PrettyQt Documentation

Release 0.50.2

Philipp Temminghoff

Feb 14, 2020

PrettyQt

PrettyQt is a thin wrapper for PyQt5 / PySide2 APIs.

Source code https://github.com/phil65/prettyqt

CHAPTER 1

PrettyQt

Pythonic layer on top of PyQt5 / PySide2

- Free software: MIT license
- Documentation: https://prettyqt.readthedocs.io.

1.1 Features

- TODO

1.2 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install PrettyQt, run this command in your terminal:

```
$ pip install prettyqt
```

This is the preferred method to install PrettyQt, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for PrettyQt can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/phil65/prettyqt
```

Or download the tarball:

```
$ curl  -OL https://github.com/phil65/prettyqt/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use PrettyQt in a project:

```
import prettyqt
```

CHAPTER 4

prettyqt

CHAPTER 5

API documentation

core module

contains QtCore-based classes

class prettyqt.core.**Object**
Bases: PyQt5.QtCore.QObject

class prettyqt.core.**CoreApplication**
Bases: PyQt5.QtCore.QCoreApplication

class prettyqt.core.**IODevice**
Bases: PyQt5.QtCore.QIODevice

class prettyqt.core.**FileDevice**
Bases: PyQt5.QtCore.QFileDevice

class prettyqt.core.**File**
Bases: PyQt5.QtCore.QFile

class prettyqt.core.**Buffer**
Bases: PyQt5.QtCore.QBuffer

class prettyqt.core.**Settings**(**args*, *settings_id=None*)
Bases: PyQt5.QtCore.QSettings

classmethod get_default_format()
returns default settings format

possible values are “native”, “ini”

Return type str

Returns default settings format

get_scope()
returns scope

possible values are “user”, “system”

Return type str

Returns scope

group (*prefix*)
context manager for setting groups

Parameters **prefix** (str) – setting prefix for group

read_array (*prefix*)
context manager for reading arrays

Parameters **prefix** (str) – prefix for settings array

classmethod set_default_format (*fmt*)
sets the default format

possible values are "native", "ini"

Parameters **mode** – the default format to use

Raises ValueError – invalid format

classmethod set_path (*fmt*, *scope*, *path*)
sets the path to the settings file

Parameters

- **fmt** – the default format to use
- **scope** (str) – the scope to use

Raises ValueError – invalid format or scope

value (*key*, *default=None*)

write_array (*prefix*, *size=-1*)
context manager for writing arrays

Parameters

- **prefix** (str) – prefix for settings array
- **size** (int) – size of settings array

class prettyqt.core.**Date**
Bases: PyQt5.QtCore.QDate

class prettyqt.core.**DateTime**
Bases: PyQt5.QtCore.QDateTime

prettyqt.core.**Size**
alias of PyQt5.QtCore.QSize

prettyqt.core.**SizeF**
alias of PyQt5.QtCore.QSizeF

prettyqt.core.**Point**
alias of PyQt5.QtCore.QPoint

prettyqt.core.**PointF**
alias of PyQt5.QtCore.QPointF

class prettyqt.core.**Timer**
Bases: PyQt5.QtCore.QTimer

class prettyqt.core.**Translator**
Bases: PyQt5.QtCore.QTranslator

class prettyqt.core.**Thread**
Bases: PyQt5.QtCore.QThread

prettyqt.core.**Rect**
alias of PyQt5.QtCore.QRect

prettyqt.core.**RectF**
alias of PyQt5.QtCore.QRectF

class prettyqt.core.**MimeData**
Bases: PyQt5.QtCore.QMimeData

class prettyqt.core.**Dir**
Bases: PyQt5.QtCore.QDir

class prettyqt.core.**DirIterator**
Bases: PyQt5.QtCore.QDirIterator

prettyqt.core.**Slot**()
@pyqtSlot(*types, name: Optional[str], result: Optional[str])

This is a decorator applied to Python methods of a QObject that marks them as Qt slots. The non-keyword arguments are the types of the slot arguments and each may be a Python type object or a string specifying a C++ type. name is the name of the slot and defaults to the name of the method. result is type of the value returned by the slot.

prettyqt.core.**Property**
alias of PyQt5.QtCore.pyqtProperty

class prettyqt.core.**RegExp**
Bases: PyQt5.QtCore.QRegExp

class prettyqt.core.**RegularExpression**
Bases: PyQt5.QtCore.QRegularExpression

class prettyqt.core.**Runnable**
Bases: PyQt5.QtCore.QRunnable

class prettyqt.core.**ModelIndex**
Bases: PyQt5.QtCore.QModelIndex

class prettyqt.core.**ThreadPool**
Bases: PyQt5.QtCore.QThreadPool

prettyqt.core.**Signal**
alias of PyQt5.QtCore.pyqtSignal

class prettyqt.core.**AbstractItemModel**
Bases: PyQt5.QtCore.QAbstractItemModel

change_layout()
content manager to change the layout

wraps calls with correct signals emitted at beginning: layoutAboutToBeChanged emitted at end: layoutChanged

reset_model()
content manager to reset the model

wraps calls with correct signals emitted at beginning: beginResetModel emitted at end: endResetModel

class prettyqt.core.**AbstractProxyModel**
Bases: PyQt5.QtCore.QAbstractProxyModel

class prettyqt.core.**AbstractListModel**
Bases: PyQt5.QtCore.QAbstractListModel

class prettyqt.core.**SortFilterProxyModel**
Bases: PyQt5.QtCore.QSortFilterProxyModel

class prettyqt.core.**AbstractTableModel**
Bases: PyQt5.QtCore.QAbstractTableModel

widgets module

contains QtWidgets-based classes

class prettyqt.widgets.**Application**
Bases: PyQt5.QtWidgets.QApplication

classmethod copy_to_clipboard(*text*)
Sets clipboard to supplied text

set_icon(*icon*)
set the icon for the menu

Parameters **icon** (Union[QIcon, str, None]) – icon to use

class prettyqt.widgets.**AbstractSlider**
Bases: PyQt5.QtWidgets.QAbstractSlider

is_horizontal()
check if silder is horizontal

Return type bool

Returns True if horizontal, else False

is_vertical()
check if silder is vertical

Return type bool

Returns True if vertical, else False

scroll_to_max()
scroll to the maximum value of the slider

scroll_to_min()
scroll to the minimum value of the slider

set_horizontal()
set slider orientation to horizontal

set_vertical()
set slider orientation to vertical

class prettyqt.widgets.**AbstractButton**
Bases: PyQt5.QtWidgets.QAbstractButton

set_icon(*icon*)
set the icon for the button

Parameters **icon** (Union[QIcon, str, None]) – icon to use

class prettyqt.widgets.**AbstractSpinBox**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QAbstractSpinBox

get_button_symbols()
returns button symbol type

possible values are “none”, “up_down”, “plus_minus”

Return type str

Returns button symbol type

get_correction_mode()
returns correction mode

possible values are “to_previous”, “to_nearest”

Return type str

Returns correction mode

get_step_type()
returns step type

possible values are “default”, “adaptive”

Return type str

Returns step type

set_button_symbols(*mode*)
sets button symbol type

possible values are “none”, “up_down”, “plus_minus”

Parameters **mode** (str) – button symbol type to use

Raises ValueError – invalid button symbol type

set_correction_mode(*mode*)
sets correction mode

possible values are “to_previous”, “to_nearest”

Parameters **mode** (str) – correction mode to use

Raises ValueError – invalid correction mode

set_step_type(*mode*)
sets step type

possible values are “default”, “adaptive”

Parameters **mode** (str) – step type to use

Raises ValueError – invalid step type

class prettyqt.widgets.**AbstractScrollArea**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QAbstractScrollArea

get_size_adjust_policy()
returns size adjust policy

possible values are “content”, “first_show”, “ignored”

Return type str

Returns size adjust policy

scroll_to_bottom()
scroll to the bottom of the scroll area

scroll_to_top()
scroll to the top of the scroll area

set_horizontal_scrollbar_policy(*mode*)
sets the horizontal scrollbar visibility

possible values are “always_on”, “always_off”, “as_needed”

Parameters **mode** (str) – visibilty to set

Raises ValueError – invalid scrollbar policy

set_horizontal_scrollbar_width(*width*)
sets the horizontal scrollbar width

Parameters **width** (int) – width in pixels

set_scrollbar_policy(*mode*)
sets the policy for both scrollbars

possible values are “always_on”, “always_off”, “as_needed”

Parameters **mode** (str) – visibilty to set

Raises ValueError – invalid scrollbar policy

set_scrollbar_width(*width*)
sets the width for both scrollbars

Parameters **width** (int) – width in pixels

set_size_adjust_policy(*policy*)
set size adjust policy

Valid values are “content”, “first_show”, “ignored”

Parameters **policy** (str) – size adjust policy to use

Raises ValueError – invalid size adjust policy

set_vertical_scrollbar_policy(*mode*)
sets the vertical scrollbar visibility

possible values are “always_on”, “always_off”, “as_needed”

Parameters **mode** (str) – visibilty to set

Raises ValueError – invalid scrollbar policy

set_vertical_scrollbar_width(*width*)
sets the vertical scrollbar width

Parameters **width** (int) – width in pixels

class prettyqt.widgets.**AbstractItemView**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QAbstractItemView

get_selection_behaviour()
returns current selection behaviour

Possible values: “rows”, “columns”, “items”

Return type str

Returns selection behaviour

get_selection_mode()
returns current selection mode

Possible values: “single”, “extended”, “multi” or “none”

