the *SpinBox*.

msl.qt.widgets.toggle_switch module

A toggle switch QtWidgets.QWidget.

class msl.qt.widgets.toggle_switch.**ToggleSwitch**(*, *parent=None*, *height=None*, *on_color='#009688'*, *off_color='#B4B4B4'*, *tooltip=None*)

Bases: QAbstractButton

Constructs a toggle switch,

Parameters

- **parent** (QtWidgets.QWidget, optional) – The parent widget.
- **height** (int, optional) – The height, in pixels, of the toggle switch.
- **on_color** – The color when the *ToggleSwitch* is on. See *to_qcolor()* for details about the different data types that are supported.
- **off_color** – The color when the *ToggleSwitch* is off. See *to_qcolor()* for details about the different data types that are supported.
- **tooltip** (str, optional) – The tooltip to use for the *ToggleSwitch*.

Example

To view an example with the *ToggleSwitch* run:

```
>>> from msl.examples.qt import toggle_switch
>>> toggle_switch.show()
```

is_on
Whether the *ToggleSwitch* is on or off.

Type bool

off_color()
Get the color of the *ToggleSwitch* when it is off.

Returns QtGui.QColor – The off color.

on_color()
Get the color of the *ToggleSwitch* when it is on.

Returns QtGui.QColor – The on color.

set_off_color(*color*)
Set the color of the *ToggleSwitch* when it is off.

Parameters **color** – The color when the *ToggleSwitch* is off. See *to_qcolor()* for details about the different data types that are supported.

set_on_color(*color*)
Set the color of the *ToggleSwitch* when it is on.

Parameters **color** – The color when the *ToggleSwitch* is on. See *to_qcolor()* for details about the different data types that are supported.

turn_off()
Turn the *ToggleSwitch* off.

turn_on()
Turn the *ToggleSwitch* on.

1.3 License

```
MIT License

Copyright (c) 2017 - 2020, Measurement Standards Laboratory of New Zealand

Permission is hereby granted, free of charge, to any person obtaining a␣
↪copy
of this software and associated documentation files (the "Software"), to␣
↪deal
in the Software without restriction, including without limitation the␣
↪rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in␣
↪all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING␣
↪FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN␣
↪THE
SOFTWARE.
```

1.4 Developers

- Joseph Borbely <joseph.borbely@measurement.govt.nz>
- Rebecca Hawke <rebecca.hawke@measurement.govt.nz>

1.5 Release Notes

1.5.1 Version 0.1.0

CHAPTER 2

Indices and tables

- genindex
- modindex

Python Module Index

m

Index

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z