pyqtribbon

Release 0.5.1

WANG Hailin

Mar 13, 2023

CONTENTS:

Ribbon Bar for PyQt or PySide applications.

- GitHub Repository: github.com/haiiliin/pyqtribbon.
- PyPI: pypi.org/project/pyqtribbon.
- Documentation: pyqtribbon.haiiliin.com/en/stable.
- Read the Docs: readthedocs.org/projects/pyqtribbon.

CHAPTER

ONE

GETTING STARTED

1.1 Installation

PyQtRibbon is distribued to PyPI, you can use pip to install it:

```
pip install pyqtribbon
```

You can also install the package from source:

```
pip install git+https://github.com/haiiliin/pyqtribbon.git@main
```

CHAPTER

TWO

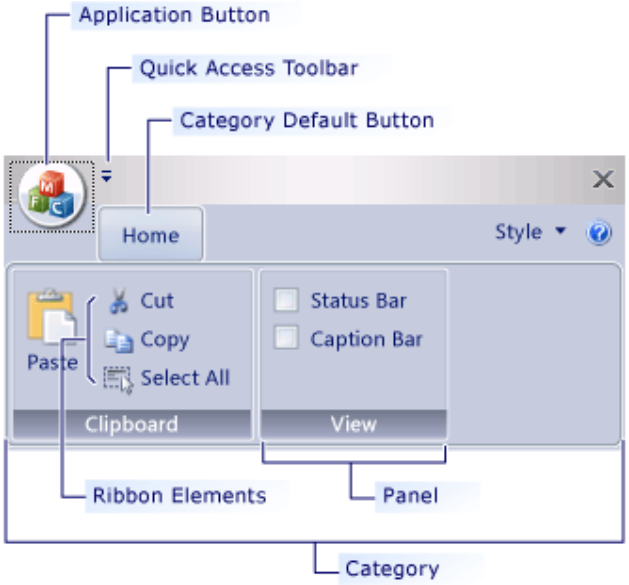
THE RIBBON BAR

2.1 Introduction

The ribbon is first introduced by Microsoft in the 2000's. It is a toolbar with a tabbed interface. According to Microsoft:

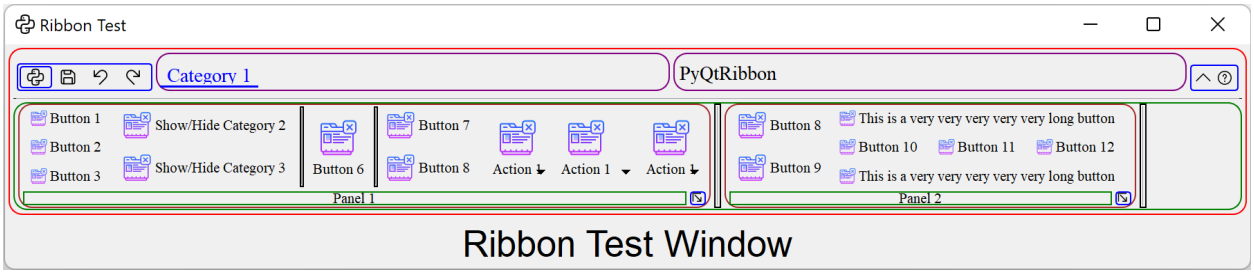
Note: A ribbon is a user interface (UI) element that organizes commands into logical groups. These groups appear on separate tabs in a strip across the top of the window. The ribbon replaces the menu bar and toolbars. A ribbon can significantly improve application usability. For more information, see Ribbons. The following illustration shows a ribbon. A ribbon can significantly improve application usability. For more information, see Ribbons. The following illustration shows a ribbon.

2.2 Definitions of Ribbon Elements



- **Application button**: The button that appears on the upper-left corner of a ribbon. The Application button replaces the File menu and is visible even when the ribbon is minimized. When the button is clicked, a menu that has a list of commands is displayed.
- **Quick Access toolbar**: A small, customizable toolbar that displays frequently used commands.
- **Category**: The logical grouping that represents the contents of a ribbon tab.
- **Category Default button**: The button that appears on the ribbon when the ribbon is minimized. When the button is clicked, the category reappears as a menu.
- **Panel**: An area of the ribbon bar that displays a group of related controls. Every ribbon category contains one or more ribbon panels.
- **Ribbon elements**: Controls in the panels, for example, buttons and combo boxes. To see the various controls that can be hosted on a ribbon, see RibbonGadgets Sample: Ribbon Gadgets Application.

