Qarbon Documentation

Release 0.1.0

Qarbon team

April 29, 2014

Contents

Welcome to qarbon 0.1.0 documentation

CHAPTER 1

Overview

qarbon is ...

CHAPTER 2

FAQ

CHAPTER 3

Screenshots

CHAPTER 4

Examples

CHAPTER 5

API

5.1 qarbon.color

Helper functions to translate state to color.

Functions

getBgColorFromState	Returns the background color for the given state:
getCSSColorFromState	Returns a CSS string representing the color for the given state.
getColorFromState	Returns the background a foreground color for the given state:
getFgColorFromState	Returns the foreground color for the given state:
getStateColorMap	Returns the map used for color states.

qarbon.color.**getStateColorMap**()
Returns the map used for color states.

dict<State, tuple<bg color(tuple<R (int), G (int), B (int)>, A (int)>), fg color(tuple<R (int), G (int), B (int), A (int)>)>>

Returns map of the state colors

Return type dict

qarbon.color.**getColorFromState**(*state*)
Returns the background a foreground color for the given state:

tuple<bg color(tuple<R (int), G (int), B (int)>, A (int)>), fg color(tuple<R (int), G (int), B (int), A (int)>)> :return: background a foreground color for the given state :rtype: tuple

qarbon.color.**getCSSColorFromState**(*state*)
Returns a CSS string representing the color for the given state.

Returns a CSS string representing the color for the given state

Return type str

qarbon.color.**getBgColorFromState**(*state*)
Returns the background color for the given state: tuple<R (int), G (int), B (int)>, A (int)>

Returns the background color for the given state

Return type tuple

qarbon.color.**getFgColorFromState**(*state*)
Returns the foreground color for the given state: tuple<R (int), G (int), B (int)>, A (int)>

Returns the foreground color for the given state

Return type tuple

5.2 qarbon.config

Global configuration.

qarbon.config.**NAMESPACE = 'qarbon'**
qarbon namespace

qarbon.config.**DEFAULT_QT_AUTO_INIT = True**
Auto initialize Qt

qarbon.config.**DEFAULT_QT_AUTO_API = 'PyQt4'**
Set preffered API if not is already loaded

qarbon.config.**DEFAULT_QT_AUTO_STRICT = False**
Whether or not should be strict in choosing Qt API

qarbon.config.**DEFAULT_QT_AUTO_INIT_LOG = True**
Auto initialize Qt logging to python logging

qarbon.config.**DEFAULT_QT_AUTO_INIT_RES = True**
Auto initialize Qarbon resources (icons)

qarbon.config.**DEFAULT_QT_AUTO_REMOVE_INPUTHOOK = True**
Remove input hook (only valid for PyQt4)

qarbon.config.**QT_AUTO_INIT = False**
Auto initialize Qt

qarbon.config.**QT_AUTO_API = 'PyQt4'**
Set preffered API if not is already loaded

qarbon.config.**QT_AUTO_STRICT = False**
Whether or not should be strict in choosing Qt API

qarbon.config.**QT_AUTO_INIT_LOG = True**
Auto initialize Qt logging to python logging

qarbon.config.**QT_AUTO_INIT_RES = True**
Auto initialize Qarbon resources (icons)

qarbon.config.**QT_AUTO_REMOVE_INPUTHOOK = True**
Remove input hook (only valid for PyQt4)

5.3 qarbon.core

Model core module.

Functions

Database	Helper to get the database corresponding to the given name.
Device	Helper to get the device corresponding to the given name.
Attribute	Helper to get the attribute corresponding to the given name.

Classes

Continued on next page

Table 5.3 – continued from previous page

Quality	Quality enum.
Access	Access enum.
DisplayLevel	Display level enum.
DataType	Data type enum.
State	State enum.
IScheme	Base scheme class.
IDatabase	Base database class.
IDevice	Base device class.
IAttribute	Base attribute class.

class qarbon.core.**Quality**
Bases: qarbon.external.enum._enum.Enum

Quality enum.

class qarbon.core.**Access**
Bases: qarbon.external.enum._enum.Enum

Access enum.

class qarbon.core.**DisplayLevel**
Bases: qarbon.external.enum._enum.Enum

Display level enum.

class qarbon.core.**DataType**
Bases: qarbon.external.enum._enum.Enum

Data type enum.

static to_python_type(*dtype*)
Convert from DataType to python type

static to_data_type(*dtype*)
Convert from type to DataType

class qarbon.core.**State**
Bases: qarbon.external.enum._enum.Enum

State enum.

qarbon.core.**Manager**()
Returns the one and only core Manager.

class qarbon.core.**IScheme**(*name=None*, *parent=None*)
Bases: qarbon.node.Node

Base scheme class.

Plugins should provide an implementation of this class.

schemes = ()

get_database(*name*)

get_device(*name*)

get_attribute(*name*)

class qarbon.core.**IDatabase**(*name*, *parent=None*)
Bases: qarbon.node.Node

Base database class.

Plugins should provide an implementation of this class as a response to a get_database from their Scheme

get_device(*object_name*)

class qarbon.core.**IDevice**(*name*, *parent=None*)
Bases: qarbon.node.Node

Base device class.

Plugins should provide an implementation of this class as a response to a get_device from their Scheme

database

get_attribute(*attr_name*)

execute(*cmd*, **args*, ***kwargs*)

class qarbon.core.**IAttribute**(*name*, *parent=None*)
Bases: qarbon.node.Node

Base attribute class.

Plugins should provide an implementation of this class as a response to a get_attribute from their Scheme

device

read()

write(*value*)

qarbon.core.**Database**(*name=None*)
Helper to get the database corresponding to the given name.

qarbon.core.**Device**(*name*)
Helper to get the device corresponding to the given name.

qarbon.core.**Attribute**(*name*)
Helper to get the attribute corresponding to the given name.

5.4 qarbon.executor

5.5 qarbon.log

Helper logging functions.

Functions

log	
debug	
info	
warn	
warning	
error	
exception	
critical	
fatal	
initialize	Initializes logging.
is_initialized	
log_it	partial(func, *args, **keywords) - new function with partial application
debug_it	partial(func, *args, **keywords) - new function with partial application
info_it	partial(func, *args, **keywords) - new function with partial application
warn_it	partial(func, *args, **keywords) - new function with partial application
error_it	partial(func, *args, **keywords) - new function with partial application
fatal_it	partial(func, *args, **keywords) - new function with partial application

qarbon.log.**log**(*level*, *msg*, **args*, ***kwargs*)

qarbon.log.**debug**(*msg*, **args*, ***kwargs*)

qarbon.log.**info**(*msg*, **args*, ***kwargs*)

qarbon.log.**warn**(*msg*, **args*, ***kwargs*)

qarbon.log.**warning**(*msg*, **args*, ***kwargs*)

qarbon.log.**error**(*msg*, **args*, ***kwargs*)

qarbon.log.**exception**(*msg*, **args*, ***kwargs*)

qarbon.log.**fatal**(*msg*, **args*, ***kwargs*)

qarbon.log.**critical**(*msg*, **args*, ***kwargs*)

qarbon.log.**is_initialized**()

qarbon.log.**initialize**(*log_level=None*, *log_format=None*, *stream=None*, *file_name=None*, *file_size=None*, *file_number=None*)

Initializes logging. Configures the Root logger with the given log_level. If file_name is given, a rotating log file handler is added. Otherwise, adds a default output stream handler with the given log_format.

5.6 qarbon.node

Node module.

Classes

Node	Node class representing a node in a tree.

class qarbon.node.**Node**(*name*, *parent=None*)

Bases: object

Node class representing a node in a tree.

A strong reference is kept on the parent node. Weak references are kept on the childs.

name

get_parent()

get_children()

get_child(*name*)

has_child(*name*)

add_child(*name*, *child*)

5.7 qarbon.plugin

Plugin extension manager.

Functions

get_plugins	

Continued on next page

Table 5.6 – continued from previous page

get_plugin_info	
IPlugin	Decorator that transforms the decorated class into a plugin point.

qarbon.plugin.**get_plugins**()

qarbon.plugin.**get_plugin_info**(*plugin*)

qarbon.plugin.**IPlugin**(*klass=None*, ***kwargs*)

Decorator that transforms the decorated class into a plugin point.

5.8 qarbon.qt.gui.action

Helper functions to access QAction.

Example:

```
from qarbon.external.qt import QtGui
from qarbon.qt.gui.application import Application
from qarbon.qt.gui.action import Action
from qarbon.qt.gui.icon import Icon

def onImageFileOpen():
    fileName = QtGui.QFileDialog.getOpenFileName(None,
        "Open Image", "/home/homer",
        "Image Files (*.png *.jpg *.bmp)")
    print (fileName)

app = Application()
window = QtGui.QMainWindow()
openImageAction = Action("Open &image...", parent=window,
                         icon=Icon("folder-open"),
                         shortcut=QtGui.QKeySequence.Open,
                         tooltip="open an existing image file",
                         triggered=onImageFileOpen)

menuBar = window.menuBar()
fileMenu = menuBar.addMenu("&File")
fileMenu.addAction(openImageAction)
window.show()
app.exec_()
```

Functions

Action	Create a new QAction.

qarbon.qt.gui.action.**Action**(*text*, *parent=None*, *shortcut=None*, *icon=None*, *tooltip=None*, *toggled=None*, *triggered=None*, *data=None*, *context=1*)

Create a new QAction.

Example:

```
from qarbon.external.qt import QtGui
from qarbon.qt.gui.application import Application
from qarbon.qt.gui.action import Action
from qarbon.qt.gui.icon import Icon

def onImageFileOpen():
    fileName = QtGui.QFileDialog.getOpenFileName(None,
```

```
            "Open Image", "/home/homer",
            "Image Files (*.png *.jpg *.bmp)")
        print (fileName)

    app = Application()
    window = QtGui.QMainWindow()
    openImageAction = Action("Open &image...", parent=window,
                             icon=Icon("folder-open"),
                             shortcut=QtGui.QKeySequence.Open,
                             tooltip="open an existing image file",
                             triggered=onImageFileOpen)

    menuBar = window.menuBar()
    fileMenu = menuBar.addMenu("&File")
    fileMenu.addAction(openImageAction)
    window.show()
    app.exec_()
```

Parameters

- **text** (*str*) – label for the action
- **parent** (*QObject*) – parent QObject
- **shortcut** – optional shortcut
- **icon** (*QIcon or str*) – optional icon. Can be a QIcon or a string
- **tooltip** (*str*) – optional tool tip
- **toggled** (*callable*) – optional toggled slot
- **data** (*object*) – optional data
- **context** (*ShortcutContext*) – action context

Returns a customized QAction

Return type QAction

5.9 qarbon.qt.gui.application

Helper functions to manage QApplication.

Most common use case:

```
from qarbon.external.qt import QtGui
from qarbon.qt.gui.application import Application

app = Application()
label = QtGui.QLabel("Hello, world!")
label.show()
app.exec_()
```

The advantage here is you can call `Application()` anywhere on your program.

Functions

`Application`	Returns a QApplication.

qarbon.qt.gui.application.**Application**(*argv=None*, ***properties*)

Returns a QApplication.

If the process has initialized before a QApplication it returns the existing instance, otherwise it creates a new one.

When a QApplication is created it takes argv into account. If argv is None (default), it take arguments from `sys.argv`.

If argv is given and a QApplication already exists, argv will have no effect.

Parameters **argv** – optional arguments to QApplication. If the QApplication is already initialized, argv will have no effect.

Example:

```
from qarbon.external.qt import QtGui
from qarbon.qt.gui.application import Application

app = Application()
label = QtGui.QLabel("Hello, world!")
label.show()
app.exec_()
```

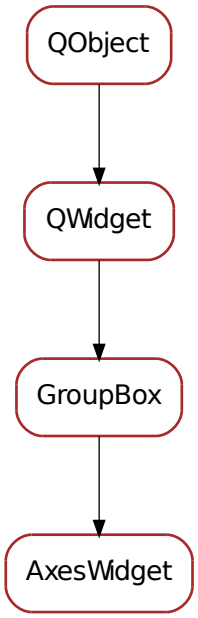
Parameters **properties** – currently unused

Returns the QApplication

Return type QtGui.QApplication

5.10 qarbon.qt.gui.axeswidget

Multiple axis (axes) widget.