Return type str

Returns selection mode

highlight_when_inactive()
also highlight items when widget does not have focus

jump_to_column(*col_num*)
make sure column at given index is visible

scrolls to column at given index

Parameters **col_num** (int) – column to scroll to

num_selected()
returns amount of selected rows

Return type int

Returns amount of selected rows

scroll_to_bottom()
override to use abstractitemview-way of scrolling to bottom

scroll_to_top()
override to use abstractitemview-way of scrolling to top

selectAll()
Override, we dont want to selectAll for too many items for performance reasons

selected_data()
returns generator yielding selected userData

Return type Generator[Any, None, None]

selected_indexes()
returns list of selected indexes in first row

Return type List[QModelIndex]

selected_names()
returns generator yielding item names

Return type Generator[Any, None, None]

selected_rows()
returns generator yielding row nums

Return type Generator[int, None, None]

setModel(*model*)
delete old selection model explicitely, seems to help with memory usage

set_selection_behaviour(*behaviour*)
set selection behaviour for given item view

Allowed values are “rows”, “columns”, “items”

Parameters **behaviour** (str) – selection behaviour to use

Raises ValueError – behaviour does not exist

set_selection_mode(*mode*)
set selection mode for given item view

Allowed values are “single”, “extended”, “multi” or “none”

Parameters **mode** (str) – selection mode to use

Raises ValueError – mode does not exist

toggle_select_all()
select all items from list (deselect when all selected)

class prettyqt.widgets.**MdiSubWindow**
Bases: PyQt5.QtWidgets.QMdiSubWindow

class prettyqt.widgets.**MdiArea**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QMdiArea

get_tab_position()
returns current tab position

Possible values: “north”, “south”, “west”, “east”

Return type str

Returns tab position

get_view_mode()
returns current view mode

Possible values: “default”, “tabbed”

Return type str

Returns view mode

get_window_order()
returns current window order

Possible values: “creation”, “stacking”, “activation”

Return type str

Returns view mode

set_tab_position(*position*)
set tab position for the MDI area

Valid values are “north”, “south”, “west”, “east”

Parameters **position** (str) – tabs position to use

Raises ValueError – tab position does not exist

set_view_mode(*mode*)
set view mode for the MDI area

Valid values are “default”, “tabbed”

Parameters **mode** (str) – view mode to use

Raises ValueError – view mode does not exist

set_window_order(*mode*)
set the window order behaviour for the MDI area

Valid values are “creation”, “stacking”, “activation”

Parameters **mode** (str) – window order behaviour to use

Raises ValueError – window order mode not existing.

class prettyqt.widgets.**ScrollBar**(*orientation='horizontal'*, *parent=None*)
Bases: PyQt5.QtWidgets.QScrollBar

class prettyqt.widgets.**ScrollArea**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QScrollArea

class prettyqt.widgets.**Widget**
Bases: PyQt5.QtWidgets.QWidget

get_contextmenu_policy()
returns current contextmenu policy

Possible values: "none", "prevent", "default", "actions", "custom", "showhide_menu"

Return type str

Returns contextmenu policy

get_modality()
get the current modality modes as a string

Possible values: "modeless", "window", "application"

Return type str

Returns modality mode str

resize(*self*, *QSize*)
resize(self, int, int)

set_contextmenu_policy(*policy*)
set contextmenu policy for given item view

Allowed values are "none", "prevent", "default", "actions", "custom", "showhide_menu"

Parameters **policy** (str) – contextmenu policy to use

Raises ValueError – policy does not exist

set_modality(*modality='window'*)
set modality for the dialog

Valid values for modality: "modeless", "window", "application"

Parameters **modality** (str) – modality for the main window (default: {"window"})

Raises ValueError – modality type does not exist

set_size_policy(*horizontal=None*, *vertical=None*)
sets the sizes policy

possible values for both parameters are "fixed", "minimum", "maximum", "preferred", "expanding", "minimum_expanding" and "ignored"

Parameters

- **horizontal** (Optional[str]) – horizontal size policy
- **vertical** (Optional[str]) – vertical size policy

class prettyqt.widgets.**DesktopWidget**
Bases: PyQt5.QtWidgets.QDesktopWidget

class prettyqt.widgets.**GraphicsItem**
Bases: PyQt5.QtWidgets.QGraphicsItem

class prettyqt.widgets.**Style**
Bases: PyQt5.QtWidgets.QStyle

class prettyqt.widgets.**StyleOption**
Bases: PyQt5.QtWidgets.QStyleOption

class prettyqt.widgets.**SpacerItem**
Bases: PyQt5.QtWidgets.QSpacerItem

class prettyqt.widgets.**SizePolicy**
Bases: PyQt5.QtWidgets.QSizePolicy

get_control_type()
returns control type

possible values are “default”, “buttonbox”, “checkbox”, “combobox”, “frame”, “groupbox”, “label”, “line”, “lineedit”, “pushbutton”, “radiobutton”, “slider”, “spinbox”, “tabwidget”, “toolbutton”

Return type str

Returns control type

get_horizontal_policy()
returns size policy

possible values are “fixed”, “minimum”, “maximum”, “preferred”, “expanding”, “minimum_expanding” and “ignored”

Return type str

Returns horizontal size policy

get_vertical_policy()
returns size policy

possible values are “fixed”, “minimum”, “maximum”, “preferred”, “expanding”, “minimum_expanding” and “ignored”

Return type str

Returns vertical size policy

set_control_type(*mode*)
sets the control type

possible values are “default”, “buttonbox”, “checkbox”, “combobox”, “frame”, “groupbox”, “label”, “line”, “lineedit”, “pushbutton”, “radiobutton”, “slider”, “spinbox”, “tabwidget”, “toolbutton”

Parameters **mode** (str) – control type to set

set_horizontal_policy(*mode*)
sets the horizontal policy

possible values are “fixed”, “minimum”, “maximum”, “preferred”, “expanding”, “minimum_expanding” and “ignored”

Parameters **mode** (str) – policy to set

set_vertical_policy(*mode*)
sets the horizontal policy

possible values are “fixed”, “minimum”, “maximum”, “preferred”, “expanding”, “minimum_expanding” and “ignored”

Parameters **mode** (str) – policy to set

class prettyqt.widgets.**StylePainter**
Bases: PyQt5.QtWidgets.QStylePainter

class prettyqt.widgets.**BaseDialog**
Bases: PyQt5.QtWidgets.QDialog

keyPressEvent(*self*, *QKeyEvent*)

set_icon(*icon*)
set the icon for the menu

Parameters **icon** (Union[QIcon, str, None]) – icon to use

class prettyqt.widgets.**Dialog**(*title=”*, *icon=None*, *parent=None*, *delete_on_close=True*, *layout=None*)
Bases: prettyqt.widgets.dialog.BaseDialog

class prettyqt.widgets.**MessageBox**(*icon=None*, *title=None*, *text=”*, *details=”*, *buttons=None*, *parent=None*)
Bases: PyQt5.QtWidgets.QMessageBox

add_button(*button*)
add a default button

Valid arguments: “none”, “cancel”, “ok”, “save”, “open”, “close”, “discard”, “apply”, “reset”, “restore_defaults”, “help”, “save_all”, “yes”, “yes_to_all”, “no”, “no_to_all”, “abort”, “retry”, “ignore”

Parameters **button** (str) – button to add

Returns created button

Raises ValueError – Button type not available

get_text_format()
returns current text format

Possible values: “rich”, “plain”, “auto”

Return type str

Returns text format

set_icon(*icon*)
set the icon for the menu

Parameters **icon** – icon to use

set_text_format(*text_format*)
set the text format

Allowed values are “rich”, “plain”, “auto”

Parameters **mode** – text format to use

Raises ValueError – text format does not exist

class prettyqt.widgets.**ErrorMessage**
Bases: PyQt5.QtWidgets.QErrorMessage

class prettyqt.widgets.**FileSystemModel**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QFileSystemModel

Class to populate a filesystem treeview

data(*self*, *QModelIndex*, *role: int = Qt.DisplayRole*) → Any

class prettyqt.widgets.**LayoutItem**
Bases: PyQt5.QtWidgets.QLayoutItem

class prettyqt.widgets.**Layout**
Bases: PyQt5.QtWidgets.QLayout

get_size_mode()
returns current size mode

Possible values: “maximum”, “fixed”

Return type str

Returns size mode

set_alignment(*alignment*, *item=None*)
Sets the alignment for widget / layout to alignment and returns true if w is found in this layout (not including child layouts)

Allowed values for alignment: “left”, “right”, “top”, “bottom”

Parameters

- **alignment** (str) – alignment for the layout
- **item** – set alignment for specific child only

Raises ValueError – alignment does not exist

set_size_mode(*mode*)
set the size mode of the layout

Allowed values are “maximum”, “fixed”

Parameters **mode** (str) – size mode for the layout

Raises ValueError – size mode does not exist

class prettyqt.widgets.**FormLayout**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QFormLayout

class prettyqt.widgets.**BoxLayout**(*orientation='horizontal'*, *parent=None*, *margin=None*)
Bases: PyQt5.QtWidgets.QBoxLayout

class prettyqt.widgets.**StackedLayout**
Bases: PyQt5.QtWidgets.QStackedLayout

class prettyqt.widgets.**GridLayout**
Bases: PyQt5.QtWidgets.QGridLayout

class prettyqt.widgets.**ToolBox**
Bases: PyQt5.QtWidgets.QToolBox

class prettyqt.widgets.**Slider**(*orientation='horizontal'*, *parent=None*)
Bases: PyQt5.QtWidgets.QSlider

get_tick_position()
returns tick position

possible values are “none”, “both_sides”, “above”, “below”

Return type str

Returns tick position

set_tick_position(*position*)
sets the tick position for the slider

allowed values are “none”, “both_sides”, “above”, “below”, “left”, “right” for vertical orientation of the slider, “above” equals to “left” and “below” to “right”