2.3 Ribbon Elements in PyQtRibbon



CHAPTER

THREE

USER MANUAL

3.1 The RibbonScreenShotWindow Class

The `RibbonScreenShotWindow` class is just for taking a screenshot of the window, the window will be closed 0.1s after it is shown. It is just used for documenting the window.

class pyqtribbon.screenshotwindow.**RibbonScreenShotWindow**(*fileName: str = 'shot.jpg'*, **args*, ***kwargs*)

This class is just for taking a screenshot of the window, the window will be closed 0.1s after it is shown.

Initialize the class.

Parameters
fileName – The file name for the screenshot.

setScreenShotFileName(*fileName: str*)

Set the file name for the screenshot.

Parameters
fileName – The file name for the screenshot.

takeScreenShot()

Take a screenshot of the window.

3.2 Instantiate a Ribbon Bar

`RibbonBar` is inherited from `QMenuBar`, you can use the *setMenuBar* method of `QMainWindow` to set the ribbon bar as the main menu bar.

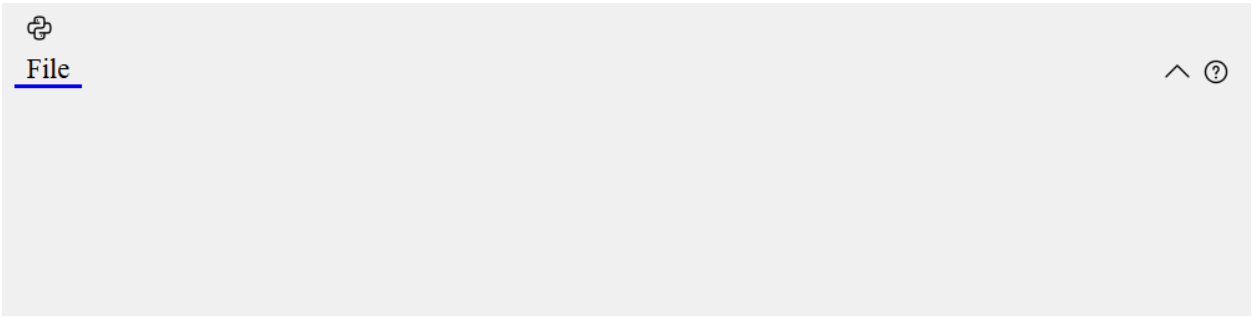
```
from pyqtribbon import RibbonBar

window = QtWidgets.QMainWindow()
ribbon = RibbonBar()
window.setMenuBar(ribbon)
```

3.2.1 Example

For example, using the following code,

```
import sys

from qtpy import QtWidgets, QtGui

from pyqtribbon import RibbonBar
from pyqtribbon.screenshotwindow import RibbonScreenShotWindow

if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    app.setFont(QtGui.QFont("Times New Roman", 8))
    window = RibbonScreenShotWindow('ribbonbar.png')

    # Ribbon bar
    ribbonbar = RibbonBar()
    window.setMenuBar(ribbonbar)

    # Show the window
    window.resize(1000, 250)
    window.show()

    sys.exit(app.exec_())
```

You can get a window like this:



3.3 Customize Ribbon Bar

3.3.1 General Setups

RibbonBar.setRibbonStyle(style)	Set the style of the ribbon.
RibbonBar.ribbonHeight()	Get the total height of the ribbon.
RibbonBar.setRibbonHeight(height)	Set the total height of the ribbon.
RibbonBar.showRibbon()	Show the ribbon.
RibbonBar.hideRibbon()	Hide the ribbon.
RibbonBar.ribbonVisible()	Get the visibility of the ribbon.
RibbonBar.setRibbonVisible(visible)	Set the visibility of the ribbon.

3.3.2 Setup Application Button

RibbonBar.applicationOptionButton()	Return the application button.
RibbonBar.setApplicationIcon(icon)	Set the application icon.
RibbonBar.addFileMenu()	Add a file menu to the ribbon.