Classes

AxesWidget	A multiple axis widget.

class qarbon.qt.gui.axeswidget.**Axis**(*axis_info*, *axes*, *parent=None*)

Bases: PyQt4.QtCore.QObject

positionChanged = <PyQt4.QtCore.pyqtSignal object at 0x3181510>

limitsChanged = <PyQt4.QtCore.pyqtSignal object at 0x3181550>

stepsChanged = <PyQt4.QtCore.pyqtSignal object at 0x3181590>

currentStepChanged = <PyQt4.QtCore.pyqtSignal object at 0x31815d0>

stateChanged = <PyQt4.QtCore.pyqtSignal object at 0x3181610>

labelChanged = <PyQt4.QtCore.pyqtSignal object at 0x3181650>

unitChanged = <PyQt4.QtCore.pyqtSignal object at 0x3181690>

axes

refresh()

getPosition(*cache=True*)

setPosition(*position*, *emit=True*)

position = <PyQt4.QtCore.pyqtProperty object at 0x3181790>

This property contains the axis position

Access functions:

- Axis.getPosition()
- Axis.setPosition()

getLimits(*cache=True*)

setLimits(*limits*, *emit=True*)

limits = <PyQt4.QtCore.pyqtProperty object at 0x3181850>

This property contains the axis limits

Access functions:

- Axis.getLimits()
- Axis.setLimits()

getState(*cache=True*)

setState(*state*, *emit=True*)

state = <PyQt4.QtCore.pyqtProperty object at 0x3181910>

This property contains the axis state

Access functions:

- Axis.getState()
- Axis.setState()

getLabel()

setLabel(*label*, *emit=True*)

label = <PyQt4.QtCore.pyqtProperty object at 0x3181990>

This property contains the axis label

Access functions:

- Axis.getLabel()
- Axis.setLabel()

getSteps()

setSteps(*steps*, *emit=True*)

steps = <PyQt4.QtCore.pyqtProperty object at 0x3181a10>
This property contains the axis steps

Access functions:

- Axis.getSteps()
- Axis.setSteps()

getCurrentStep()

setCurrentStep(*current_step*, *emit=True*)

currentStep = <PyQt4.QtCore.pyqtProperty object at 0x3181a90>
This property contains the axis current step size

Access functions:

- Axis.getCurrentStep()
- Axis.setCurrentStep()

getUnit()

setUnit(*unit*, *emit=True*)

unit = <PyQt4.QtCore.pyqtProperty object at 0x3181b10>
This property contains the axis unit

Access functions:

- Axis.getUnit()
- Axis.setUnit()

move(*absolute_position*)

moveRelative(*relative_position*)

moveUp()

moveDown()

stepUp()

stepDown()

stop()

ToolTipTemplate = '<html>axis <u>{axis.label}</u> is in {axis.state.name} state, at position {axis.pos

toolTip()

class qarbon.qt.gui.axeswidget.**AxesWidget**(*title=None*, *axes=None*, *parent=None*)
Bases: qarbon.qt.gui.groupbox.GroupBox

A multiple axis widget.

DefaultUpdateStatusBar = True

axes()

setAxes(*axes*)

addAxis(*axis*)

removeAxisID(*axis_id*)

removeAxis(*axis*)

getAxis(*name*)

getAxisByRole(*role*)

axisColumnWidget(*axis*, *role*)

axisIDColumnWidget(*name*, *role*)

setAxisColumnVisible(*axis*, *role*, *show=True*)

setAxisIDColumnVisible(*name*, *role*, *show=True*)

setColumnVisible(*role*, *show=True*)

refreshAxes()

onUserPositionApplied()

onUserPositionChanged(*value*)

onUserCurrentStepsChanged(*index*)

onUserStepLeft()

onUserStepRight()

onUserStop()

onAxisPositionChanged(*name*, *position*)

onAxisStateChanged(*name*, *old_state*, *state*)

onAxisLabelChanged(*name*, *label*)

onAxisStepsChanged(*name*, *steps*)

onAxisCurrentStepChanged(*name*, *step*)
Steps changed from the Axis model:

- change the current step value on the combo box
- change the step size on the position spin box
- update (enable/disable the stepLeft and stepRight buttons according to the current axis limits)

onAxisLimitsChanged(*name*, *limits*)

onAxisUnitChanged(*name*, *unit*)

setUpdateStatusBar(*update*)

getUdpateStatusBar()

resetUpdateStatusBar()

classmethod getQtDesignerPluginInfo()

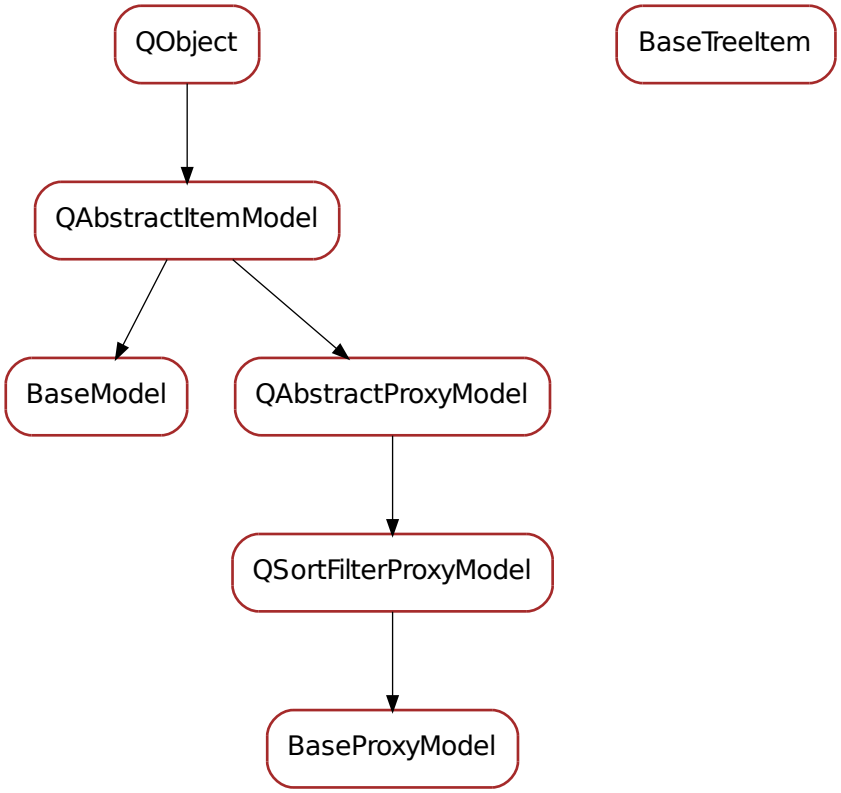
updateStatusBar = <PyQt4.QtCore.pyqtProperty object at 0x3181cd0>
This property sets if the widget should update stauts bar with messages

Access functions:

- AxesWidget.getUdpateStatusBar()
- AxesWidget.setUpdateStatusBar()
- AxesWidget.resetUpdateStatusBar()

5.11 qarbon.qt.gui.basemodel

A base model and a base tree item.



Classes

BaseTreeItem	A generic node
BaseModel	The base class for all Qt models.
BaseProxyModel	A base Qt filter & sort model

class qarbon.qt.gui.basemodel.**BaseTreeItem**(*model*, *data*, *parent=None*)

Bases: object

A generic node

DisplayFunc

alias of str

itemData()

The internal itemData object

Returns (object) object holding the data of this item

depth()

Depth of the node in the hierarchy

Returns (int) the node depth

appendChild(*child*)
Adds a new child node

Parameters **child** – (BaseTreeItem) child to be added

child(*row*)
Returns the child in the given row

Returns (BaseTreeItem) the child node for the given row

childCount()
Returns the number of childs for this node

Returns (int) number of childs for this node

hasChildren()

data(*index*)
Returns the data of this node for the given index

Returns (object) the data for the given index

icon(*index*)

toolTip(*index*)

setData(*index*, *data*)
Sets the node data

Parameters **data** – (object) the data to be associated with this node

parent()
Returns the parent node or None if no parent exists

Returns (BaseTreeItem) the parent node

row()
Returns the row for this node

Returns (int) row number for this node

display()
Returns the display string for this node

Returns (str) the node's display string

mimeData(*index*)

role()
Returns the prefered role for the item. This implementation returns string *Unknown*

This method should be able to return any kind of python object as long as the model that is used is compatible.

Returns the role in form of element type

class qarbon.qt.gui.basemodel.**BaseModel**(*parent=None*, *data=None*)
Bases: PyQt4.QtCore.QAbstractItemModel

The base class for all Qt models.

ColumnNames = ()

ColumnRoles = ((),)

DftFont = <PyQt4.QtGui.QFont object at 0x3126d90>

createNewRootItem()

refresh(*refresh_source=False*)

setupModelData(*data*)

roleIcon(*role*)

roleSize(*role*)

roleToolTip(*role*)

setDataSource(*data_src*)

dataSource()

setSelectables(*seq_elem_types*)

selectables()

role(*column*, *depth=0*)

columnCount(*parent=<PyQt4.QtCore.QModelIndex object at 0x3126e50>*)

columnIcon(*column*)

columnToolTip(*column*)

columnSize(*column*)

pyData(*index*, *role=0*)

data(*index*, *role=0*)

flags(*index*)

headerData(*section*, *orientation*, *role=0*)

index(*row*, *column*, *parent=<PyQt4.QtCore.QModelIndex object at 0x3126fd0>*)

parent(*index*)

rowCount(*parent=<PyQt4.QtCore.QModelIndex object at 0x3128090>*)

hasChildren(*parent=<PyQt4.QtCore.QModelIndex object at 0x3128110>*)

class qarbon.qt.gui.basemodel.**BaseProxyModel**(*parent=None*)
Bases: PyQt4.QtGui.QSortFilterProxyModel

A base Qt filter & sort model

5.12 qarbon.qt.gui.basetree

A base tree widget and toolbar.

Classes

BaseTreeWidget	A pure Qt tree widget implementing a tree with a navigation toolbar

class qarbon.qt.gui.basetree.**BaseTreeWidget**(*parent=None*, *with_navigation_bar=True*, *with_filter_widget=True*, *with_selection_widget=True*, *with_refresh_widget=True*, *perspective=None*, *proxy=None*)
Bases: qarbon.qt.gui.baseview.BaseModelWidget

A pure Qt tree widget implementing a tree with a navigation toolbar

createToolArea()

createViewWidget(*klass=None*)

treeView()

goIntoAction()

goTopAction()

goUpAction()

expandAllTree()

onExpanded()

collapseAllTree()

expandSelectionTree()

collapseSelectionTree()

resizeColumns()

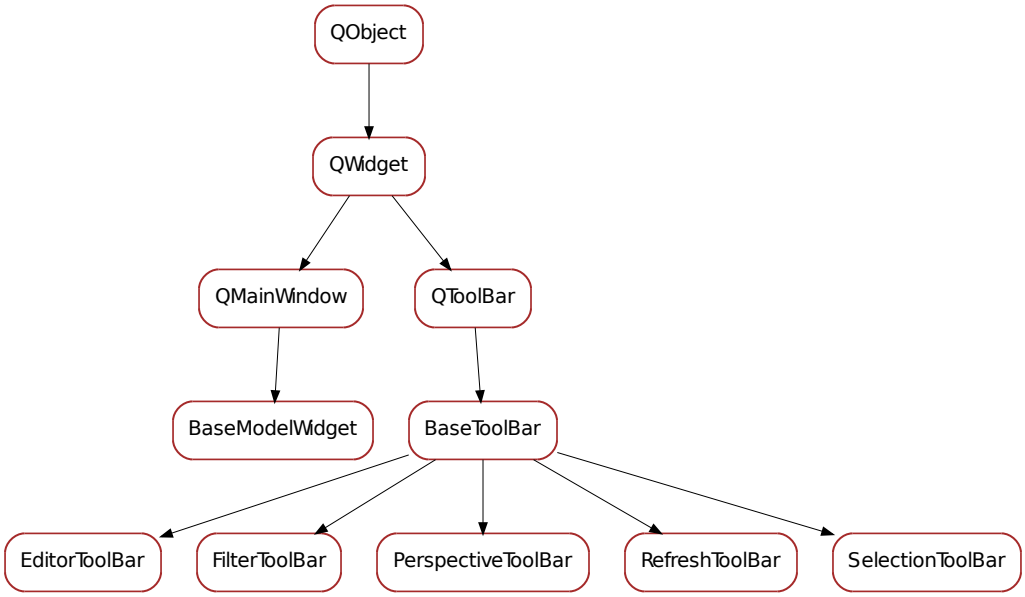
goIntoTree()

goUpTree()

goTopTree()

5.13 qarbon.qt.gui.baseview

A base view widget and toolbar.