Parameters **position** (str) – position for the ticks

class prettyqt.widgets.**StyleOptionSlider**
Bases: PyQt5.QtWidgets.QStyleOptionSlider

is_horizontal()
check if silder is horizontal

Return type bool

Returns True if horizontal, else False

is_vertical()
check if silder is vertical

Return type bool

Returns True if vertical, else False

set_horizontal()
set slider orientation to horizontal

set_vertical()
set slider orientation to vertical

class prettyqt.widgets.**Frame**
Bases: PyQt5.QtWidgets.QFrame

class prettyqt.widgets.**ListWidgetItem**
Bases: PyQt5.QtWidgets.QListWidgetItem

get_checkstate()
returns checkstate

possible values are “unchecked”, “partial”, “checked”

Return type str

Returns checkstate

set_checkstate(*state*)
set checkstate of the checkbox

valid values are: unchecked, partial, checked

Parameters **state** (str) – checkstate to use

Raises ValueError – invalid checkstate

set_icon(*icon*)
set the icon for the action

Parameters **icon** (Union[QIcon, str, None]) – icon to use

class prettyqt.widgets.**TreeWidgetItem**
Bases: PyQt5.QtWidgets.QTreeWidgetItem

get_checkstate()
returns checkstate

possible values are “unchecked”, “partial”, “checked”

Return type str

Returns checkstate

set_checkstate(*state*)
set checkstate of the checkbox

valid values are: unchecked, partial, checked

Parameters **state** (str) – checkstate to use

Raises ValueError – invalid checkstate

set_icon(*icon*)
set the icon for the action

Parameters **icon** (Union[QIcon, str, None]) – icon to use

class prettyqt.widgets.**Action**(*text=”*, *icon=None*, *parent=None*, *shortcut=”*, *tooltip=”*)
Bases: PyQt5.QtWidgets.QAction

get_priority()
returns current priority

Possible values: “low”, “normal”, “high”

Return type str

Returns priority

get_shortcut_context()
returns shortcut context

Possible values: “widget”, “widget_with_children”, “window”, “application”

Return type str

Returns shortcut context

set_icon(*icon*)
set the icon for the action

Parameters **icon** (Union[QIcon, str, None]) – icon to use

set_priority(*priority*)
set priority of the action

Allowed values are “low”, “normal”, “high”

Parameters **mode** – priority for the action

Raises ValueError – priority does not exist

set_shortcut_context(*context*)
set shortcut context

Allowed values are “widget”, “widget_with_children”, “window”, “application”

Parameters **mode** – shortcut context

Raises ValueError – shortcut context does not exist

class prettyqt.widgets.**WidgetAction**(*text=''*, *icon=None*, *parent=None*, *shortcut=''*, *tooltip=''*)

Bases: PyQt5.QtWidgets.QWidgetAction

class prettyqt.widgets.**ToolButton**

Bases: PyQt5.QtWidgets.QToolButton

get_arrow_type()

returns arrow type

possible values are "none", "up", "down", "left", "right"

Return type str

Returns arrow type

get_popup_mode()

returns popup mode

possible values are "delayed", "menu_button", "instant"

Return type str

Returns popup mode

set_arrow_type(*mode*)

sets the arrow type of the toolbutton

valid values are: "none", "up", "down", "left", "right"

Parameters **mode** (str) – arrow type to use

Raises ValueError – invalid arrow type

set_popup_mode(*mode*)

sets the popup mode of the toolbutton

valid values are: "delayed", "menu_button", "instant"

Parameters **mode** (str) – popup mode to use

Raises ValueError – invalid popup mode

class prettyqt.widgets.**ToolTip**

Bases: PyQt5.QtWidgets.QToolTip

class prettyqt.widgets.**Menu**(*title=''*, *icon=None*, *parent=None*)

Bases: PyQt5.QtWidgets.QMenu

add_action(*label*, *callback=None*, *icon=None*, *checkable=False*, *checked=False*, *shortcut=None*, *status_tip=None*)

Add an action to the menu

Parameters

- **label** (Union[str, Action]) – Label for button
- **callback** (Optional[Callable]) – gets called when action is triggered
- **icon** (Optional[Any]) – icon for button (default: {None})
- **checkable** (bool) – as checkbox button (default: {False})
- **checked** (bool) – if checkable, turn on by default (default: {False})
- **shortcut** (Optional[str]) – Shortcut for action (a) (default: {None})
- **status_tip** (Optional[str]) – Status tip to be shown in status bar (default: {None})

Return type Action

Returns Action added to menu

set_icon(*icon*)
set the icon for the menu

Parameters **icon** (Union[QIcon, str, None]) – icon to use

class prettyqt.widgets.**MenuBar**
Bases: PyQt5.QtWidgets.QMenuBar

class prettyqt.widgets.**StatusBar**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QStatusBar

class prettyqt.widgets.**TabWidget**(*parent=None*, *closable=False*, *detachable=False*)
Bases: PyQt5.QtWidgets.QTabWidget

Widget for managing the tabs section

attach_tab(*widget*, *name*, *icon*, *insert_at=None*)
Re-attach the tab by removing the content from the DetachedTab window, closing it, and placing the content back into the DetachableTabWidget

@param widget the content widget from the DetachedTab window @param name the name of the detached tab @param icon the window icon for the detached tab @param insert_at insert the re-attached tab at the given index

close_detached_tabs()
Close all tabs that are currently detached

detach_tab(*index*, *point*)
Detach the tab by removing it's contents and placing them in a DetachedTab window

@param index index location of the tab to be detached @param point screen pos for creating the new DetachedTab window

get_tab_shape()
returns tab shape

possible values are "roundes", "triangular"

Return type str

Returns tab shape

open_widget(*widget*, *title='Unnamed'*)
create a tab containing delivered widget

set_tab_shape(*shape*)
set tab shape for the tabwidget

Valid values are "rounded" and "triangular"

Parameters **shape** (str) – tab shape to use

Raises ValueError – tab shape does not exist

class prettyqt.widgets.**TabBar**(*parent=None*)
Bases: PyQt5.QtWidgets.QTabBar

get_elide_mode()
returns elide mode

possible values are "left", "right", "middle", "none"

Return type str

Returns elide mode

get_remove_behaviour()
returns remove behaviour

possible values are “left_tab”, “right_tab”, “previous_tab”

Return type str

Returns remove behaviour

mouseDoubleClickEvent(*self*, *QMouseEvent*)

set_elide_mode(*mode*)
set elide mode

Valid values are “left”, “right”, “middle”, “none”

Parameters **policy** – elide mode to use

Raises ValueError – invalid elide mode

set_remove_behaviour(*mode*)
sets the remove hehaviour

What tab should be set as current when removeTab is called if the removed tab is also the current tab. Possible values: left, right, previous :type mode: str :param mode: new remove behaviour

class prettyqt.widgets.**ToolBar**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QToolBar

get_style()
returns current style

Possible values: “icon”, “text”, “text_below_icon”, “text_beside_icon”

Return type str

Returns style

is_area_allowed(*area*)
check if toolbar is allowed at specified area

Valid values for area: “left”, “right”, “top”, “bottom”

Parameters **area** (str) – area of the toolbar

Raises ValueError – area does not exist

class prettyqt.widgets.**HeaderView**(*orientation=None*, *parent=None*)
Bases: PyQt5.QtWidgets.QHeaderView

contextMenuEvent(*event*)
context menu for our files tree

class prettyqt.widgets.**DockWidget**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QDockWidget

Customized DockWidget class contains a custom TitleBar with maximise button

class prettyqt.widgets.**Label**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QLabel

get_text_format()
returns current text format

Possible values: “rich”, “plain”, “auto”

Return type str

Returns text format

get_text_interaction()
returns current text interaction mode

Possible values: “none”, “by_mouse”, “by_keyboard”

Return type str

Returns text interaction mode

set_text_format(*text_format*)
set the text format

Allowed values are “rich”, “plain”, “auto”

Parameters **mode** – text format to use

Raises ValueError – text format does not exist

set_text_interaction(*interaction_type*)
set the text interaction mode

Allowed values are “none”, “by_mouse”, “by_keyboard”

Parameters **mode** – text interaction mode to use

Raises ValueError – text interaction mode does not exist

class prettyqt.widgets.**PushButton**(*label=None*, *parent=None*, *callback=None*)
Bases: PyQt5.QtWidgets.QPushButton

class prettyqt.widgets.**CommandLinkButton**(*label=None*, *parent=None*, *callback=None*)
Bases: PyQt5.QtWidgets.QCommandLinkButton

class prettyqt.widgets.**RadioButton**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QRadioButton

class prettyqt.widgets.**ComboBox**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QComboBox

get_insert_policy()
returns insert policy

possible values are “no_insert”, “top”, “current”, “bottom”, “after_current”, “before_current”, “alphabetically”

Return type str

Returns insert policy

get_size_adjust_policy()
returns size adjust policy

possible values are “content”, “first_show”, “min_length”, “min_length_with_icon”

Return type str

Returns size adjust policy

set_insert_policy(*policy*)
set insert policy

valid values are “no_insert”, “top”, “current”, “bottom”, “after_current”, “before_current”, “alphabetically”

Parameters **policy** (str) – insert policy to use

Raises ValueError – invalid insert policy

set_size_adjust_policy(*policy*)
set size adjust policy

possible values are “content”, “first_show”, “min_length”, “min_length_with_icon”

Parameters **policy** (str) – size adjust policy to use

Raises ValueError – invalid size adjust policy

class prettyqt.widgets.**SpinBox**(*parent=None*, *min_value=None*, *max_value=None*, *default_value=None*)
Bases: PyQt5.QtWidgets.QSpinBox

class prettyqt.widgets.**DoubleSpinBox**(*parent=None*, *min_value=None*, *max_value=None*, *default_value=None*)
Bases: PyQt5.QtWidgets.QDoubleSpinBox

class prettyqt.widgets.**CheckBox**(*label=''*, *parent=None*, *checked=False*)
Bases: PyQt5.QtWidgets.QCheckBox

get_checkstate()
returns checkstate

possible values are “unchecked”, “partial”, “checked”