3.3.3 Setup Title

RibbonBar.title()	Return the title of the ribbon.
RibbonBar.setTitle(title)	Set the title of the ribbon.
RibbonBar.addTitleWidget(widget)	Add a widget to the title widget.
RibbonBar.insertTitleWidget(index, widget)	Insert a widget to the title widget.
RibbonBar.removeTitleWidget(widget)	Remove a widget from the title widget.

3.3.4 Setup Category Tab Bar

RibbonBar.tabBar()	Return the tab bar of the ribbon.

3.3.5 Setup Quick Access Bar

RibbonBar.quickAccessToolBar()	Return the quick access toolbar of the ribbon.
RibbonBar.addQuickAccessButton(button)	Add a button to the quick access bar.
RibbonBar.setQuickAccessButtonHeight([height])	Set the height of the quick access buttons.

3.3.6 Setup Right Tool Bar

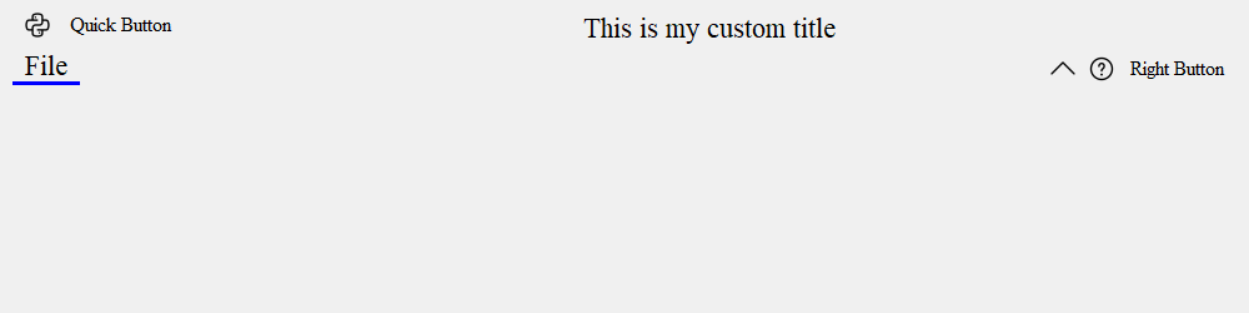
RibbonBar.rightToolBar()	Return the right toolbar of the ribbon.
RibbonBar.addRightToolButton(button)	Add a widget to the right button bar.
RibbonBar.setRightToolBarHeight([height])	Set the height of the right buttons.
RibbonBar.setHelpButtonIcon(icon)	Set the icon of the help button.
RibbonBar.removeHelpButton()	Remove the help button from the ribbon.
RibbonBar.helpButtonClicked(bool)	Signal, the help button was clicked.
RibbonBar.collapseRibbonButton()	Return the collapse ribbon button.
RibbonBar.setCollapseButtonIcon(icon)	Set the icon of the min button.
RibbonBar.removeCollapseButton()	Remove the min button from the ribbon.

3.3.7 Example

For example, using the following code,

```
import sys

from qtpy import QtGui
from qtpy.QtWidgets import QApplication, QToolButton

from pyqtribbon import RibbonBar
from pyqtribbon.screenshotwindow import RibbonScreenShotWindow

if __name__ == '__main__':
    app = QApplication(sys.argv)
    app.setFont(QtGui.QFont("Times New Roman", 8))
    window = RibbonScreenShotWindow('ribbonbar-customize.png')

    # Ribbon bar
    ribbonbar = RibbonBar()
    window.setMenuBar(ribbonbar)

    # Title of the ribbon
    ribbonbar.setTitle('This is my custom title')

    # Quick Access Bar
    qbutton = QToolButton()
    qbutton.setText('Quick Button')
    ribbonbar.addQuickAccessButton(qbutton)

    # Right toolbar
    rbutton = QToolButton()
    rbutton.setText('Right Button')
    ribbonbar.addRightToolButton(rbutton)

    # Show the window
    window.resize(1000, 250)
    window.show()

    sys.exit(app.exec_())
```

You can get a window like this:



3.3.8 Manage Categories

RibbonBar.categories()	Return a list of categories of the ribbon.
RibbonBar.addCategory(title[, style, color])	Add a new category to the ribbon.
RibbonBar.addCategoriesBy(data)	Add categories from a dict.
RibbonBar.addNormalCategory(title)	Add a new category to the ribbon.
RibbonBar.addContextCategory(title[, color])	Add a new context category to the ribbon.
RibbonBar.addContextCategories(name, titles)	Add a group of context categories with the same tab color to the ribbon.
RibbonBar.showContextCategory(category)	Show the given category or categories, if it is not a context category, nothing happens.
RibbonBar.hideContextCategory(category)	Hide the given category or categories, if it is not a context category, nothing happens.
RibbonBar.removeCategory(category)	Remove a category from the ribbon.
RibbonBar.setCurrentCategory(category)	Set the current category.
RibbonBar.currentCategory()	Return the current category.
RibbonBar.showCategoryByIndex(index)	Show category by tab index

3.4 Customize Categories

3.4.1 Setup Styles

RibbonCategory.categoryStyle()	Return the button style of the category.
RibbonCategory.setCategoryStyle(style)	Set the button style of the category.

3.4.2 Manage Panels

RibbonCategory.addPanel(title[, ...])	Add a new panel to the category.
RibbonCategory.addPanelsBy(data)	Add panels from a dictionary.
RibbonCategory.removePanel(title)	Remove a panel from the category.
RibbonCategory.takePanel(title)	Remove and return a panel from the category.
RibbonCategory.panel(title)	Return a panel from the category.
RibbonCategory.panels()	Return all panels in the category.

3.4.3 Example

For example, using the following code,

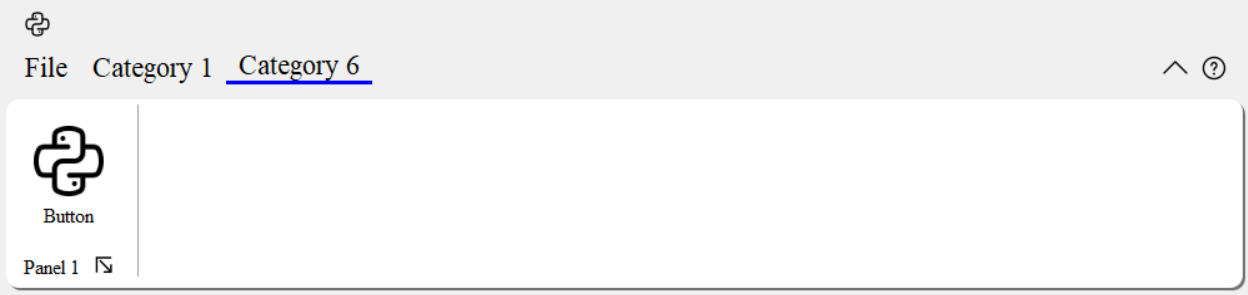
```
import sys

from qtpy import QtGui
from qtpy.QtWidgets import QApplication
from qtpy.QtGui import QIcon

from pyqtribbon import RibbonBar, RibbonCategoryStyle
```

(continues on next page)

(continued from previous page)

```
from pyqtribbon.screenshotwindow import RibbonScreenShotWindow

if __name__ == '__main__':
    app = QApplication(sys.argv)
    app.setFont(QtGui.QFont("Times New Roman", 8))
    window = RibbonScreenShotWindow('category.png')

    # Ribbon bar
    ribbonbar = RibbonBar()
    window.setMenuBar(ribbonbar)

    # Categories
    category1 = ribbonbar.addCategory('Category 1')
    panel1 = category1.addPanel('Panel 1')
    panel1.addLargeButton('Large Button 1', QIcon('python.png'))

    category2 = ribbonbar.addContextCategory('Category 2')
    panel12 = category2.addPanel('Panel 2')
    panel12.addLargeButton('Large Button 2', QIcon('python.png'))

    categories = ribbonbar.addCategoriesBy({
        'Category 6': {
            "style": RibbonCategoryStyle.Normal,
            "panels": {
                "Panel 1": {
                    "showPanelOptionButton": True,
                    "widgets": {
                        "Button 1": {
                            "type": "Button",
                            "arguments": {
                                "icon": QIcon("python.png"),
                                "text": "Button",
                                "tooltip": "This is a tooltip",
                            }
                        },
                    }
                },
            }
        }
    })
    ribbonbar.setCurrentCategory(categories['Category 6'])

    # Show the window
    window.resize(1000, 250)
    window.show()

    sys.exit(app.exec_())
```

You can get a window like this:



3.5 Customize Panels

3.5.1 Setup Title Label

RibbonPanel.title()	Get the title of the panel.
RibbonPanel.setTitle(title)	Set the title of the panel.