Classes

BaseModelWidget	A pure Qt widget designed to display a Qt view widget (QTreeView for example), envolved by optio
BaseToolBar	
FilterToolBar	Internal widget providing quick filter to be placed in a _QToolArea
EditorToolBar	Internal widget to be placed in a _QToolArea providing buttons for
SelectionToolBar	

Table 5.12 – continued from previous page

RefreshToolBar
PerspectiveToolBar

class qarbon.qt.gui.baseview.**BaseToolBar**(*name=None*, *view=None*, *parent=None*, *designMode=False*)

Bases: PyQt4.QtGui.QToolBar

viewWidget()

class qarbon.qt.gui.baseview.**FilterToolBar**(*view=None*, *parent=None*, *designMode=False*)

Bases: qarbon.qt.gui.baseview.BaseToolBar

Internal widget providing quick filter to be placed in a _QToolArea

clearFilterTriggered = <PyQt4.QtCore.pyqtSignal object at 0x3123610>

filterChanged = <PyQt4.QtCore.pyqtSignal object at 0x3123690>

filterEdited = <PyQt4.QtCore.pyqtSignal object at 0x3123710>

getFilterLineEdit()

onClearFilter()

onFilterChanged(*text=None*)

onFilterEdited(*text=None*)

setFilterText(*text*)

class qarbon.qt.gui.baseview.**EditorToolBar**(*view=None*, *parent=None*, *designMode=False*)

Bases: qarbon.qt.gui.baseview.BaseToolBar

Internal widget to be placed in a _QToolArea providing buttons for moving, adding and removing items from a view based widget

addTriggered = <PyQt4.QtCore.pyqtSignal object at 0x3123790>

removeTriggered = <PyQt4.QtCore.pyqtSignal object at 0x3123810>

moveTopTriggered = <PyQt4.QtCore.pyqtSignal object at 0x3123850>

moveUpTriggered = <PyQt4.QtCore.pyqtSignal object at 0x3123890>

moveDownTriggered = <PyQt4.QtCore.pyqtSignal object at 0x31238d0>

moveBottomTriggered = <PyQt4.QtCore.pyqtSignal object at 0x3123910>

onAdd()

onRemove()

onMoveTop()

onMoveUp()

onMoveDown()

onMoveBottom()

class qarbon.qt.gui.baseview.**SelectionToolBar**(*view=None*, *parent=None*, *designMode=False*)

Bases: qarbon.qt.gui.baseview.BaseToolBar

selectAllTriggered = <PyQt4.QtCore.pyqtSignal object at 0x3123950>

clearSelectionTriggered = <PyQt4.QtCore.pyqtSignal object at 0x31239d0>

onSelectAll()

onclearSelection()

class qarbon.qt.gui.baseview.**RefreshToolBar**(*view=None*, *parent=None*, *designMode=False*)

Bases: qarbon.qt.gui.baseview.BaseToolBar

refreshTriggered = <PyQt4.QtCore.pyqtSignal object at 0x3123a10>

onRefresh()

class qarbon.qt.gui.baseview.**PerspectiveToolBar**(*perspective*, *view=None*, *parent=None*, *designMode=False*)

Bases: qarbon.qt.gui.baseview.BaseToolBar

perspectiveChanged = <PyQt4.QtCore.pyqtSignal object at 0x3123ad0>

switchPerspectiveButton()

Returns the QToolButton that handles the switch perspective.

Returns (PyQt4.QtGui.QToolButton) the switch perspective tool button

onSwitchPerspective()

perspective()

class qarbon.qt.gui.baseview.**BaseModelWidget**(*parent=None*, *designMode=False*, *with_filter_widget=True*, *with_selection_widget=True*, *with_refresh_widget=True*, *perspective=None*, *proxy=None*)

Bases: PyQt4.QtGui.QMainWindow

A pure Qt widget designed to display a Qt view widget (QTreeView for example), envolved by optional toolbar and statusbar. The Qt model associated with the internal Qt view widget should be a Framework4.GUI.Qt.Model.BaseModel

KnownPerspectives = {}

DftPerspective = None

itemClicked = <PyQt4.QtCore.pyqtSignal object at 0x3123b50>

itemDoubleClicked = <PyQt4.QtCore.pyqtSignal object at 0x3123b90>

itemSelectionChanged = <PyQt4.QtCore.pyqtSignal object at 0x3123bd0>

currentItemChanged = <PyQt4.QtCore.pyqtSignal object at 0x3123c10>

createViewWidget(*klass=None*)

createStatusBar()

createToolArea()

getPerspectiveBar()

getFilterBar()

getSelectionBar()

getRefreshBar()

onRefreshModel()

onSelectAll()

onClearSelection()

viewWidget()

getQModel()

getBaseQModel()

usesProxyQModel()

setQModel (*qmodel*)

viewSelectionChanged (*selected*, *deselected*)

viewCurrentIndexChanged (*current*, *previous*)

selectedItems ()
Returns a list of all selected non-hidden items

Returns (list<BaseTreeItem>)

onFilterChanged (*new_filter*)

refresh ()

perspective ()

onSwitchPerspective (*perspective*)

addToolBar (*toolbar*)

insertToolBar (*before*, *toolbar*)

5.14 qarbon.qt.gui.color

Helper functions to colors from state

Functions

getBgQColorFromState	Returns a background QColor from the given State.
getFgQColorFromState	Returns a foreground QColor from the given State.
getQColorFromState	Returns a tuple<background color (QColor), foreground color (QColor)> from the given State.
getStateColorMap	Returns the map used for color states.

qarbon.qt.gui.color.**getQColorFromState** (*state*)
Returns a tuple<background color (QColor), foreground color (QColor)> from the given State.

Parameters **state** (*State*) – the state

Returns a tuple of background a foreground color for the given state

Return type tuple<QColor, QColor>

qarbon.qt.gui.color.**getBgQColorFromState** (*state*)
Returns a background QColor from the given State.

Parameters **state** (*State*) – the state

Returns a background QColor for the given state

Return type QColor

qarbon.qt.gui.color.**getFgQColorFromState** (*state*)
Returns a foreground QColor from the given State.

Parameters **state** (*State*) – the state

Returns a foreground QColor for the given state

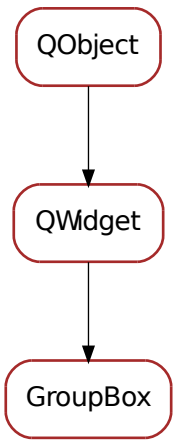
Return type QColor

5.15 qarbon.qt.gui.groupbox

A colapsable container widget with (optional) title.

Here is a simple example that shows how to create a `GroupBox` with some content inside:

```
from qarbon.external.qt import QtGui
from qarbon.qt.gui.application import Application
from qarbon.qt.gui.icon import Icon
from qarbon.qt.gui.groupbox import GroupBox

app = Application()
panel = QtGui.QWidget()
layout = QtGui.QVBoxLayout()
panel.setLayout(layout)
gb = GroupBox()
layout.addWidget(gb)

gb.title = "Configuration"
gb.icon = Icon("applications-accessories")

content = QtGui.QWidget()
gb.setContent(content)
contentLayout = QtGui.QHBoxLayout()
content.setLayout(contentLayout)
label = QtGui.QLabel("File:")
edit = QtGui.QLineEdit()
button = QtGui.QPushButton(Icon("folder-open"), "Open a file...")
contentLayout.addWidget(label)
contentLayout.addWidget(edit)
contentLayout.addWidget(button)

panel.show()
app.exec_()
```



Classes

GroupBox	An expandable/collapsible container widget

class qarbon.qt.gui.groupbox.**GroupBox**(*parent=None*)
Bases: PyQt4.QtGui.QWidget

An expandable/collapsible container widget

DefaultContentVisible = True

DefaultTitleBarVisible = True

DefaultTitleBarHeight = 16

DefaultStyle = {'content_start_color': 'rgb(224, 224, 224)', 'content_border_radius': '4px', 'title_stop_color': 'rg

content()
Returns the contents widget

Returns the current content widget or None if no content is set

Return type QWidget

setContent(*qwidget*)
Sets the content widget

Parameters **qwidget** (*QWidget*) – the content widget or None

titleBar()
Returns the title bar widget

Returns the title bar widget

Return type QFrame

titleButton()
Returns the title button widget

Returns the title button widget

Return type QToolButton

collapseButton()
Returns the collapse button widget

Returns the collapse button widget

Return type QToolButton

setTitle(*title*)
Sets this widget's title

:param title:the new widget title :type title: str

getTitle()
Returns this widget's title

Returns this widget's title

Return type str

setTitleIcon(*icon*)
Sets this widget's title icon

Parameters **icon** – (Qt.QIcon) the new widget title icon

getTitleIcon()
Returns this widget's title icon

Returns this widget's title icon

Return type QIcon

switchContentVisible()
Switches this widget's contents visibility

isContentVisible()
Returns this widget's contents visibility

Returns this widget's contents visibility

Return type bool

resetContentVisible()
Resets this widget's contents visibility

setContentVisible(*show*)
Sets this widget's contents visibility

Parameters **show** (*bool*) – the new widget contents visibility

isTitleVisible()
Returns this widget's title visibility

Returns this widget's title visibility

Return type bool

resetTitleVisible()
Resets this widget's title visibility

setTitleVisible(*show*)
Sets this widget's title visibility

Parameters **show** (*bool*) – the new widget title visibility

getTitleHeight()
Returns this widget's title height

Returns this widget's title height

Return type int

setTitleHeight(*height*)
Sets this widget's title height

Parameters **height** (*int*) – the new widget title height

resetTitleHeight()
Resets this widget's title height

getStyleMap()
Returns this widget's style

Returns this widget's style

Return type dict

setStyleMap(*style_map*)
Sets this widget's title style Used key/values for style_map:

- 'title_start_color' : brush (Ex.: '#E0E0E0')
- 'title_stop_color' : brush (Ex.: '#E0E0E0')
- 'title_font_color' : brush (Ex.: '#E0E0E0')
- 'title_border_radius': radius (Ex.: '5px')
- 'content_start_color' : brush (Ex.: '#E0E0E0')
- 'content_stop_color' : brush (Ex.: '#E0E0E0')
- 'content_border_radius': radius (Ex.: '5px')

Parameters **style_map** (*dict*) – the new widget title style

resetStyleMap()
Resets this widget's title style

classmethod getQtDesignerPluginInfo()

title = <PyQt4.QtCore.pyqtProperty object at 0x317ccd0>
This property contains the widget's title

Access functions:

- getTitle()
- setTitle()

titleIcon = <PyQt4.QtCore.pyqtProperty object at 0x317cd10>
This property contains the widget's title icon

Access functions:

- getTitleIcon()
- setTitleIcon()

titleHeight = <PyQt4.QtCore.pyqtProperty object at 0x317cd50>
This property contains the widget's title height

Access functions:

- getTitleHeight()
- setTitleHeight()
- resetTitleHeight()

titleVisible = <PyQt4.QtCore.pyqtProperty object at 0x317cd90>
This property contains the widget's title visibility

Access functions:

- isTitleVisible()
- setTitleVisible()

styleMap = <PyQt4.QtCore.pyqtProperty object at 0x317cdd0>
This property contains the widget's style map