Return type bool

Returns checkstate

set_checkstate(*state*)
set checkstate of the checkbox

valid values are: unchecked, partial, checked

Parameters **state** (str) – checkstate to use

Raises ValueError – invalid checkstate

class prettyqt.widgets.**LineEdit**(*default_value=''*, *read_only=False*, *parent=None*)
Bases: PyQt5.QtWidgets.QLineEdit

font()

Return type Font

get_echo_mode()
returns echo mode

possible values are “normal”, “no_echo”, “password”, “echo_on_edit”

Return type str

Returns echo mode

set_echo_mode(*mode*)
set echo mode

Valid values are “normal”, “no_echo”, “password”, “echo_on_edit”

Parameters **policy** – echo mode to use

Raises ValueError – invalid echo mode

set_read_only(*value=True*)
set test to read only

Parameters **value** (bool) – True, for read-only, otherwise False

class prettyqt.widgets.**KeySequenceEdit**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QKeySequenceEdit

class prettyqt.widgets.**TextEdit**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QTextEdit

class prettyqt.widgets.**DateEdit**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QDateEdit

class prettyqt.widgets.**TimeEdit**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QTimeEdit

class prettyqt.widgets.**DateTimeEdit**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QDateTimeEdit

class prettyqt.widgets.**CalendarWidget**
Bases: PyQt5.QtWidgets.QCalendarWidget

class prettyqt.widgets.**PlainTextEdit**(*text=''*, *parent=None*, *read_only=False*)
Bases: PyQt5.QtWidgets.QPlainTextEdit

set_read_only(*value=True*)
set test to read only

Parameters **value** (bool) – True, for read-only, otherwise False

class prettyqt.widgets.**TextBrowser**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QTextBrowser

class prettyqt.widgets.**ProgressBar**(*text_visible=True*, *parent=None*)
Bases: PyQt5.QtWidgets.QProgressBar

Progress dialog

wrapper for QtWidgets.QProgressBar

get_alignment()
returns current alignment

Possible values: "left", "right", "top", "bottom"

Return type str

Returns alignment

get_text_direction()
returns current text direction

Possible values are "top_to_bottom", "bottom_to_top"

Return type str

Returns Text direction

set_alignment(*alignment*)
set the alignment of the layout

Allowed values are "left", "right", "top", "bottom"

Parameters **mode** – alignment for the layout

Raises ValueError – alignment does not exist

set_text_direction(*text_direction*)
set the text direction of the layout

Allowed values are "top_to_bottom", "bottom_to_top"

Parameters **mode** – text direction for the layout

Raises ValueError – text direction does not exist

class prettyqt.widgets.**ColumnView**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QColumnView

class prettyqt.widgets.**ListView**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QListView

get_view_mode()
returns view mode

possible values are "list", "icon"

Return type str

Returns view mode

set_view_mode(*mode*)
set view mode

possible values are "list", "icon"

Parameters **mode** (str) – view mode to use

Raises ValueError – invalid view mode

class prettyqt.widgets.**ListWidget**(*parent=None*, *selection_mode='single'*)
Bases: PyQt5.QtWidgets.QListWidget

class prettyqt.widgets.**TreeView**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QTreeView

class prettyqt.widgets.**TreeWidget**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QTreeWidget

class prettyqt.widgets.**TableView**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QTableView

class prettyqt.widgets.**TableWidget**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QTableWidget

class prettyqt.widgets.**SplashScreen**(*path*, *width=None*)
Bases: PyQt5.QtWidgets.QSplashScreen

class prettyqt.widgets.**ProgressDialog**(*parent=None*)
Bases: PyQt5.QtWidgets.QProgressDialog

Progress dialog

wrapper for QtWidgets.QProgressDialog

class prettyqt.widgets.**FontDialog**
Bases: PyQt5.QtWidgets.QFontDialog

class prettyqt.widgets.**FileDialog**(*path=None*, *mode='open'*, *caption=''*, *path_id=None*, *extension_filter=None*, *parent=None*)

Bases: PyQt5.QtWidgets.QFileDialog

simple dialog used to display some widget

directory()
return current directory

returns current directory level as a Pathlib object

Return type Path

Returns Pathlib object

get_accept_mode()
returns accept mode

possible values are "save", "open"

Return type str

Returns accept mode

get_file_mode()
returns file mode

possible values are "existing_file", "existing_files", "any_file", "directory"

Return type str

Returns file mode

get_label_text(*label*)
returns label text

possible values are "look_in", "filename", "filetype", "accept", "reject"

Return type str

Returns label text

get_view_mode()
returns view mode

possible values are "detail", "list"

Return type str

Returns view mode

set_accept_mode(*mode*)
set accept mode

possible values are "save", "open"

Parameters **mode** (str) – accept mode to use

Raises ValueError – invalid accept mode

set_file_mode(*mode*)
sets the file mode of the dialog

allowed values are "existing_file", "existing_files", "any_file" "directory"

Parameters **mode** (str) – mode to use

set_filter(*extension_dict*)
set filter based on given dictionary

dict must contain "'name': ['.ext1', '.ext2']" as key-value pairs

Parameters **extension_dict** (dict) – filter dictionary

set_label_text(*label*, *text*)
sets the label text for button label

possible values for label are "look_in", "filename", "filetype", "accept", "reject"

Parameters

- **label** (str) – button to set text for
- **text** (str) – text to use

set_view_mode(*mode*)
set view mode

possible values are "detail", "list"

Parameters **mode** (str) – view mode to use

Raises ValueError – invalid view mode

class prettyqt.widgets.**ColorDialog**
Bases: PyQt5.QtWidgets.QColorDialog

class prettyqt.widgets.**InputDialog**
Bases: PyQt5.QtWidgets.QInputDialog

class prettyqt.widgets.**DialogButtonBox**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QDialogButtonBox

add_button(*button*, *role='accept'*, *callback=None*)
add a button

Parameters

- **button** – button to add
- **role** – role of the button
- **callback** – function to call when button gets clicked

Returns created button

Raises ValueError – Button type not available

add_default_button(*button*, *callback=None*)
add a default button

Valid arguments: "cancel", "ok", "save", "open", "close", "discard", "apply", "reset", "restore_defaults", "help", "save_all", "yes", "yes_to_all", "no", "no_to_all", "abort", "retry", "ignore"

Parameters

- **button** (str) – button to add
- **callback** – function to call when button gets clicked

Returns created button

Raises ValueError – Button type not available

classmethod create(*self*, *window: sip.voidptr = 0*, *initializeWindow: bool = True*, *destroyOldWindow: bool = True*)

get_orientation()
returns current orientation

Possible values: "horizontal", "vertical"

Return type str

Returns orientation

set_orientation(*orientation*)
set the orientation of the button box

Allowed values are "horizontal", "vertical"

Parameters **mode** – orientation for the button box

Raises ValueError – orientation does not exist

class prettyqt.widgets.**ButtonGroup**
Bases: PyQt5.QtWidgets.QButtonGroup

class prettyqt.widgets.**GroupBox**(*title=''*, *checkable=False*, *parent=None*)
Bases: PyQt5.QtWidgets.QGroupBox

GroupBox widget

A group box provides a frame, a title on top, a keyboard shortcut, and displays various other widgets inside itself. The keyboard shortcut moves keyboard focus to one of the group box's child widgets.

class prettyqt.widgets.**Splitter**(*orientation='horizontal'*, *parent=None*)
Bases: PyQt5.QtWidgets.QSplitter

get_orientation()
returns current orientation

Possible values: "horizontal", "vertical"

Return type str

Returns orientation

set_orientation(*orientation*)
set the orientation of the splitter

Allowed values are "horizontal", "vertical"

Parameters **mode** – orientation for the splitter

Raises ValueError – orientation does not exist

class prettyqt.widgets.**Wizard**
Bases: PyQt5.QtWidgets.QWizard

class prettyqt.widgets.**WizardPage**
Bases: PyQt5.QtWidgets.QWizardPage

class prettyqt.widgets.**MainWindow**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QMainWindow

Class for our mainWindow includes all docks, a centralwidget and a toolbar

add_toolbar(*toolbar*, *position='top'*)
adds a toolbar to the mainmenu at specified area

Valid values for position: "left", "right", "top", "bottom"

Parameters

- **toolbar** – toolbar to use
- **position** (str) – position of the toolbar

Raises ValueError – position does not exist

add_toolbar_break(*position='top'*)
Adds a toolbar break to the given area after all the other objects that are present.

Valid values for position: "left", "right", "top", "bottom"

Parameters **position** (str) – position of the toolbar

Raises ValueError – position does not exist

closeEvent(*event*)
override, gets executed when app gets closed. saves GUI settings

createPopupMenu(*self*) → QMenu

set_icon(*icon*)
set the icon for the menu

Parameters **icon** (Union[QIcon, str, None]) – icon to use

toggle_fullscreen()
toggle between fullscreen and regular size

class prettyqt.widgets.**AbstractItemDelegate**
Bases: PyQt5.QtWidgets.QAbstractItemDelegate

class prettyqt.widgets.**ItemDelegate**
Bases: PyQt5.QtWidgets.QItemDelegate

class prettyqt.widgets.**StyledItemDelegate**
Bases: PyQt5.QtWidgets.QStyledItemDelegate

CHAPTER 6

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

6.1 Types of Contributions

6.1.1 Report Bugs

Report bugs at https://github.com/phil65/prettyqt/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

6.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

6.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

6.1.4 Write Documentation

PrettyQt could always use more documentation, whether as part of the official PrettyQt docs, in docstrings, or even on the web in blog posts, articles, and such.