Access functions:

- getStyleMap()
- setStyleMap()
- resetStyleMap()

contentVisible = <PyQt4.QtCore.pyqtProperty object at 0x317ce10>
This property contains the widget's content's visibility

Access functions:

- isContentVisible()
- setContentVisible()
- resetContentVisible()

5.16 qarbon.qt.gui.icon

Helper functions to handle icons and pixmaps

Most common use cases are:

```
from qarbon.external.qt import QtGui
from qarbon.qt.qui.application import Application
from qarbon.qt.gui.icon import Icon

app = Application()

# get a theme icon
icon = Icon("folder-open")

button = QtGui.QPushButton(icon, "Open file...")
button.show()
app.exec_()
```

or in a label:

```
from qarbon.external.qt import QtGui
from qarbon.qt.qui.application import Application
from qarbon.qt.gui.icon import Icon

app = Application()

# get a theme pixmap
pixmap = Pixmap("folder-open")

label = QtGui.QLabel()
label.setPixmap(pixmap)
label.show()
app.exec_()
```

Functions

Icon	Returns a QIcon for the given icon.
Pixmap	Returns a QPixmap for the given pixmap.
getIcon	Returns a QIcon for the given icon.
getPixmap	Returns a QPixmap for the given pixmap.
getQarbonIcon	Returns a QIcon for the given qarbon icon name.
getQarbonPixmap	Returns a QPixmap for the given pixmap name.
getStandardIcon	Returns a QIcon for the given icon ID (QtGui.QStyle.SP_*).
getStandardPixmap	Returns a QPixmap for the given icon ID (QtGui.QStyle.SP_*).
getStateIcon	Returns a QIcon for the given State.
getThemeIcon	Returns a QIcon for the given theme icon name.
getThemePixmap	Returns a QPixmap for the given theme pixmap name.

qarbon.qt.gui.icon.**getThemeIcon**(*icon_name*)

Returns a QIcon for the given theme icon name.

Example:

```
from qarbon.external.qt import QtGui
from qarbon.qt.qui.application import Application
from qarbon.qt.gui.icon import getThemeIcon

app = Application()
icon = getThemeIcon("folder-open")
button = QtGui.QPushButton(icon, "Open folder")
button.show()
app.exec_()
```

Parameters **icon_name** (*str*) – icon name

Returns the QIcon corresponding to the given theme name. If the theme icon doesn't exist it returns a Null icon

Return type QtGui.QIcon

qarbon.qt.gui.icon.**getThemePixmap**(*pixmap_name*, *width*, *height=None*, *mode=0*, *state=1*)

Returns a QPixmap for the given theme pixmap name.

Example:

```
from qarbon.external.qt import QtGui
from qarbon.qt.qui.application import Application
from qarbon.qt.gui.icon import getThemePixmap

app = Application()
pixmap = getThemePixmap("folder-open", 32)
label = QtGui.QLabel()
label.setPixmap(pixmap)
label.show()
app.exec_()
```

Parameters

- **pixmap_name** (*str*) – pixmap name
- **width** (*int*) – pixmap width
- **height** (*int*) – pixmap height [default: None meaning use given width]
- **mode** (*QtGui.QIcon.Mode*) – icon mode
- **state** (*QtGui.QIcon.State*) – icon state

Returns the QPixmap corresponding to the given theme name. If the theme icon doesn't exist it returns a Null pixmap

Return type QtGui.QPixmap

qarbon.qt.gui.icon.**getStandardIcon**(*icon_id*)

Returns a QIcon for the given icon ID (QtGui.QStyle.SP_*).

Example:

```
from qarbon.external.qt import QtGui
from qarbon.qt.qui.application import Application
from qarbon.qt.gui.icon import getStandardIcon

app = Application()
icon = getStandardIcon(QtGui.QStyle.SP_MessageBoxWarning)
button = QtGui.QPushButton(icon, "Open hutch")
button.show()
app.exec_()
```

Parameters **icon_id** (*QtGui.QStyle.SP*) – icon name

Returns the QIcon corresponding to the given id. If the standard ID doesn't exist it returns a Null icon

Return type QtGui.QIcon

qarbon.qt.gui.icon.**getStandardPixmap**(*pixmap_id*, *width*, *height=None*, *mode=0*, *state=1*)

Returns a QPixmap for the given icon ID (QtGui.QStyle.SP_*).

Example:

```
from qarbon.external.qt import QtGui
from qarbon.qt.qui.application import Application
from qarbon.qt.gui.icon import getStandardPixmap

app = Application()
pixmap = getStandardPixmap(QtGui.QStyle.SP_MessageBoxWarning, 32)
label = QtGui.QLabel()
label.setPixmap(pixmap)
label.show()
app.exec_()
```

Parameters

- **pixmap_id** (*QtGui.QStyle.SP*) – pixmap name
- **width** (*int*) – pixmap width
- **height** (*int*) – pixmap height [default: None meaning use given width]
- **mode** (*QtGui.QIcon.Mode*) – icon mode
- **state** (*QtGui.QIcon.State*) – icon state

Returns the QPixmap corresponding to the given id. If the standard ID doesn't exist it returns a Null QPixmap

Return type QtGui.QPixmap

qarbon.qt.gui.icon.**getQarbonIcon**(*icon_name*)

Returns a QIcon for the given qarbon icon name.

Example:

```
from qarbon.external.qt import QtGui
from qarbon.qt.qui.application import Application
from qarbon.qt.gui.icon import getQarbonIcon

app = Application()
icon = getQarbonIcon(":/controls/collapse.png")
button = QtGui.QPushButton(icon, "Collapse")
button.show()
app.exec_()
```

Parameters **icon_name** (*str*) – icon name

Returns the QIcon corresponding to the given qarbon name. If the icon doesn't exist it returns a Null icon

Return type QtGui.QIcon

qarbon.qt.gui.icon.**getQarbonPixmap**(*pixmap_name*, *width*, *height=None*, *mode=0*, *state=1*)

Returns a QPixmap for the given pixmap name.

Example:

```
from qarbon.external.qt import QtGui
from qarbon.qt.qui.application import Application
from qarbon.qt.gui.icon import getQarbonPixmap

app = Application()
pixmap = getQarbonPixmap(":/controls/collapse.png", 32)
label = QtGui.QLabel()
label.setPixmap(pixmap)
label.show()
app.exec_()
```

Parameters

- **pixmap_name** (*str*) – pixmap name
- **width** (*int*) – pixmap width
- **height** (*int*) – pixmap height [default: None meaning use given width]
- **mode** (*QtGui.QIcon.Mode*) – icon mode
- **state** (*QtGui.QIcon.State*) – icon state

Returns the QPixmap corresponding to the given id. If the standard ID doesn't exist it returns a Null QPixmap

Return type QtGui.QPixmap

qarbon.qt.gui.icon.**getIcon**(*icon*)

Returns a QIcon for the given icon.

Example:

```
from qarbon.external.qt import QtGui
from qarbon.qt.qui.application import Application
from qarbon.qt.gui.icon import getIcon

app = Application()

# == getThemeIcon("folder-open")
icon = getIcon("folder-open")

# == getQarbonIcon(":/controls/collapse.png")
icon = getIcon(":/controls/collapse.png")

# == Qt.QIcon("MyResource:/bla.png")
icon = getIcon("MyResource:/bla.png")

# == getStandardIcon(QtGui.QStyle.SP_MessageBoxWarning)
icon = getIcon(QtGui.QStyle.SP_MessageBoxWarning)

button = QtGui.QPushButton(icon, "Something")
button.show()
app.exec_()
```

Parameters **icon** (*str or int*) – icon name or ID

Returns the QIcon corresponding to the given icon. If the icon doesn't exist it returns a Null icon

Return type QtGui.QIcon

qarbon.qt.gui.icon.**getPixmap**(*pixmap*, *width*, *height=None*, *mode=0*, *state=1*)

Returns a QPixmap for the given pixmap.

Example:

```
from qarbon.external.qt import QtGui
from qarbon.qt.qui.application import Application
from qarbon.qt.gui.icon import getPixmap

app = Application()

# == getThemePixmap("folder-open", 32)
pixmap = getPixmap("folder-open", 32)

# == getQarbonPixmap(":/controls/collapse.png", 32)
```

```
pixmap = getPixmap(":/controls/collapse.png", 32)

# == QtGui.QPixmap("MyResource:/bla.png")
pixmap = getPixmap("MyResource:/bla.png", 32)

label = QtGui.QLabel()
label.setPixmap(pixmap)
label.show()
app.exec_()
```

Parameters

- **pixmap** (*str or int*) – pixmap name or ID
- **width** (*int*) – pixmap width
- **height** (*int*) – pixmap height [default: None meaning use given width]
- **mode** (*QtGui.QIcon.Mode*) – icon mode
- **state** (*QtGui.QIcon.State*) – icon state

Returns the QPixmap corresponding to the given pixmap. If the pixmap doesn't exist it returns a Null QPixmap

Return type QtGui.QPixmap

qarbon.qt.gui.icon.**Icon**(*obj*)

Returns a QIcon for the given icon.

Example:

```
from qarbon.external.qt import QtGui
from qarbon.qt.qui.application import Application
from qarbon.qt.gui.icon import Icon

app = Application()

# == getThemeIcon("folder-open")
icon = Icon("folder-open")

# == getQarbonIcon(":/controls/collapse.png")
icon = Icon(":/controls/collapse.png")

# == Qt.QIcon("MyResource:/bla.png")
icon = Icon("MyResource:/bla.png")

# == getStandardIcon(QtGui.QStyle.SP_MessageBoxWarning)
icon = Icon(QtGui.QStyle.SP_MessageBoxWarning)

button = QtGui.QPushButton(icon, "Something")
button.show()
app.exec_()
```

Parameters **icon** (*str or int*) – icon name or ID

Returns the QIcon corresponding to the given icon. If the icon doesn't exist it returns a Null icon

Return type QtGui.QIcon

qarbon.qt.gui.icon.**Pixmap**(*obj*, *width*, *height=None*, *mode=0*, *state=1*)

Returns a QPixmap for the given pixmap.

Example:

```
from qarbon.external.qt import QtGui
from qarbon.qt.qui.application import Application
from qarbon.qt.gui.icon import Pixmap

app = Application()

# == getThemePixmap("folder-open", 32)
pixmap = Pixmap("folder-open", 32)

# == getQarbonPixmap(":/controls/collapse.png", 32)
pixmap = Pixmap(":/controls/collapse.png", 32)

# == QtGui.QPixmap("MyResource:/bla.png")
pixmap = Pixmap("MyResource:/bla.png", 32)

label = QtGui.QLabel()
label.setPixmap(pixmap)
label.show()
app.exec_()
```

Parameters

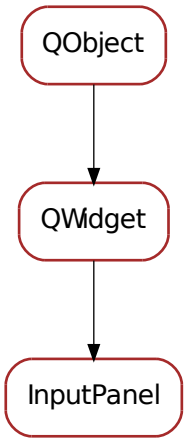
- **pixmap** (*str or int*) – pixmap name or ID
- **width** (*int*) – pixmap width
- **height** (*int*) – pixmap height [default: None meaning use given width]
- **mode** (*QtGui.QIcon.Mode*) – icon mode
- **state** (*QtGui.QIcon.State*) – icon state

Returns the QPixmap corresponding to the given pixmap. If the pixmap doesn't exist it returns a Null QPixmap

Return type QtGui.QPixmap

5.17 qarbon.qt.gui.input

A set of widgets to get input from the user.



Classes

InputPanel	A panel design to get an input from the user.

class qarbon.qt.gui.input.**InputPanel**(*input_data*, *parent=None*)
Bases: PyQt4.QtGui.QWidget

A panel design to get an input from the user.