6.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/phil65/prettyqt/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

6.2 Get Started!

Ready to contribute? Here's how to set up *prettyqt* for local development.

1. Fork the *prettyqt* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/prettyqt.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv prettyqt
$ cd prettyqt/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 prettyqt tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

6.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.4, 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/phil65/prettyqt/pull_requests and make sure that the tests pass for all supported Python versions.

6.4 Tips

To run a subset of tests:

```
$ py.test tests.test_prettyqt
```

6.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 7

Credits

7.1 Development Lead

- Philipp Temminghoff <phil65@kodi.tv>

7.2 Contributors

None yet. Why not be the first?

CHAPTER 8

History

8.1 0.6.0 (2019-03-24)

- First release on PyPI.

CHAPTER 9

main

core module

contains QtCore-based classes

class prettyqt.core.**Object**
Bases: PyQt5.QtCore.QObject

class prettyqt.core.**CoreApplication**
Bases: PyQt5.QtCore.QCoreApplication

class prettyqt.core.**IODevice**
Bases: PyQt5.QtCore.QIODevice

class prettyqt.core.**FileDevice**
Bases: PyQt5.QtCore.QFileDevice

class prettyqt.core.**File**
Bases: PyQt5.QtCore.QFile

class prettyqt.core.**Buffer**
Bases: PyQt5.QtCore.QBuffer

class prettyqt.core.**Settings**(**args*, *settings_id=None*)
Bases: PyQt5.QtCore.QSettings

classmethod get_default_format()
returns default settings format

possible values are “native”, “ini”

Return type str

Returns default settings format

get_scope()
returns scope

possible values are “user”, “system”

Return type str

Returns scope

group (*prefix*)
context manager for setting groups

Parameters **prefix** (str) – setting prefix for group

read_array (*prefix*)
context manager for reading arrays

Parameters **prefix** (str) – prefix for settings array

classmethod set_default_format (*fmt*)
sets the default format

possible values are "native", "ini"

Parameters **mode** – the default format to use

Raises ValueError – invalid format

classmethod set_path (*fmt*, *scope*, *path*)
sets the path to the settings file

Parameters

- **fmt** – the default format to use
- **scope** (str) – the scope to use

Raises ValueError – invalid format or scope

value (*key*, *default=None*)

write_array (*prefix*, *size=-1*)
context manager for writing arrays

Parameters

- **prefix** (str) – prefix for settings array
- **size** (int) – size of settings array

class prettyqt.core.**Date**
Bases: PyQt5.QtCore.QDate

class prettyqt.core.**DateTime**
Bases: PyQt5.QtCore.QDateTime

prettyqt.core.**Size**
alias of PyQt5.QtCore.QSize

prettyqt.core.**SizeF**
alias of PyQt5.QtCore.QSizeF

prettyqt.core.**Point**
alias of PyQt5.QtCore.QPoint

prettyqt.core.**PointF**
alias of PyQt5.QtCore.QPointF

class prettyqt.core.**Timer**
Bases: PyQt5.QtCore.QTimer

class prettyqt.core.**Translator**
Bases: PyQt5.QtCore.QTranslator

class prettyqt.core.**Thread**
Bases: PyQt5.QtCore.QThread

prettyqt.core.**Rect**
alias of PyQt5.QtCore.QRect

prettyqt.core.**RectF**
alias of PyQt5.QtCore.QRectF

class prettyqt.core.**MimeData**
Bases: PyQt5.QtCore.QMimeData

class prettyqt.core.**Dir**
Bases: PyQt5.QtCore.QDir

class prettyqt.core.**DirIterator**
Bases: PyQt5.QtCore.QDirIterator

prettyqt.core.**Slot**()
@pyqtSlot(*types, name: Optional[str], result: Optional[str])

This is a decorator applied to Python methods of a QObject that marks them as Qt slots. The non-keyword arguments are the types of the slot arguments and each may be a Python type object or a string specifying a C++ type. name is the name of the slot and defaults to the name of the method. result is type of the value returned by the slot.

prettyqt.core.**Property**
alias of PyQt5.QtCore.pyqtProperty

class prettyqt.core.**RegExp**
Bases: PyQt5.QtCore.QRegExp

class prettyqt.core.**RegularExpression**
Bases: PyQt5.QtCore.QRegularExpression

class prettyqt.core.**Runnable**
Bases: PyQt5.QtCore.QRunnable

class prettyqt.core.**ModelIndex**
Bases: PyQt5.QtCore.QModelIndex

class prettyqt.core.**ThreadPool**
Bases: PyQt5.QtCore.QThreadPool

prettyqt.core.**Signal**
alias of PyQt5.QtCore.pyqtSignal

class prettyqt.core.**AbstractItemModel**
Bases: PyQt5.QtCore.QAbstractItemModel

change_layout()
content manager to change the layout

wraps calls with correct signals emitted at beginning: layoutAboutToBeChanged emitted at end: layoutChanged

reset_model()
content manager to reset the model

wraps calls with correct signals emitted at beginning: beginResetModel emitted at end: endResetModel

class prettyqt.core.**AbstractProxyModel**
Bases: PyQt5.QtCore.QAbstractProxyModel

class prettyqt.core.**AbstractListModel**
Bases: PyQt5.QtCore.QAbstractListModel

class prettyqt.core.**SortFilterProxyModel**
Bases: PyQt5.QtCore.QSortFilterProxyModel

class prettyqt.core.**AbstractTableModel**
Bases: PyQt5.QtCore.QAbstractTableModel

widgets module

contains QtWidgets-based classes

class prettyqt.widgets.**Application**
Bases: PyQt5.QtWidgets.QApplication

classmethod copy_to_clipboard(*text*)
Sets clipboard to supplied text

set_icon(*icon*)
set the icon for the menu

Parameters **icon** (Union[QIcon, str, None]) – icon to use

class prettyqt.widgets.**AbstractSlider**
Bases: PyQt5.QtWidgets.QAbstractSlider

is_horizontal()
check if silder is horizontal

Return type bool

Returns True if horizontal, else False

is_vertical()
check if silder is vertical

Return type bool

Returns True if vertical, else False

scroll_to_max()
scroll to the maximum value of the slider

scroll_to_min()
scroll to the minimum value of the slider

set_horizontal()
set slider orientation to horizontal

set_vertical()
set slider orientation to vertical

class prettyqt.widgets.**AbstractButton**
Bases: PyQt5.QtWidgets.QAbstractButton

set_icon(*icon*)
set the icon for the button

Parameters **icon** (Union[QIcon, str, None]) – icon to use

class prettyqt.widgets.**AbstractSpinBox**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QAbstractSpinBox

get_button_symbols()
returns button symbol type

possible values are “none”, “up_down”, “plus_minus”

Return type str

Returns button symbol type

get_correction_mode()
returns correction mode

possible values are “to_previous”, “to_nearest”

Return type str

Returns correction mode

get_step_type()
returns step type

possible values are “default”, “adaptive”

Return type str

Returns step type

set_button_symbols(*mode*)
sets button symbol type

possible values are “none”, “up_down”, “plus_minus”

Parameters **mode** (str) – button symbol type to use

Raises ValueError – invalid button symbol type

set_correction_mode(*mode*)
sets correction mode

possible values are “to_previous”, “to_nearest”

Parameters **mode** (str) – correction mode to use

Raises ValueError – invalid correction mode

set_step_type(*mode*)
sets step type

possible values are “default”, “adaptive”

Parameters **mode** (str) – step type to use

Raises ValueError – invalid step type

class prettyqt.widgets.**AbstractScrollArea**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QAbstractScrollArea

get_size_adjust_policy()
returns size adjust policy

possible values are “content”, “first_show”, “ignored”

Return type str

Returns size adjust policy

scroll_to_bottom()
scroll to the bottom of the scroll area

scroll_to_top()
scroll to the top of the scroll area

set_horizontal_scrollbar_policy(*mode*)
sets the horizontal scrollbar visibility

possible values are “always_on”, “always_off”, “as_needed”

Parameters **mode** (str) – visibilty to set

Raises ValueError – invalid scrollbar policy

set_horizontal_scrollbar_width(*width*)
sets the horizontal scrollbar width

Parameters **width** (int) – width in pixels

set_scrollbar_policy(*mode*)
sets the policy for both scrollbars

possible values are “always_on”, “always_off”, “as_needed”

Parameters **mode** (str) – visibilty to set

Raises ValueError – invalid scrollbar policy

set_scrollbar_width(*width*)
sets the width for both scrollbars

Parameters **width** (int) – width in pixels

set_size_adjust_policy(*policy*)
set size adjust policy

Valid values are “content”, “first_show”, “ignored”

Parameters **policy** (str) – size adjust policy to use

Raises ValueError – invalid size adjust policy

set_vertical_scrollbar_policy(*mode*)
sets the vertical scrollbar visibility

possible values are “always_on”, “always_off”, “as_needed”

Parameters **mode** (str) – visibilty to set

Raises ValueError – invalid scrollbar policy

set_vertical_scrollbar_width(*width*)
sets the vertical scrollbar width

Parameters **width** (int) – width in pixels

class prettyqt.widgets.**AbstractItemView**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QAbstractItemView

get_selection_behaviour()
returns current selection behaviour

Possible values: “rows”, “columns”, “items”

Return type str

Returns selection behaviour

get_selection_mode()
returns current selection mode

Possible values: “single”, “extended”, “multi” or “none”