The input_data is a dictionary which contains information on how to build the input dialog. It **must** contains the following keys:

- *prompt* <str>: message to be displayed

The following are optional keys (and their corresponding default values):

- *title* <str> (doesn't have default value)
- *key* <str> (doesn't have default value): a label to be presented left to the input box represeting the label
- *unit* <str> (doesn't have default value): a label to be presented right to the input box representing the units
- *data_type* <str or sequence> ('String'): type of data to be requested. Standard accepted data types are 'String', 'Integer', 'Float', 'Boolean', 'Text'. A list of elements will be interpreted as a selection. Default TaurusInputPanel class will interpret any custom data types as 'String' and will display input widget accordingly. Custom data types can be handled differently by supplying a different input_panel_klass.
- *minimum* <int/float> (-sys.maxint): minimum value (makes sence when data_type is 'Integer' or 'Float')
- *maximum* <int/float> (sys.maxint): maximum value (makes sence when data_type is 'Integer' or 'Float')
- *step* <int/float> (1): step size value (makes sence when data_type is 'Integer' or 'Float')
- *decimals* <int> (1): number of decimal places to show (makes sence when data_type is 'Float')
- *default_value* <obj> (doesn't have default value): default value
- *allow_multiple* <bool> (False): allow more than one value to be selected (makes sence when data_type is a sequence of possibilities)

Example:

```
app = Application()

d = dict(prompt="What's your favourite car brand?",
         data_type=["Mazda", "Skoda", "Citroen", "Mercedes", "Audi",
                    "Ferrari"],
         default_value="Mercedes")
w = InputPanel(d)

class Listener(object):
    def onAccept(self):
        print "user selected", w.value()

l = Listener()
w.buttonBox().accepted.connect(l.onAccept)
w.show()
app.exec_()
```

fill_main_panel(*panel*, *input_data*)

create_single_input_panel(*input_data*)

create_custom_panel(*input_data*)

create_selection_panel(*input_data*)

create_integer_panel(*input_data*)

create_float_panel(*input_data*)

create_string_panel(*input_data*)

create_text_panel(*input_data*)

create_boolean_panel(*input_data*)

inputPanel()

buttonBox()
Returns the button box from this panel

Returns the button box from this panel

Return type PyQt4.Qt.QDialogButtonBox

addButton(*button*, *role=3*)
Adds the given button with the given to the button box

Parameters

- **button** (*PyQt4.QtGui.QPushButton*) – the button to be added
- **role** (*PyQt4.Qt.QDialogButtonBox.ButtonRole*) – button role

setIconPixmap(*pixmap*)
Sets the icon to the dialog

Parameters **pixmap** (*PyQt4.Qt.QPixmap*) – the icon pixmap

setText(*text*)
Sets the text of this panel

Parameters **text** (*str*) – the new text

getText()
Returns the current text of this panel

Returns the text for this panel

Return type str

setInputFocus()

5.18 qarbon.qt.gui.led

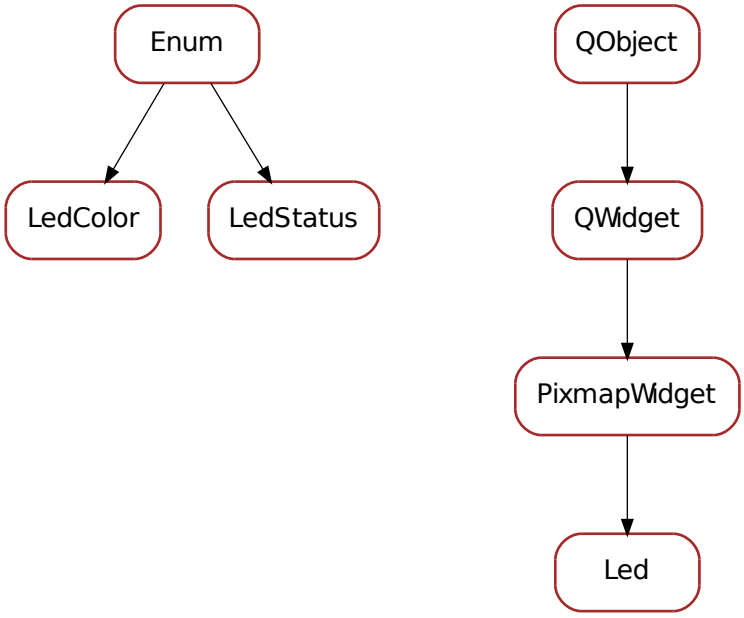
A LED (light-emitting diode) widget.

This widget represents a led. The led has a color and a status (On/Off).

Here is an example showing how to display all possible combinations of color, status:

```
from qarbon.external.qt import QtGui
from qarbon.qt.gui.application import Application
from qarbon.qt.gui.led import Led, LedStatus, LedColor

app = Application()
panel = QtGui.QWidget()
layout = QtGui.QGridLayout()
panel.setLayout(layout)
for i, color in enumerate(LedColor):
    led = Led()
```

```
    led.ledColor = color
    led.ledStatus = LedStatus.Off
    layout.addWidget(led, i, 0)
    led = Led()
    led.ledColor = color
    led.ledStatus = LedStatus.On
    layout.addWidget(led, i, 1)
panel.show()

app.exec_()
```



Classes

Led	A LED (light-emitting diode) like widget

class qarbon.qt.gui.led.**LedColor**
Bases: qarbon.external.enum._enum.Enum

possible led colors

class qarbon.qt.gui.led.**LedStatus**
Bases: qarbon.external.enum._enum.Enum

possible led status

class qarbon.qt.gui.led.**Led**(*parent=None*)
Bases: qarbon.qt.gui.pixmapwidget.PixmapWidget

A LED (light-emitting diode) like widget

DefaultLedPattern = 'qarbon:/led/led_{color}_{status}.png'
constant defining default led image filename pattern

DefaultLedColor = <LedColor.Green: 1>
constant defining default led color (green)

DefaultLedStatus = <LedStatus.On: 1>
constant defining default led status (On)

DefaultLedInverted = False
constant defining default led status invertion (False)

sizeHint()

minimumSizeHint()
Overwrite the default minimum size hint (0,0) to be (8, 8)

Returns the minimum size hint 8, 8

Return type QSize

toLedName(*status=None*, *color=None*, *inverted=None*)
Gives the led name for the given status and color. If status or color are not given, the current led status or color are used.

Parameters

- **status** (*bool*) – the status
- **color** (*str*) – the color

Returns string containing the led name

Return type str

isLedColorValid(*name*)
Determines if the given color name is valid.

Parameters **color** (*str*) – the color

Returns True is the given color name is valid or False otherwise

Return type bool

getLedPatternName()
Returns the current led pattern name :return: led pattern name :rtype: str

setLedPatternName(*name*)
Sets the led pattern name. Should be a string containing a path to valid images. The string can contain the keywords:

1. {status} - transformed to 'on' of 'off' according to the status
2. {color} - transformed to the current led color

Example: **:leds/images256/led_{color}_{status}.png** will be transformed to **:leds/images256/led_red_on.png** when the led status is True and the led color is red.

Parameters **name** (*str*) – new pattern

resetLedPatternName()
Resets the led pattern to **fwk4:/Leds/led_{color}_{status}.png**.

getLedStatus()
Returns the led status :return: led status :rtype: bool

setLedStatus(*status*)
Sets the led status :param status: the new status :type status: bool

resetLedStatus()
Resets the led status

toggleLedStatus()
toggles the current status of the led

getLedInverted()
Returns if the led is inverted. :return: inverted mode :rtype: bool

setLedInverted(*inverted*)
Sets the led inverted mode :param status: the new inverted mode :type status: bool

resetLedInverted()
Resets the led inverted mode

getLedColor()
Returns the led color :return: led color :rtype: LedColor

setLedColor(*color*)
Sets the led color :param status: the new color :type status: LedColor

resetLedColor()
Resets the led color

classmethod getQtDesignerPluginInfo()

ledStatus = <PyQt4.QtCore.pyqtProperty object at 0x318e290>
This property holds the led status: False means OFF, True means ON

Access functions:

- Led.getLedStatus()
- Led.setLedStatus()
- Led.resetLedStatus()

ledColor = <PyQt4.QtCore.pyqtProperty object at 0x318e650>
This property holds the led color

Access functions:

- Led.getLedColor()
- Led.setLedColor()
- Led.resetLedColor()

ledInverted = <PyQt4.QtCore.pyqtProperty object at 0x318e690>
This property holds the led inverted: False means do not invert the

Access functions:

- Led.getLedInverted()
- Led.setLedInverted()
- Led.resetLedInverted()

ledPattern = <PyQt4.QtCore.pyqtProperty object at 0x318e6d0>
This property holds the led pattern name

Access functions:

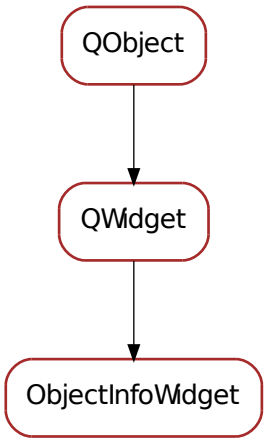
- Led.getLedPatternName()
- Led.setLedPatternName()
- Led.resetLedPatternName()

5.19 qarbon.qt.gui.objectinfowidget

A widget which displays/edits information about a QObject.

Example:

```
from qarbon.external.qt import QtCore, QtGui
from qarbon.qt.gui.application import Application
from qarbon.qt.gui.qobjectinfowidget import ObjectInfoWidget

app = Application()

# mw will be the QObject to be "seen"
mw = QtGui.QMainWindow()
mw.setObjectName("main window")
w = QtGui.QWidget()
w.setObjectName("central widget")
mw.setCentralWidget(w)
l = QtGui.QVBoxLayout()
w.setLayout(l)
l1 = QtGui.QLabel("H1")
l1.setObjectName("label 1")
l.addWidget(l1)
l2 = QtGui.QLabel("H2")
l2.setObjectName("label 2")
l.addWidget(l2)
mw.show()

inspector = ObjectInfoWidget(qobject=mw)
inspector.setAttribute(QtCore.Qt.WA_QuitOnClose, False)
inspector.show()
app.exec_()
```



Classes

ObjectInfoWidget	A widget which displays/edits information about a QObject.

class qarbon.qt.gui.objectinfowidget.**ObjectInfoWidget**(*parent=None*, *qobject=None*)

Bases: PyQt4.QtGui.QWidget

A widget which displays/edits information about a QObject.

setQObject(*qobject*)

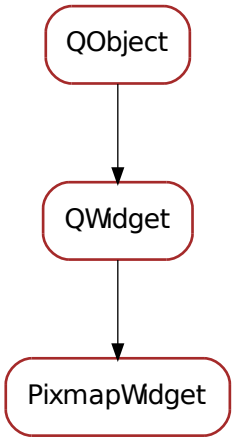
5.20 qarbon.qt.gui.pixmapwidget

A widget that displays an image (pixmap).

You can adjust properties like the aligment inside the widget space, aspect ratio and transformation mode (quality).

Example:

```
from qarbon.external.qt import QtGui
from qarbon.qt.gui.application import Application
from qarbon.qt.gui.icon import Pixmap
from qarbon.qt.gui.pixmapwidget import PixmapWidget

app = Application()
panel = QtGui.QWidget()
layout = QtGui.QGridLayout()
panel.setLayout(layout)
img = PixmapWidget()
layout.addWidget(img)

img.setPixmap(Pixmap(":/led/led_red_on.png"))
img.show()
app.exec_()
```



Classes

PixmapWidget	This widget displays an image (pixmap).

class qarbon.qt.gui.pixmapwidget.**PixmapWidget**(*parent=None*)

Bases: PyQt4.QtGui.QWidget

This widget displays an image (pixmap). By default the pixmap is scaled to the widget size and the aspect ratio is kept. The default alignment of the pixmap inside the widget space is horizontal left, vertical center.

DefaultAlignment = 129

DefaultAspectRatioMode = 1

DefaultTransformationMode = 1

pixmapChanged = <PyQt4.QtCore.pyqtSignal object at 0x318e090>
Signal emited when pixmap source changes

recalculatePixmap()

paintEvent(*paintEvent*)
Overwrite the paintEvent from QWidget to draw the pixmap

resizeEvent(*event*)

sizeHint()

getPixmap()
Returns the pixmap.Returns None if no pixmap is set.

Returns the current pixmap

Return type QtGui.QPixmap

setPixmap(*pixmap*)
Sets the pixmap for this widget. Setting it to None disables pixmap

Parameters **pixmap** (*QtGui.QPixmap*) – the new pixmap

resetPixmap()
Resets the pixmap for this widget.

getAspectRatioMode()
Returns the aspect ratio to apply when drawing the pixmap.

Returns the current aspect ratio

Return type QtCore.Qt.AspectRatioMode

setAspectRatioMode(*aspect*)
Sets the aspect ratio mode to apply when drawing the pixmap.

Parameters **pixmap** (*QtCore.Qt.AspectRatioMode*) – the new aspect ratio mode

resetAspectRatioMode()
Resets the aspect ratio mode to KeepAspectRatio

getTransformationMode()
Returns the transformation mode to apply when drawing the pixmap.

Returns the current transformation mode

Return type QtCore.Qt.TransformationMode

setTransformationMode(*transformation*)
Sets the transformation mode to apply when drawing the pixmap.

Parameters **pixmap** (*QtCore.Qt.TransformationMode*) – the new transformation mode

resetTransformationMode()
Resets the transformation mode to SmoothTransformation

getAlignment()
Returns the alignment to apply when drawing the pixmap.

Returns the current alignment

Return type QtCore.Qt.Alignment

setAlignment(*alignment*)
Sets the alignment to apply when drawing the pixmap.

Parameters **pixmap** (*QtCore.Qt.Alignment*) – the new alignment

resetAlignment()
Resets the transformation mode to QtCore.Qt.AlignLeft | QtCore.Qt.AlignVCenter

pixmap = <PyQt4.QtCore.pyqtProperty object at 0x318e150>
This property holds the widget's pixmap

Access functions:

- PixmapWidget.getPixmap()
- PixmapWidget.setPixmap()
- PixmapWidget.resetPixmap()

aspectRatioMode = <PyQt4.QtCore.pyqtProperty object at 0x318e190>
This property holds the widget's pixmap aspect ratio mode

Access functions:

- PixmapWidget.getAspectRatioMode()
- PixmapWidget.setAspectRatioMode()
- PixmapWidget.resetAspectRatioMode()

transformationMode = <PyQt4.QtCore.pyqtProperty object at 0x318e1d0>
This property holds the widget's pixmap transformation mode

Access functions:

- PixmapWidget.getTransformationMode()
- PixmapWidget.setTransformationMode()
- PixmapWidget.resetTransformationMode()

alignment = <PyQt4.QtCore.pyqtProperty object at 0x318e210>
This property holds the widget's pixmap alignment

Access functions:

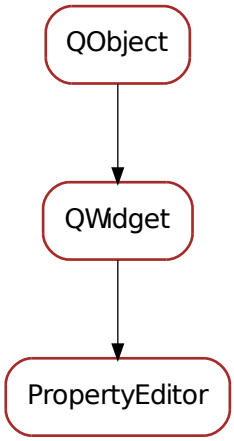
- PixmapWidget.getAlignment()
- PixmapWidget.setAlignment()
- PixmapWidget.resetAlignment()

5.21 qarbon.qt.gui.propertyeditor

A widget dedicated view/edit the properties of any QObject.

Example:

```
from qarbon.external.qt import QtGui
from qarbon.qt.gui.application import Application
from qarbon.qt.gui.propertyeditor import PropertyEditor

app = Application()
inspector = PropertyEditor(qobject=None)

# watch myself... weard
inspector.setQObject(inspector)
inspector.show()
app.exec_()
```



Classes

PropertyEditor	A widget dedicated view/edit the properties of any QObject.

class qarbon.qt.gui.propertyeditor.**PropertyEditor**(*parent=None*, *qobject=None*)
Bases: PyQt4.QtGui.QWidget

A widget dedicated view/edit the properties of any QObject.

qobject
returns the current QObject being edited or None if no QObject is associated with the editor.

Returns the current QObject being edited or None if no QObject is associated with the editor

setQObject(*qobject*)
Sets the current QObject whose properties are to been seen by the editor.

Parameters **qobject** – the new QObject (can be None)

5.22 qarbon.qt.gui.treeqobject

A tree widget representing QObject hierarchy (for development purposes).

The most common use case of this widget is to debug applications which may have "zombie" widgets lying around when some widget is removed, reparented in a dynammic GUI.

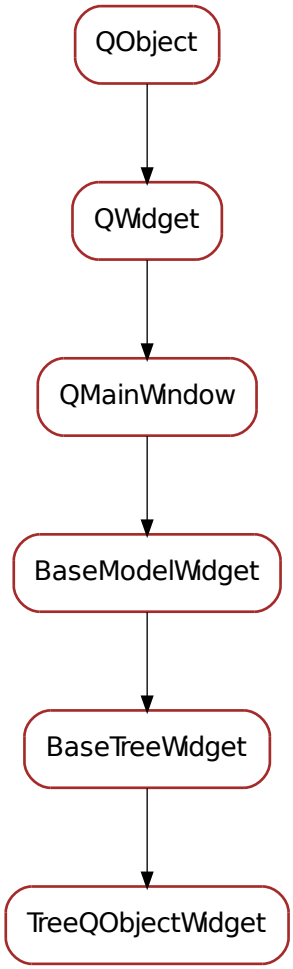
Example:

```
from qarbon.external.qt import QtCore, QtGui
from qarbon.qt.gui.application import Application
from qarbon.qt.gui.treeqobject import TreeQObjectWidget

app = Application()

# mw will be the QObject to be "seen" in the Tree (along with all its
# childs, of course)
```

```
mw = QtGui.QMainWindow()
mw.setObjectName("main window")
w = QtGui.QWidget()
w.setObjectName("central widget")
mw.setCentralWidget(w)
l = QtGui.QVBoxLayout()
w.setLayout(l)
l1 = QtGui.QLabel("H1")
l1.setObjectName("label 1")
l.addWidget(l1)
l2 = QtGui.QLabel("H2")
l2.setObjectName("label 2")
l.addWidget(l2)
mw.show()

inspector = TreeQObjectWidget(qobject=w)
inspector.setAttribute(QtCore.Qt.WA_QuitOnClose, False)
inspector.show()
app.exec_()
```

Functions

getQObjectIcon
getQObjectTree
getQObjectTreeStr
getQObjectTreeAsDict
getQObjectTreeAsList



Classes

TreeQObjectWidget	A tree representation of the selected QObject childs.

class qarbon.qt.gui.treeqobject.**QObjectRepresentation**
Bases: qarbon.external.enum._enum.Enum

possible displays of a QObject

qarbon.qt.gui.treeqobject.**getQObjectTree**(*qobject=None*, *ffilter=<function _filter at 0x31949b0>*)

qarbon.qt.gui.treeqobject.**getQObjectTreeStr**(*qobject=None*, *representation=<QObjectRepresentation.ClassName: 0>*, *ffilter=<function _filter at 0x31949b0>*)

class qarbon.qt.gui.treeqobject.**TreeQObjectInfoModel**(*parent=None*, *data=None*)
Bases: qarbon.qt.gui.basemodel.BaseModel

ColumnNames = ('Class', 'Object name')

ColumnRoles = ((<QObjectRepresentation.ClassName: 0>,), <QObjectRepresentation.ObjectName: 1>)

role(*column*, *depth=0*)

roleIcon(*taurus_role*)

roleSize(*taurus_role*)

roleToolTip(*role*)

setupModelData(*qobject*)

class qarbon.qt.gui.treeqobject.**TreeQObjectWidget**(*parent=None*, *with_navigation_bar=True*, *with_filter_widget=True*, *perspective=None*, *proxy=None*, *qobject=None*)

Bases: qarbon.qt.gui.basetree.BaseTreeWidget

A tree representation of the selected QObject childs.

The use case of this widget is to debug applications which may have "zombie" widgets lying around when some widget is removed, reparented in a dynammic GUI.

KnownPerspectives = {'Default': {'model': [<class 'qarbon.qt.gui.treeqobject.TreeQObjectInfoModel'>], 'icon':

DftPerspective = 'Default'

5.23 qarbon.qt.gui.util

Helper functions to deal with Qt GUI related stuff

Functions

isWidget	Determines if the given object is a Qt Widget.
isWidgetClass	Determines if the given object is a Qt Widget class.
getWidgetClasses	Returns the widget classes defined in a given module.
grabWidget	Grabs the given widget into the given image filename.

qarbon.qt.gui.util.**isWidget**(*obj*)

Determines if the given object is a Qt Widget.

Parameters **obj** – object

Returns True if the given object is a Qt Widget or False otherwise

Return type bool

qarbon.qt.gui.util.**isWidgetClass**(*obj*)

Determines if the given object is a Qt Widget class.

Parameters **obj** – object

Returns True if the given object is a Qt Widget class or False otherwise

Return type bool

qarbon.qt.gui.util.**getWidgetClasses**(*module_name*)

Returns the widget classes defined in a given module.

Returns:

{widget full name(str): {"klass": widget class(class), "name": widget name(str), "full_name": widget full name(str)}}

Parameters **module_name** (*str*) – name of the module in the format "a.b.c"

Returns a map with the widgets for the given module

Return type dict

qarbon.qt.gui.util.**grabWidget**(*widget*, *fileName*, *period=None*)

Grabs the given widget into the given image filename. If period is None (default) it grabs immediately once and returns. If period is given and >0 means grab the image every period (in seconds).

> **Warning:** this method **MUST** be called from the same thread which created the widget

Parameters

- **widget** (*QtWidget*) – the Qt widget to be grabbed
- **fileName** (*str*) – the name of the image file
- **period** (*float*) – period (seconds)

5.24 qarbon.qt.gui.x11

A X11 widget that may run any command and an XTermWidget runs a xterm.

Note: this widget only works on X11 systems.

Example:

```
from qarbon.external.qt import QtGui
from qarbon.qt.gui.application import Application
from qarbon.qt.gui.x11terminal import XTermWindow

app = Application()
term = XTermWindow()
term.start()
term.show()
app.exec_()
```

Classes

XEmbedCommandWidget	
XEmbedCommandWindow	
XTermWidget	A widget with an xterm console inside.
XTermWindow	The QMainWindow version of XTermWidget

class qarbon.qt.gui.x11.**XCommandWidget**(*parent=None*)

Bases: PyQt4.QtGui.QWidget

A widget displaying an X11 window inside from a command.

Example:

```
from qarbon.external.qt import QtGui
from qarbon.qt.gui.application import Application
from qarbon.qt.gui.x11 import XCommandWidget

app = Application()
w = QtGui.QMainWindow()
cmdWidget = XCommandWidget(parent=w)
cmdWidget.command = 'xterm'
```

```
cmdWidget.winIdParam = '-into'
cmdWidget.start()
w.setCentralWidget(cmdWidget)
w.show()
app.exec_()
```

DefaultAutoRestart = False

DefaultWinIdParam = '-into'

getX11WinId()

getX11Widget()

getProcess()

getCommand()

setCommand(*command*)

resetCommand()

getWinIdParam()

setWinIdParam(*winIdParam*)

resetWinIdParam()

setExtraParams(*params*)

getExtraParams()

resetExtraParams()

setAutoRestart(*yesno*)

getAutoRestart()

resetAutoRestart()

setWorkingDirectory(*wd*)

getWorkingDirectory()

start(*wait=0*)
wait < 0 -> wait forever, wait == 0 -> not wait, wait > 0 -> wait amount in seconds

restart(*wait=0*)

kill(*wait=0*)

terminate(*wait=0*)

deleteLater()

classmethod getQtDesignerPluginInfo()

command = <PyQt4.QtCore.pyqtProperty object at 0x31a3090>

winIdParam = <PyQt4.QtCore.pyqtProperty object at 0x31a30d0>

extraParams = <PyQt4.QtCore.pyqtProperty object at 0x31a3110>

autoRestart = <PyQt4.QtCore.pyqtProperty object at 0x31a3150>

workingDirectory = <PyQt4.QtCore.pyqtProperty object at 0x31a3190>

class qarbon.qt.gui.x11.**XCommandWindow**(***kwargs*)
Bases: PyQt4.QtGui.QMainWindow

The QMainWindow version of XCommandWidget.