Return type str

Returns selection mode

highlight_when_inactive()
also highlight items when widget does not have focus

jump_to_column(*col_num*)
make sure column at given index is visible

scrolls to column at given index

Parameters **col_num** (int) – column to scroll to

num_selected()
returns amount of selected rows

Return type int

Returns amount of selected rows

scroll_to_bottom()
override to use abstractitemview-way of scrolling to bottom

scroll_to_top()
override to use abstractitemview-way of scrolling to top

selectAll()
Override, we dont want to selectAll for too many items for performance reasons

selected_data()
returns generator yielding selected userData

Return type Generator[Any, None, None]

selected_indexes()
returns list of selected indexes in first row

Return type List[QModelIndex]

selected_names()
returns generator yielding item names

Return type Generator[Any, None, None]

selected_rows()
returns generator yielding row nums

Return type Generator[int, None, None]

setModel(*model*)
delete old selection model explicitely, seems to help with memory usage

set_selection_behaviour(*behaviour*)
set selection behaviour for given item view

Allowed values are “rows”, “columns”, “items”

Parameters **behaviour** (str) – selection behaviour to use

Raises ValueError – behaviour does not exist

set_selection_mode(*mode*)
set selection mode for given item view

Allowed values are “single”, “extended”, “multi” or “none”

Parameters **mode** (str) – selection mode to use

Raises ValueError – mode does not exist

toggle_select_all()
select all items from list (deselect when all selected)

class prettyqt.widgets.**MdiSubWindow**
Bases: PyQt5.QtWidgets.QMdiSubWindow

class prettyqt.widgets.**MdiArea**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QMdiArea

get_tab_position()
returns current tab position

Possible values: “north”, “south”, “west”, “east”

Return type str

Returns tab position

get_view_mode()
returns current view mode

Possible values: “default”, “tabbed”

Return type str

Returns view mode

get_window_order()
returns current window order

Possible values: “creation”, “stacking”, “activation”

Return type str

Returns view mode

set_tab_position(*position*)
set tab position for the MDI area

Valid values are “north”, “south”, “west”, “east”

Parameters **position** (str) – tabs position to use

Raises ValueError – tab position does not exist

set_view_mode(*mode*)
set view mode for the MDI area

Valid values are “default”, “tabbed”

Parameters **mode** (str) – view mode to use

Raises ValueError – view mode does not exist

set_window_order(*mode*)
set the window order behaviour for the MDI area

Valid values are “creation”, “stacking”, “activation”

Parameters **mode** (str) – window order behaviour to use

Raises ValueError – window order mode not existing.

class prettyqt.widgets.**ScrollBar**(*orientation='horizontal'*, *parent=None*)
Bases: PyQt5.QtWidgets.QScrollBar

class prettyqt.widgets.**ScrollArea**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QScrollArea

class prettyqt.widgets.**Widget**
Bases: PyQt5.QtWidgets.QWidget

get_contextmenu_policy()
returns current contextmenu policy

Possible values: "none", "prevent", "default", "actions", "custom", "showhide_menu"

Return type str

Returns contextmenu policy

get_modality()
get the current modality modes as a string

Possible values: "modeless", "window", "application"

Return type str

Returns modality mode str

resize(*self*, *QSize*)
resize(self, int, int)

set_contextmenu_policy(*policy*)
set contextmenu policy for given item view

Allowed values are "none", "prevent", "default", "actions", "custom", "showhide_menu"

Parameters **policy** (str) – contextmenu policy to use

Raises ValueError – policy does not exist

set_modality(*modality='window'*)
set modality for the dialog

Valid values for modality: "modeless", "window", "application"

Parameters **modality** (str) – modality for the main window (default: {"window"})

Raises ValueError – modality type does not exist

set_size_policy(*horizontal=None*, *vertical=None*)
sets the sizes policy

possible values for both parameters are "fixed", "minimum", "maximum", "preferred", "expanding", "minimum_expanding" and "ignored"

Parameters

- **horizontal** (Optional[str]) – horizontal size policy
- **vertical** (Optional[str]) – vertical size policy

class prettyqt.widgets.**DesktopWidget**
Bases: PyQt5.QtWidgets.QDesktopWidget

class prettyqt.widgets.**GraphicsItem**
Bases: PyQt5.QtWidgets.QGraphicsItem

class prettyqt.widgets.**Style**
Bases: PyQt5.QtWidgets.QStyle

class prettyqt.widgets.**StyleOption**
Bases: PyQt5.QtWidgets.QStyleOption

class prettyqt.widgets.**SpacerItem**
Bases: PyQt5.QtWidgets.QSpacerItem

class prettyqt.widgets.**SizePolicy**
Bases: PyQt5.QtWidgets.QSizePolicy

get_control_type()
returns control type

possible values are “default”, “buttonbox”, “checkbox”, “combobox”, “frame”, “groupbox”, “label”, “line”, “lineedit”, “pushbutton”, “radiobutton”, “slider”, “spinbox”, “tabwidget”, “toolbutton”

Return type str

Returns control type

get_horizontal_policy()
returns size policy

possible values are “fixed”, “minimum”, “maximum”, “preferred”, “expanding”, “minimum_expanding” and “ignored”

Return type str

Returns horizontal size policy

get_vertical_policy()
returns size policy

possible values are “fixed”, “minimum”, “maximum”, “preferred”, “expanding”, “minimum_expanding” and “ignored”

Return type str

Returns vertical size policy

set_control_type(*mode*)
sets the control type

possible values are “default”, “buttonbox”, “checkbox”, “combobox”, “frame”, “groupbox”, “label”, “line”, “lineedit”, “pushbutton”, “radiobutton”, “slider”, “spinbox”, “tabwidget”, “toolbutton”

Parameters **mode** (str) – control type to set

set_horizontal_policy(*mode*)
sets the horizontal policy

possible values are “fixed”, “minimum”, “maximum”, “preferred”, “expanding”, “minimum_expanding” and “ignored”

Parameters **mode** (str) – policy to set

set_vertical_policy(*mode*)
sets the horizontal policy

possible values are “fixed”, “minimum”, “maximum”, “preferred”, “expanding”, “minimum_expanding” and “ignored”

Parameters **mode** (str) – policy to set

class prettyqt.widgets.**StylePainter**
Bases: PyQt5.QtWidgets.QStylePainter

class prettyqt.widgets.**BaseDialog**
Bases: PyQt5.QtWidgets.QDialog

keyPressEvent(*self*, *QKeyEvent*)

set_icon(*icon*)
set the icon for the menu

Parameters **icon** (Union[QIcon, str, None]) – icon to use

class prettyqt.widgets.**Dialog**(*title=”*, *icon=None*, *parent=None*, *delete_on_close=True*, *layout=None*)
Bases: prettyqt.widgets.dialog.BaseDialog

class prettyqt.widgets.**MessageBox**(*icon=None*, *title=None*, *text=”*, *details=”*, *buttons=None*, *parent=None*)
Bases: PyQt5.QtWidgets.QMessageBox

add_button(*button*)
add a default button

Valid arguments: “none”, “cancel”, “ok”, “save”, “open”, “close”, “discard”, “apply”, “reset”, “restore_defaults”, “help”, “save_all”, “yes”, “yes_to_all”, “no”, “no_to_all”, “abort”, “retry”, “ignore”

Parameters **button** (str) – button to add

Returns created button

Raises ValueError – Button type not available

get_text_format()
returns current text format

Possible values: “rich”, “plain”, “auto”

Return type str

Returns text format

set_icon(*icon*)
set the icon for the menu

Parameters **icon** – icon to use

set_text_format(*text_format*)
set the text format

Allowed values are “rich”, “plain”, “auto”

Parameters **mode** – text format to use

Raises ValueError – text format does not exist

class prettyqt.widgets.**ErrorMessage**
Bases: PyQt5.QtWidgets.QErrorMessage

class prettyqt.widgets.**FileSystemModel**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QFileSystemModel

Class to populate a filesystem treeview

data(*self*, *QModelIndex*, *role: int = Qt.DisplayRole*) → Any

class prettyqt.widgets.**LayoutItem**
Bases: PyQt5.QtWidgets.QLayoutItem

class prettyqt.widgets.**Layout**
Bases: PyQt5.QtWidgets.QLayout

get_size_mode()
returns current size mode

Possible values: “maximum”, “fixed”

Return type str

Returns size mode

set_alignment(*alignment*, *item=None*)
Sets the alignment for widget / layout to alignment and returns true if w is found in this layout (not including child layouts)

Allowed values for alignment: “left”, “right”, “top”, “bottom”

Parameters

- **alignment** (str) – alignment for the layout
- **item** – set alignment for specific child only

Raises ValueError – alignment does not exist

set_size_mode(*mode*)
set the size mode of the layout

Allowed values are “maximum”, “fixed”

Parameters **mode** (str) – size mode for the layout

Raises ValueError – size mode does not exist

class prettyqt.widgets.**FormLayout**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QFormLayout

class prettyqt.widgets.**BoxLayout**(*orientation='horizontal'*, *parent=None*, *margin=None*)
Bases: PyQt5.QtWidgets.QBoxLayout

class prettyqt.widgets.**StackedLayout**
Bases: PyQt5.QtWidgets.QStackedLayout

class prettyqt.widgets.**GridLayout**
Bases: PyQt5.QtWidgets.QGridLayout

class prettyqt.widgets.**ToolBox**
Bases: PyQt5.QtWidgets.QToolBox

class prettyqt.widgets.**Slider**(*orientation='horizontal'*, *parent=None*)
Bases: PyQt5.QtWidgets.QSlider

get_tick_position()
returns tick position

possible values are “none”, “both_sides”, “above”, “below”

Return type str

Returns tick position

set_tick_position(*position*)
sets the tick position for the slider

allowed values are “none”, “both_sides”, “above”, “below”, “left”, “right” for vertical orientation of the slider, “above” equals to “left” and “below” to “right”