Example:

```
from qarbon.external.qt import QtGui
from qarbon.qt.gui.application import Application
from qarbon.qt.gui.x11 import XCommandWindow

app = Application()
w = XCommandWindow()
w.command = 'xterm'
w.winIdParam = '-into'
w.start()
w.show()
app.exec_()
```

Widget
alias of XCommandWidget

XWidget()

start(*wait=0*)

restart(*wait=0*)

terminate(*wait=0*)

getCommand()

setCommand(*command*)

resetCommand()

getWinIdParam()

setWinIdParam(*winIdParam*)

resetWinIdParam()

setExtraParams(*params*)

getExtraParams()

resetExtraParams()

setAutoRestart(*yesno*)

getAutoRestart()

resetAutoRestart()

setWorkingDirectory(*wd*)

getWorkingDirectory()

command = <PyQt4.QtCore.pyqtProperty object at 0x31a3290>

winIdParam = <PyQt4.QtCore.pyqtProperty object at 0x31a32d0>

extraParams = <PyQt4.QtCore.pyqtProperty object at 0x31a3310>

autoRestart = <PyQt4.QtCore.pyqtProperty object at 0x31a3350>

workingDirectory = <PyQt4.QtCore.pyqtProperty object at 0x31a3390>

class qarbon.qt.gui.x11.**XTermWidget**(*auto_start=False*, *parent=None*)
Bases: qarbon.qt.gui.x11.XCommandWidget

A widget with an xterm console inside.

Example:

```
from qarbon.external.qt import QtGui
from qarbon.qt.gui.application import Application
from qarbon.qt.gui.x11 import XTermWidget
```

```
app = Application()
w = QtGui.QMainWindow()
term = XTermWidget(parent=w)
term.extraParams = ["-e", "python"]
w.setCentralWidget(term)
w.start()
w.show()
app.exec_()
```

sizeHint()

classmethod getQtDesignerPluginInfo()

class qarbon.qt.gui.x11.**XTermWindow**(***kwargs*)
Bases: qarbon.qt.gui.x11.XCommandWindow

The QMainWindow version of XTermWidget

from qarbon.external.qt import QtGui from qarbon.qt.gui.application import Application from qarbon.qt.gui.x11 import XTermWidget

app = Application() term = XTermWindow() term.start() term.show() app.exec_()

Widget
alias of XTermWidget

5.25 qarbon.release

Release data for the qarbon project.

It contains the following members:

- version : (str) version string
- description : (str) brief description
- long_description : (str) a long description
- license : (str) license
- authors : (seq<seq<str,str,str>>) the list of authors
- url : (str) the project url
- download_url : (str) the project download url
- keywords : list<str> list of keywords
- classifiers : list<str> list of applicable classifiers

qarbon.release.**name = 'qarbon'**
Name of the package for release purposes. This is the name which labels the tarballs and RPMs made by distutils, so it's best to lowercase it.

qarbon.release.**version_info = (0, 1, 0, 'dev', 0)**
For versions with substrings (like 0.6.16.svn), use an extra . to separate the new substring. We have to avoid using either dashes or underscores, because bdist_rpm does not accept dashes (an RPM) convention, and bdist_deb does not accept underscores (a Debian convention).

qarbon.release.**revision = '0'**
revision number

qarbon.release.**description = 'python library of Qt widgets.'**
package description

qarbon.release.**long_description = 'Qarbon is a python library of Qt widgets.'**
long description

qarbon.release.**license** = 'GNU Lesser General Public License v3 or later (LGPLv3+)'
license

qarbon.release.**authors** = [('Tiago', 'Tiago Coutinho', 'coutinho@esrf.fr')]
authors

qarbon.release.**url** = 'http://qarbon.rtfd.org/'
package URL

qarbon.release.**download_url** = 'http://pypi.python.org/pypi/qarbon/'
download URL

qarbon.release.**keywords** = ['Python', 'Qt']
keywords

qarbon.release.**classifiers** = ['Development Status :: 2 - Pre-Alpha', 'Intended Audience :: Developers', 'License
package classifiers

qarbon.release.**requirements** = []
external requirements

5.26 qarbon.signal

Simple implementation of signal/slot pattern.

Classes

Signal	Represents typical Signal pattern with connect, disconnect and emit.

class qarbon.signal.**Signal**(**args*, ***kwargs*)
Bases: object

Represents typical Signal pattern with connect, disconnect and emit. Can be used as a descriptor. Example:

```
class Car(object):

    temperatureChanged = Signal(float)

    def set_temperature(self, temp):
        self.__temp = temp
        self.temperatureChanged.emit(temp)

car = Car()

def on_temp_changed(temp):
    print("Car temperature changed to {0}".format(temp))

car.temperatureChanged.connect(on_temp_changed)

car.set_temperature(13.4)
```

set_cache(**args*, ***kwargs*)
Fills the cache without actually emitting the signal. Not part of the API. It is a helper method for signal owners to use as necessary.

slots()
Returns the list of connected slots.

connect(*slot*)
Connect a slot to this signal.

disconnect(*slot*)
Disconnect the slot from this signal.

emit(**args*, ***kwargs*)
emit signal.

5.27 qarbon.util

Helper functions.

Functions

is_string	Determines if the given object is a string.
is_sequence	Determines if the given object is a sequence.
module_directory	Returns the location of a given module.
import_module	Import a module.
callable_weakref	This function returns a callable weak reference to a callable object.

qarbon.util.**is_string**(*obj*)
Determines if the given object is a string.

Parameters **obj** (*object*) – the object to be analysed

Returns True if the given object is a string or False otherwise

Return type bool

qarbon.util.**is_sequence**(*obj*, *inc_string=False*)
Determines if the given object is a sequence.

Parameters

- **obj** (*object*) – the object to be analysed
- **inc_string** (*bool*) – if False, exclude str/unicode objects from the list of possible sequence objects

Returns True if the given object is a sequence or False otherwise

Return type bool

qarbon.util.**module_directory**(*module*)
Returns the location of a given module.

Parameters **module** (*module*) – the module object

Returns the directory where the module is located

Return type str

qarbon.util.**import_module**(*name*, *package=None*)
Import a module.

The 'package' argument is required when performing a relative import. It specifies the package to use as the anchor point from which to resolve the relative import to an absolute import.

qarbon.util.**callable_weakref**(*obj*, *del_cb=None*)
This function returns a callable weak reference to a callable object. Object can be a callable object, a function or a method.

Parameters

- **object** (*callable object*) – a callable object

- **del_cb** (*callable object or None*) – calback function. Default is None meaning to callback.

Returns a weak reference for the given callable

Return type BoundMethodWeakref or weakref.ref

5.28 qarbon.value

Value definition.

Classes

AttributeConfig	
Value	A qarbon value.
AttributeValue	A qarbon value.

class qarbon.value.**AttributeConfig**

Bases: object

name = ‘’

label = ‘—-‘

description = ‘’

ndim = -1

format = ‘%s’

display_level = <DisplayLevel._Invalid: 4>

display_format = ‘!s’

access = <Access._Invalid: 4>

unit = None

standard_unit = None

display_unit = None

min_value = None

max_value = None

min_alarm = None

max_alarm = None

min_warning = None

max_warning = None

value_range = (None, None)

alarm_range = (None, None)

warning_range = (None, None)

is_write()

is_readonly()

is_readwrite()

is_scalar()

is_spectrum()

is_mage()

class qarbon.value.**Value**
Bases: object

A qarbon value. A container for a value read from a qarbon model. It contains the following members:

- r_value (Quantity): (aka: value) a Quantity representing the read value
- r_timestamp (datetime.datetime): the timestamp of reading the value
- w_value (Quantity): a Quantity representing the write value
- quality (Quality): the quality related to the read value
- **exc_info (tuple): a 3-tuple equivalent to sys.exc_info() if reading a** value resulted in an exception or None otherwise
- error (bool): tells the read resulted in an error

Example on how to pretty print

r_value = None

r_timestamp = None

r_ndim = None

r_quality = None

w_value = None

exc_info = None

value

timestamp

ndim

quality

error

is_scalar()

is_spectrum()

is_image()

class qarbon.value.**AttributeValue**
Bases: qarbon.value.Value

A qarbon value. A container for a value read from a qarbon model. It contains the following members:

- r_value (Quantity): (aka: value) a Quantity representing the read value
- r_timestamp (datetime.datetime): the timestamp of reading the value
- w_value (Quantity): a Quantity representing the write value
- quality (Quality): the quality related to the read value
- **exc_info (tuple): a 3-tuple equivalent to sys.exc_info() if reading a** value resulted in an exception or None otherwise
- error (bool): tells the read resulted in an error
- config (AttributeConfig): config object from which this value was obtained

Other configuration values can also be accessed:

- name (str): model name from which the value was obtained

- min_value (Quantity): minimum value allowed
- max_value (Quantity): maximum value allowed
- min_alarm (Quantity): minimum alarm value trigger
- max_alarm (Quantity): maximum alarm value trigger
- min_warning (Quantity): minimum warning value trigger
- max_warning (Quantity): maximum warning value trigger
- description (str): a description

Example on how to pretty print

config = <qarbon.value.AttributeConfig object at 0x22b8ed0>

5.29 qarbon

qarbon.color	Helper functions to translate state to color.
qarbon.config	Global configuration.
qarbon.core	Model core module.
qarbon.executor	
qarbon.log	Helper logging functions.
qarbon.node	Node module.
qarbon.plugin	Plugin extension manager.
qarbon.release	Release data for the qarbon project.
qarbon.signal	Simple implementation of signal/slot pattern.
qarbon.util	Helper functions.
qarbon.value	Value definition.

5.30 qarbon.qt.gui

5.30.1 Helpers

qarbon.qt.gui.action	Helper functions to access QAction.
qarbon.qt.gui.application	Helper functions to manage QApplication.
qarbon.qt.gui.color	Helper functions to colors from state
qarbon.qt.gui.icon	Helper functions to handle icons and pixmaps
qarbon.qt.gui.util	Helper functions to deal with Qt GUI related stuff

5.30.2 Widgets

qarbon.qt.gui.baseview	A base view widget and toolbar.
qarbon.qt.gui.basemodel	A base model and a base tree item.
qarbon.qt.gui.basetree	A base tree widget and toolbar.
qarbon.qt.gui.input	A set of widgets to get input from the user.
qarbon.qt.gui.axeswidget	Multiple axis (axes) widget.
qarbon.qt.gui.groupbox	A colapsable container widget with (optional) title.
qarbon.qt.gui.led	A LED (light-emitting diode) widget.
qarbon.qt.gui.pixmapwidget	A widget that displays an image (pixmap).
qarbon.qt.gui.propertyeditor	A widget dedicated view/edit the properties of any QObject.
qarbon.qt.gui.objectinfowidget	A widget which displays/edits information about a QObject.
qarbon.qt.gui.treeqobject	A tree widget representing QObject hierarchy (for development purposes).

Continued on next page

Table 5.30 – continued from previous page

qarbon.qt.gui.x11	A X11 widget that may run any command and an XTermWidget runs a xterm.

CHAPTER 6

Glossary

... The default Python prompt of the interactive shell when entering code for an indented code block or within a pair of matching left and right delimiters (parentheses, square brackets or curly braces).

>>> The default Python prompt of the interactive shell. Often seen for code examples which can be executed interactively in the interpreter.

API An application programming interface (API) is a particular set of rules and specifications that software programs can follow to communicate with each other. It serves as an interface between different software programs and facilitates their interaction, similar to the way the user interface facilitates interaction between humans and computers. An API can be created for applications, libraries, operating systems, etc., as a way of defining their "vocabularies" and resources request conventions (e.g. function-calling conventions). It may include specifications for routines, data structures, object classes, and protocols used to communicate between the consumer program and the implementer program of the API.

argument A value passed to a function or method, assigned to a named local variable in the function body. A function or method may have both positional arguments and keyword arguments in its definition. Positional and keyword arguments may be variable-length: $\star$ accepts or passes (if in the function definition or call) several positional arguments in a list, while $\star\star$ does the same for keyword arguments in a dictionary.

Any expression may be used within the argument list, and the evaluated value is passed to the local variable.

attribute A value associated with an object which is referenced by name using dotted expressions. For example, if an object *o* has an attribute *a* it would be referenced as *o.a*.

dictionary An associative array, where arbitrary keys are mapped to values. The keys can be any object with `__hash__()` and `__eq__()` methods. Called a hash in Perl.

CCD A charge-coupled device (CCD) is a device for the movement of electrical charge, usually from within the device to an area where the charge can be manipulated, for example conversion into a digital value. This is achieved by "shifting" the signals between stages within the device one at a time. CCDs move charge between capacitive bins in the device, with the shift allowing for the transfer of charge between bins.

class A template for creating user-defined objects. Class definitions normally contain method definitions which operate on instances of the class.

CLI A command-line interface (CLI) is a mechanism for interacting with a computer operating system or software by typing commands to perform specific tasks. This text-only interface contrasts with the use of a mouse pointer with a graphical user interface (*GUI*) to click on options, or menus on a text user interface (TUI) to select options. This method of instructing a computer to perform a given task is referred to as "entering" a command: the system waits for the user to conclude the submitting of the text command by pressing the "Enter" key (a descendant of the "carriage return" key of a typewriter keyboard). A command-line interpreter then receives, parses, and executes the requested user command. The command-line interpreter may be run in a text terminal or in a terminal emulator window as a remote shell client such as PuTTY. Upon completion, the command usually returns output to the user in the form of text lines on the CLI. This output may be an answer if the command was a question, or otherwise a summary of the operation.

client-server model The client-server model of computing is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters,

called clients. Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server machine is a host that is running one or more server programs which share their resources with clients. A client does not share any of its resources, but requests a server's content or service function. Clients therefore initiate communication sessions with servers which await incoming requests.

daemon In Unix and other computer multitasking operating systems, a daemon is a computer program that runs in the background, rather than under the direct control of a user. They are usually initiated as background processes. Typically daemons have names that end with the letter "d": for example, *syslogd*, the daemon that handles the system log, or *sshd*, which handles incoming SSH connections.

dial See *dial position*

dial position Position in controller units (See also *user position*).

expression A piece of syntax which can be evaluated to some value. In other words, an expression is an accumulation of expression elements like literals, names, attribute access, operators or function calls which all return a value. In contrast to many other languages, not all language constructs are expressions. There are also *statement*s which cannot be used as expressions, such as `print()` or `if`. Assignments are also statements, not expressions.

function A series of statements which returns some value to a caller. It can also be passed zero or more arguments which may be used in the execution of the body. See also *argument* and *method*.

generator A function which returns an iterator. It looks like a normal function except that it contains `yield` statements for producing a series a values usable in a for-loop or that can be retrieved one at a time with the `next()` function. Each `yield` temporarily suspends processing, remembering the location execution state (including local variables and pending try-statements). When the generator resumes, it picks-up where it left-off (in contrast to functions which start fresh on every invocation).

generator expression An expression that returns an iterator. It looks like a normal expression followed by a `for` expression defining a loop variable, range, and an optional `if` expression. The combined expression generates values for an enclosing function:

```
>>> sum(i*i for i in range(10))         # sum of squares 0, 1, 4, ... 81
285
```

GUI A graphical user interface (GUI) is a type of user interface that allows users to interact with electronic devices with images rather than text commands. GUIs can be used in computers, hand-held devices such as MP3 players, portable media players or gaming devices, household appliances and office equipment. A GUI represents the information and actions available to a user through graphical icons and visual indicators such as secondary notation, as opposed to text-based interfaces (*CLI*), typed command labels or text navigation. The actions are usually performed through direct manipulation of the graphical elements.

interactive Python has an interactive interpreter which means you can enter statements and expressions at the interpreter prompt, immediately execute them and see their results. Just launch `python` with no arguments (possibly by selecting it from your computer's main menu). It is a very powerful way to test out new ideas or inspect modules and packages (remember `help(x)`).

interpreted Python is an interpreted language, as opposed to a compiled one, though the distinction can be blurry because of the presence of the bytecode compiler. This means that source files can be run directly without explicitly creating an executable which is then run. Interpreted languages typically have a shorter development/debug cycle than compiled ones, though their programs generally also run more slowly. See also *interactive*.

iterable An object capable of returning its members one at a time. Examples of iterables include all sequence types (such as `list`, `str`, and `tuple`) and some non-sequence types like `dict` and `file` and objects of any classes you define with an `__iter__()` or `__getitem__()` method. Iterables can be used in a `for` loop and in many other places where a sequence is needed (`zip()`, `map()`, ...). When an iterable object is passed as an argument to the built-in function `iter()`, it returns an iterator for the object. This iterator is good for one pass over the set of values. When using iterables, it is usually not necessary to call `iter()` or deal with iterator objects yourself. The `for` statement does that automatically for you, creating a temporary unnamed variable to hold the iterator for the duration of the loop. See also *iterator*, *sequence*, and *generator*.

iterator An object representing a stream of data. Repeated calls to the iterator's `next()` method return successive items in the stream. When no more data are available a `StopIteration` exception is raised instead. At this point, the iterator object is exhausted and any further calls to its `next()` method just raise `StopIteration` again. Iterators are required to have an `__iter__()` method that returns the iterator object itself so every iterator is also iterable and may be used in most places where other iterables are accepted. One notable exception is code which attempts multiple iteration passes. A container object (such as a `list`) produces a fresh new iterator each time you pass it to the `iter()` function or use it in a `for` loop. Attempting this with an iterator will just return the same exhausted iterator object used in the previous iteration pass, making it appear like an empty container.

More information can be found in *Iterator Types*.

key function A key function or collation function is a callable that returns a value used for sorting or ordering. For example, `locale.strxfrm()` is used to produce a sort key that is aware of locale specific sort conventions.

A number of tools in Python accept key functions to control how elements are ordered or grouped. They include `min()`, `max()`, `sorted()`, `list.sort()`, `heapq.nsmallest()`, `heapq.nlargest()`, and `itertools.groupby()`.

There are several ways to create a key function. For example. the `str.lower()` method can serve as a key function for case insensitive sorts. Alternatively, an ad-hoc key function can be built from a `lambda` expression such as `lambda r: (r[0], r[2])`. Also, the `operator` module provides three key function constructors: `attrgetter()`, `itemgetter()`, and `methodcaller()`. See the *Sorting HOW TO* for examples of how to create and use key functions.

keyword argument Arguments which are preceded with a `variable_name=` in the call. The variable name designates the local name in the function to which the value is assigned. `**` is used to accept or pass a dictionary of keyword arguments. See *argument*.

lambda An anonymous inline function consisting of a single *expression* which is evaluated when the function is called. The syntax to create a lambda function is `lambda [arguments]: expression`

list A built-in Python *sequence*. Despite its name it is more akin to an array in other languages than to a linked list since access to elements are O(1).

list comprehension A compact way to process all or part of the elements in a sequence and return a list with the results. `result = ["0x%02x" % x for x in range(256) if x % 2 == 0]` generates a list of strings containing even hex numbers (0x..) in the range from 0 to 255. The `if` clause is optional. If omitted, all elements in `range(256)` are processed.

MCA Multichannel Analyzer (MCA) is a device for ...

method A function which is defined inside a class body. If called as an attribute of an instance of that class, the method will get the instance object as its first *argument* (which is usually called `self`). See *function* and *nested scope*.

namespace The place where a variable is stored. Namespaces are implemented as dictionaries. There are the local, global and built-in namespaces as well as nested namespaces in objects (in methods). Namespaces support modularity by preventing naming conflicts. For instance, the functions `__builtin__.open()` and `os.open()` are distinguished by their namespaces. Namespaces also aid readability and maintainability by making it clear which module implements a function. For instance, writing `random.seed()` or `itertools.izip()` makes it clear that those functions are implemented by the `random` and `itertools` modules, respectively.

nested scope The ability to refer to a variable in an enclosing definition. For instance, a function defined inside another function can refer to variables in the outer function. Note that nested scopes work only for reference and not for assignment which will always write to the innermost scope. In contrast, local variables both read and write in the innermost scope. Likewise, global variables read and write to the global namespace.

new-style class Any class which inherits from `object`. This includes all built-in types like `list` and `dict`. Only new-style classes can use Python's newer, versatile features like `__slots__`, descriptors, properties, and `__getattribute__()`.

object Any data with state (attributes or value) and defined behavior (methods). Also the ultimate base class of any *new-style class*.

OS An operating system (OS) is software, consisting of programs and data, that runs on computers, manages computer hardware resources, and provides common services for execution of various application software. Operating system is the most important type of system software in a computer system. Without an operating system, a user cannot run an application program on their computer, unless the application program is self booting.

plug-in a plug-in (or plugin) is a set of software components that adds specific abilities to a larger software application. If supported, plug-ins enable customizing the functionality of an application. For example, plug-ins are commonly used in web browsers to play video, scan for viruses, and display new file types.

plugin See *plug-in*.

positional argument The arguments assigned to local names inside a function or method, determined by the order in which they were given in the call. * is used to either accept multiple positional arguments (when in the definition), or pass several arguments as a list to a function. See *argument*.

Python 3000 Nickname for the Python 3.x release line (coined long ago when the release of version 3 was something in the distant future.) This is also abbreviated "Py3k".

Pythonic An idea or piece of code which closely follows the most common idioms of the Python language, rather than implementing code using concepts common to other languages. For example, a common idiom in Python is to loop over all elements of an iterable using a `for` statement. Many other languages don't have this type of construct, so people unfamiliar with Python sometimes use a numerical counter instead:

```
for i in range(len(food)):
    print food[i]
```

As opposed to the cleaner, Pythonic method:

```
for piece in food:
    print piece
```

SCADA supervisory control and data acquisition (SCADA) generally refers to industrial control systems: computer systems that monitor and control industrial, infrastructure, or facility-based processes.

SDS Sardana Device server (SDS) is the sardana tango device server *daemon*.

sequence An *iterable* which supports efficient element access using integer indices via the `__getitem__()` special method and defines a `len()` method that returns the length of the sequence. Some built-in sequence types are `list`, `str`, `tuple`, and `unicode`. Note that `dict` also supports `__getitem__()` and `__len__()`, but is considered a mapping rather than a sequence because the lookups use arbitrary *immutable* keys rather than integers.

slice An object usually containing a portion of a *sequence*. A slice is created using the subscript notation, `[]` with colons between numbers when several are given, such as in `variable_name[1:3:5]`. The bracket (subscript) notation uses `slice` objects internally (or in older versions, `__getslice__()` and `__setslice__()`).

statement A statement is part of a suite (a "block" of code). A statement is either an *expression* or a one of several constructs with a keyword, such as `if`, `while` or `for`.

triple-quoted string A string which is bound by three instances of either a quotation mark (") or an apostrophe ('). While they don't provide any functionality not available with single-quoted strings, they are useful for a number of reasons. They allow you to include unescaped single and double quotes within a string and they can span multiple lines without the use of the continuation character, making them especially useful when writing docstrings.

type The type of a Python object determines what kind of object it is; every object has a type. An object's type is accessible as its `__class__` attribute or can be retrieved with `type(obj)`.

user See *user position*

user position Moveable position in user units (See also *dial position*). Dial and user units are related by the following expressions:

user = sign x dial + offset dial = controller_position / steps_per_unit

where *sign* is -1 or 1. *offset* can be any number and *steps_per_unit* must be non zero.

CHAPTER 7

Revision

Contributers T. Coutinho

Last Update April 29, 2014

7.1 History of modifications

Date	Revision	Description	Author
15/10/13	0.1	Initial Version	T. Coutinho

7.2 Version history

version	Changes
0.1	First official release

CHAPTER 8

Documentation to be done

Qarbon is a python library of Qt widgets.

An *Overview* guide will help you getting started with the basic qarbon concepts. The *FAQ* will answer many of your questions.

For sampling, see the *Examples* and *Screenshots* chapters.

q