Parameters **position** (str) – position for the ticks

class prettyqt.widgets.**StyleOptionSlider**
Bases: PyQt5.QtWidgets.QStyleOptionSlider

is_horizontal()
check if silder is horizontal

Return type bool

Returns True if horizontal, else False

is_vertical()
check if silder is vertical

Return type bool

Returns True if vertical, else False

set_horizontal()
set slider orientation to horizontal

set_vertical()
set slider orientation to vertical

class prettyqt.widgets.**Frame**
Bases: PyQt5.QtWidgets.QFrame

class prettyqt.widgets.**ListWidgetItem**
Bases: PyQt5.QtWidgets.QListWidgetItem

get_checkstate()
returns checkstate

possible values are “unchecked”, “partial”, “checked”

Return type str

Returns checkstate

set_checkstate(*state*)
set checkstate of the checkbox

valid values are: unchecked, partial, checked

Parameters **state** (str) – checkstate to use

Raises ValueError – invalid checkstate

set_icon(*icon*)
set the icon for the action

Parameters **icon** (Union[QIcon, str, None]) – icon to use

class prettyqt.widgets.**TreeWidgetItem**
Bases: PyQt5.QtWidgets.QTreeWidgetItem

get_checkstate()
returns checkstate

possible values are “unchecked”, “partial”, “checked”

Return type str

Returns checkstate

set_checkstate(*state*)
set checkstate of the checkbox

valid values are: unchecked, partial, checked

Parameters **state** (str) – checkstate to use

Raises ValueError – invalid checkstate

set_icon(*icon*)
set the icon for the action

Parameters **icon** (Union[QIcon, str, None]) – icon to use

class prettyqt.widgets.**Action**(*text=”*, *icon=None*, *parent=None*, *shortcut=”*, *tooltip=”*)
Bases: PyQt5.QtWidgets.QAction

get_priority()
returns current priority

Possible values: “low”, “normal”, “high”

Return type str

Returns priority

get_shortcut_context()
returns shortcut context

Possible values: “widget”, “widget_with_children”, “window”, “application”

Return type str

Returns shortcut context

set_icon(*icon*)
set the icon for the action

Parameters **icon** (Union[QIcon, str, None]) – icon to use

set_priority(*priority*)
set priority of the action

Allowed values are “low”, “normal”, “high”

Parameters **mode** – priority for the action

Raises ValueError – priority does not exist

set_shortcut_context(*context*)
set shortcut context

Allowed values are “widget”, “widget_with_children”, “window”, “application”

Parameters **mode** – shortcut context

Raises ValueError – shortcut context does not exist

class prettyqt.widgets.**WidgetAction**(*text=''*, *icon=None*, *parent=None*, *shortcut=''*, *tooltip=''*)

Bases: PyQt5.QtWidgets.QWidgetAction

class prettyqt.widgets.**ToolButton**

Bases: PyQt5.QtWidgets.QToolButton

get_arrow_type()

returns arrow type

possible values are "none", "up", "down", "left", "right"

Return type str

Returns arrow type

get_popup_mode()

returns popup mode

possible values are "delayed", "menu_button", "instant"

Return type str

Returns popup mode

set_arrow_type(*mode*)

sets the arrow type of the toolbutton

valid values are: "none", "up", "down", "left", "right"

Parameters **mode** (str) – arrow type to use

Raises ValueError – invalid arrow type

set_popup_mode(*mode*)

sets the popup mode of the toolbutton

valid values are: "delayed", "menu_button", "instant"

Parameters **mode** (str) – popup mode to use

Raises ValueError – invalid popup mode

class prettyqt.widgets.**ToolTip**

Bases: PyQt5.QtWidgets.QToolTip

class prettyqt.widgets.**Menu**(*title=''*, *icon=None*, *parent=None*)

Bases: PyQt5.QtWidgets.QMenu

add_action(*label*, *callback=None*, *icon=None*, *checkable=False*, *checked=False*, *shortcut=None*, *status_tip=None*)

Add an action to the menu

Parameters

- **label** (Union[str, Action]) – Label for button
- **callback** (Optional[Callable]) – gets called when action is triggered
- **icon** (Optional[Any]) – icon for button (default: {None})
- **checkable** (bool) – as checkbox button (default: {False})
- **checked** (bool) – if checkable, turn on by default (default: {False})
- **shortcut** (Optional[str]) – Shortcut for action (a) (default: {None})
- **status_tip** (Optional[str]) – Status tip to be shown in status bar (default: {None})

Return type Action

Returns Action added to menu

set_icon(*icon*)
set the icon for the menu

Parameters **icon** (Union[QIcon, str, None]) – icon to use

class prettyqt.widgets.**MenuBar**
Bases: PyQt5.QtWidgets.QMenuBar

class prettyqt.widgets.**StatusBar**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QStatusBar

class prettyqt.widgets.**TabWidget**(*parent=None*, *closable=False*, *detachable=False*)
Bases: PyQt5.QtWidgets.QTabWidget

Widget for managing the tabs section

attach_tab(*widget*, *name*, *icon*, *insert_at=None*)
Re-attach the tab by removing the content from the DetachedTab window, closing it, and placing the content back into the DetachableTabWidget

@param widget the content widget from the DetachedTab window @param name the name of the detached tab @param icon the window icon for the detached tab @param insert_at insert the re-attached tab at the given index

close_detached_tabs()
Close all tabs that are currently detached

detach_tab(*index*, *point*)
Detach the tab by removing it's contents and placing them in a DetachedTab window

@param index index location of the tab to be detached @param point screen pos for creating the new DetachedTab window

get_tab_shape()
returns tab shape

possible values are "roundes", "triangular"

Return type str

Returns tab shape

open_widget(*widget*, *title='Unnamed'*)
create a tab containing delivered widget

set_tab_shape(*shape*)
set tab shape for the tabwidget

Valid values are "rounded" and "triangular"

Parameters **shape** (str) – tab shape to use

Raises ValueError – tab shape does not exist

class prettyqt.widgets.**TabBar**(*parent=None*)
Bases: PyQt5.QtWidgets.QTabBar

get_elide_mode()
returns elide mode

possible values are "left", "right", "middle", "none"

Return type str

Returns elide mode

get_remove_behaviour()
returns remove behaviour

possible values are “left_tab”, “right_tab”, “previous_tab”

Return type str

Returns remove behaviour

mouseDoubleClickEvent(*self*, *QMouseEvent*)

set_elide_mode(*mode*)
set elide mode

Valid values are “left”, “right”, “middle”, “none”

Parameters **policy** – elide mode to use

Raises ValueError – invalid elide mode

set_remove_behaviour(*mode*)
sets the remove hehaviour

What tab should be set as current when removeTab is called if the removed tab is also the current tab. Possible values: left, right, previous :type mode: str :param mode: new remove behaviour

class prettyqt.widgets.**ToolBar**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QToolBar

get_style()
returns current style

Possible values: “icon”, “text”, “text_below_icon”, “text_beside_icon”

Return type str

Returns style

is_area_allowed(*area*)
check if toolbar is allowed at specified area

Valid values for area: “left”, “right”, “top”, “bottom”

Parameters **area** (str) – area of the toolbar

Raises ValueError – area does not exist

class prettyqt.widgets.**HeaderView**(*orientation=None*, *parent=None*)
Bases: PyQt5.QtWidgets.QHeaderView

contextMenuEvent(*event*)
context menu for our files tree

class prettyqt.widgets.**DockWidget**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QDockWidget

Customized DockWidget class contains a custom TitleBar with maximise button

class prettyqt.widgets.**Label**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QLabel

get_text_format()
returns current text format

Possible values: “rich”, “plain”, “auto”

Return type str

Returns text format

get_text_interaction()
returns current text interaction mode

Possible values: “none”, “by_mouse”, “by_keyboard”

Return type str

Returns text interaction mode

set_text_format(*text_format*)
set the text format

Allowed values are “rich”, “plain”, “auto”

Parameters **mode** – text format to use

Raises ValueError – text format does not exist

set_text_interaction(*interaction_type*)
set the text interaction mode

Allowed values are “none”, “by_mouse”, “by_keyboard”

Parameters **mode** – text interaction mode to use

Raises ValueError – text interaction mode does not exist

class prettyqt.widgets.**PushButton**(*label=None*, *parent=None*, *callback=None*)
Bases: PyQt5.QtWidgets.QPushButton

class prettyqt.widgets.**CommandLinkButton**(*label=None*, *parent=None*, *callback=None*)
Bases: PyQt5.QtWidgets.QCommandLinkButton

class prettyqt.widgets.**RadioButton**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QRadioButton

class prettyqt.widgets.**ComboBox**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QComboBox

get_insert_policy()
returns insert policy

possible values are “no_insert”, “top”, “current”, “bottom”, “after_current”, “before_current”, “alphabetically”

Return type str

Returns insert policy

get_size_adjust_policy()
returns size adjust policy

possible values are “content”, “first_show”, “min_length”, “min_length_with_icon”

Return type str

Returns size adjust policy

set_insert_policy(*policy*)
set insert policy

valid values are “no_insert”, “top”, “current”, “bottom”, “after_current”, “before_current”, “alphabetically”

Parameters **policy** (str) – insert policy to use

Raises ValueError – invalid insert policy

set_size_adjust_policy(*policy*)
set size adjust policy

possible values are “content”, “first_show”, “min_length”, “min_length_with_icon”

Parameters **policy** (str) – size adjust policy to use

Raises ValueError – invalid size adjust policy

class prettyqt.widgets.**SpinBox**(*parent=None*, *min_value=None*, *max_value=None*, *default_value=None*)
Bases: PyQt5.QtWidgets.QSpinBox

class prettyqt.widgets.**DoubleSpinBox**(*parent=None*, *min_value=None*, *max_value=None*, *default_value=None*)
Bases: PyQt5.QtWidgets.QDoubleSpinBox

class prettyqt.widgets.**CheckBox**(*label=''*, *parent=None*, *checked=False*)
Bases: PyQt5.QtWidgets.QCheckBox

get_checkstate()
returns checkstate

possible values are “unchecked”, “partial”, “checked”

Return type bool

Returns checkstate

set_checkstate(*state*)
set checkstate of the checkbox

valid values are: unchecked, partial, checked

Parameters **state** (str) – checkstate to use

Raises ValueError – invalid checkstate

class prettyqt.widgets.**LineEdit**(*default_value=''*, *read_only=False*, *parent=None*)
Bases: PyQt5.QtWidgets.QLineEdit

font()

Return type Font

get_echo_mode()
returns echo mode

possible values are “normal”, “no_echo”, “password”, “echo_on_edit”

Return type str

Returns echo mode

set_echo_mode(*mode*)
set echo mode

Valid values are “normal”, “no_echo”, “password”, “echo_on_edit”

Parameters **policy** – echo mode to use

Raises ValueError – invalid echo mode

set_read_only(*value=True*)
set test to read only

Parameters **value** (bool) – True, for read-only, otherwise False

class prettyqt.widgets.**KeySequenceEdit**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QKeySequenceEdit

class prettyqt.widgets.**TextEdit**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QTextEdit

class prettyqt.widgets.**DateEdit**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QDateEdit

class prettyqt.widgets.**TimeEdit**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QTimeEdit

class prettyqt.widgets.**DateTimeEdit**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QDateTimeEdit

class prettyqt.widgets.**CalendarWidget**
Bases: PyQt5.QtWidgets.QCalendarWidget

class prettyqt.widgets.**PlainTextEdit**(*text=''*, *parent=None*, *read_only=False*)
Bases: PyQt5.QtWidgets.QPlainTextEdit

set_read_only(*value=True*)
set test to read only

Parameters **value** (bool) – True, for read-only, otherwise False

class prettyqt.widgets.**TextBrowser**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QTextBrowser

class prettyqt.widgets.**ProgressBar**(*text_visible=True*, *parent=None*)
Bases: PyQt5.QtWidgets.QProgressBar

Progress dialog

wrapper for QtWidgets.QProgressBar

get_alignment()
returns current alignment

Possible values: "left", "right", "top", "bottom"

Return type str

Returns alignment

get_text_direction()
returns current text direction

Possible values are "top_to_bottom", "bottom_to_top"

Return type str

Returns Text direction

set_alignment(*alignment*)
set the alignment of the layout

Allowed values are "left", "right", "top", "bottom"

Parameters **mode** – alignment for the layout

Raises ValueError – alignment does not exist

set_text_direction(*text_direction*)
set the text direction of the layout

Allowed values are "top_to_bottom", "bottom_to_top"

Parameters **mode** – text direction for the layout

Raises ValueError – text direction does not exist

class prettyqt.widgets.**ColumnView**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QColumnView

class prettyqt.widgets.**ListView**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QListView

get_view_mode()
returns view mode

possible values are "list", "icon"

Return type str

Returns view mode

set_view_mode(*mode*)
set view mode

possible values are "list", "icon"

Parameters **mode** (str) – view mode to use

Raises ValueError – invalid view mode

class prettyqt.widgets.**ListWidget**(*parent=None*, *selection_mode='single'*)
Bases: PyQt5.QtWidgets.QListWidget

class prettyqt.widgets.**TreeView**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QTreeView

class prettyqt.widgets.**TreeWidget**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QTreeWidget

class prettyqt.widgets.**TableView**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QTableView

class prettyqt.widgets.**TableWidget**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QTableWidget

class prettyqt.widgets.**SplashScreen**(*path*, *width=None*)
Bases: PyQt5.QtWidgets.QSplashScreen

class prettyqt.widgets.**ProgressDialog**(*parent=None*)
Bases: PyQt5.QtWidgets.QProgressDialog

Progress dialog

wrapper for QtWidgets.QProgressDialog

class prettyqt.widgets.**FontDialog**
Bases: PyQt5.QtWidgets.QFontDialog

class prettyqt.widgets.**FileDialog**(*path=None*, *mode='open'*, *caption=''*, *path_id=None*, *extension_filter=None*, *parent=None*)

Bases: PyQt5.QtWidgets.QFileDialog

simple dialog used to display some widget

directory()
return current directory

returns current directory level as a Pathlib object

Return type Path

Returns Pathlib object

get_accept_mode()
returns accept mode

possible values are "save", "open"

Return type str

Returns accept mode

get_file_mode()
returns file mode

possible values are "existing_file", "existing_files", "any_file", "directory"

Return type str

Returns file mode

get_label_text(*label*)
returns label text

possible values are "look_in", "filename", "filetype", "accept", "reject"

Return type str

Returns label text

get_view_mode()
returns view mode

possible values are "detail", "list"

Return type str

Returns view mode

set_accept_mode(*mode*)
set accept mode

possible values are "save", "open"

Parameters **mode** (str) – accept mode to use

Raises ValueError – invalid accept mode

set_file_mode(*mode*)
sets the file mode of the dialog

allowed values are "existing_file", "existing_files", "any_file" "directory"

Parameters **mode** (str) – mode to use

set_filter(*extension_dict*)
set filter based on given dictionary

dict must contain "'name': ['.ext1', '.ext2']" as key-value pairs

Parameters **extension_dict** (dict) – filter dictionary

set_label_text(*label*, *text*)
sets the label text for button label

possible values for label are "look_in", "filename", "filetype", "accept", "reject"

Parameters

- **label** (str) – button to set text for
- **text** (str) – text to use

set_view_mode(*mode*)
set view mode

possible values are "detail", "list"

Parameters **mode** (str) – view mode to use

Raises ValueError – invalid view mode

class prettyqt.widgets.**ColorDialog**
Bases: PyQt5.QtWidgets.QColorDialog

class prettyqt.widgets.**InputDialog**
Bases: PyQt5.QtWidgets.QInputDialog

class prettyqt.widgets.**DialogButtonBox**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QDialogButtonBox

add_button(*button*, *role='accept'*, *callback=None*)
add a button

Parameters

- **button** – button to add
- **role** – role of the button
- **callback** – function to call when button gets clicked

Returns created button

Raises ValueError – Button type not available

add_default_button(*button*, *callback=None*)
add a default button

Valid arguments: "cancel", "ok", "save", "open", "close", "discard", "apply", "reset", "restore_defaults", "help", "save_all", "yes", "yes_to_all", "no", "no_to_all", "abort", "retry", "ignore"

Parameters

- **button** (str) – button to add
- **callback** – function to call when button gets clicked

Returns created button

Raises ValueError – Button type not available

classmethod create(*self*, *window: sip.voidptr = 0*, *initializeWindow: bool = True*, *destroyOldWindow: bool = True*)

get_orientation()
returns current orientation

Possible values: "horizontal", "vertical"

Return type str

Returns orientation

set_orientation(*orientation*)
set the orientation of the button box

Allowed values are "horizontal", "vertical"

Parameters **mode** – orientation for the button box

Raises ValueError – orientation does not exist

class prettyqt.widgets.**ButtonGroup**
Bases: PyQt5.QtWidgets.QButtonGroup

class prettyqt.widgets.**GroupBox**(*title=''*, *checkable=False*, *parent=None*)
Bases: PyQt5.QtWidgets.QGroupBox

GroupBox widget

A group box provides a frame, a title on top, a keyboard shortcut, and displays various other widgets inside itself. The keyboard shortcut moves keyboard focus to one of the group box's child widgets.

class prettyqt.widgets.**Splitter**(*orientation='horizontal'*, *parent=None*)
Bases: PyQt5.QtWidgets.QSplitter

get_orientation()
returns current orientation

Possible values: "horizontal", "vertical"

Return type str

Returns orientation

set_orientation(*orientation*)
set the orientation of the splitter

Allowed values are "horizontal", "vertical"

Parameters **mode** – orientation for the splitter

Raises ValueError – orientation does not exist

class prettyqt.widgets.**Wizard**
Bases: PyQt5.QtWidgets.QWizard

class prettyqt.widgets.**WizardPage**
Bases: PyQt5.QtWidgets.QWizardPage

class prettyqt.widgets.**MainWindow**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QMainWindow

Class for our mainWindow includes all docks, a centralwidget and a toolbar

add_toolbar(*toolbar*, *position='top'*)
adds a toolbar to the mainmenu at specified area

Valid values for position: "left", "right", "top", "bottom"

Parameters

- **toolbar** – toolbar to use
- **position** (str) – position of the toolbar

Raises ValueError – position does not exist

add_toolbar_break(*position='top'*)
Adds a toolbar break to the given area after all the other objects that are present.

Valid values for position: "left", "right", "top", "bottom"

Parameters **position** (str) – position of the toolbar

Raises ValueError – position does not exist

closeEvent(*event*)
override, gets executed when app gets closed. saves GUI settings

createPopupMenu(*self*) → QMenu

set_icon(*icon*)
set the icon for the menu

Parameters **icon** (Union[QIcon, str, None]) – icon to use

toggle_fullscreen()
toggle between fullscreen and regular size

class prettyqt.widgets.**AbstractItemDelegate**
Bases: PyQt5.QtWidgets.QAbstractItemDelegate

class prettyqt.widgets.**ItemDelegate**
Bases: PyQt5.QtWidgets.QItemDelegate

class prettyqt.widgets.**StyledItemDelegate**
Bases: PyQt5.QtWidgets.QStyledItemDelegate

Python Module Index

p

Index

T

V

W