3.5.2 Setup Panel Option Button

RibbonPanel.panelOptionButton()	Return the panel option button.
RibbonPanel.setPanelOptionToolTip(text)	Set the tooltip of the panel option button.
RibbonPanel.panelOptionClicked(bool)	int = ..., arguments: Sequence = ...) -> PYQT_SIGNAL

3.5.3 Add Widgets to Panels

RibbonPanel.addWidget(widget[, rowSpan, ...])	Add a widget to the panel.
RibbonPanel.addWidgetsBy(data)	Add widgets to the panel.
RibbonPanel.removeWidget(widget)	Remove a widget from the panel.
RibbonPanel.widget(index)	Get the widget at the given index.
RibbonPanel.widgets()	Get all the widgets in the panel.
RibbonPanel.addSmallWidget(widget[, mode, ...])	Add a small widget to the panel.
RibbonPanel.addMediumWidget(widget[, mode, ...])	Add a medium widget to the panel.
RibbonPanel.addLargeWidget(widget[, mode, ...])	Add a large widget to the panel.
RibbonPanel.addButton([text, icon, style, ...])	Add a button to the panel.
RibbonPanel.addSmallButton([text, icon, ...])	Add a small button to the panel.
RibbonPanel.addMediumButton([text, icon, ...])	Add a medium button to the panel.
RibbonPanel.addLargeButton([text, icon, ...])	Add a large button to the panel.
RibbonPanel.addToggleButton([text, icon, ...])	Add a toggle button to the panel.
RibbonPanel.addSmallToggleButton([text, ...])	Add a small toggle button to the panel.
RibbonPanel.addMediumToggleButton([text, ...])	Add a medium toggle button to the panel.
RibbonPanel.addLargeToggleButton([text, ...])	Add a large toggle button to the panel.
RibbonPanel.addComboBox(items[, rowSpan, ...])	Add a combo box to the panel.
RibbonPanel.addFontComboBox([rowSpan, ...])	Add a font combo box to the panel.
RibbonPanel.addLineEdit([rowSpan, colSpan, ...])	Add a line edit to the panel.

continues on next page

Table 1 – continued from previous page

RibbonPanel.addTextEdit([rowSpan, colSpan, ...])	Add a text edit to the panel.
RibbonPanel.addPlainTextEdit([rowSpan, ...])	Add a plain text edit to the panel.
RibbonPanel.addLabel(text[, rowSpan, ...])	Add a label to the panel.
RibbonPanel.addProgressBar([rowSpan, ...])	Add a progress bar to the panel.
RibbonPanel.addSlider([rowSpan, colSpan, ...])	Add a slider to the panel.
RibbonPanel.addSpinBox([rowSpan, colSpan, ...])	Add a spin box to the panel.
RibbonPanel.addDoubleSpinBox([rowSpan, ...])	Add a double spin box to the panel.
RibbonPanel.addDateEdit([rowSpan, colSpan, ...])	Add a date edit to the panel.
RibbonPanel.addTimeEdit([rowSpan, colSpan, ...])	Add a time edit to the panel.
RibbonPanel.addDateTimeEdit([rowSpan, ...])	Add a date time edit to the panel.
RibbonPanel.addTableWidget([rowSpan, ...])	Add a table widget to the panel.
RibbonPanel.addTreeWidget([rowSpan, ...])	Add a tree widget to the panel.
RibbonPanel.addListWidget([rowSpan, ...])	Add a list widget to the panel.
RibbonPanel.addCalendarWidget([rowSpan, ...])	Add a calendar widget to the panel.
RibbonPanel.addSeparator([orientation, ...])	Add a separator to the panel.
RibbonPanel.addHorizontalSeparator([width, ...])	Add a horizontal separator to the panel.
RibbonPanel.addVerticalSeparator([width, ...])	Add a vertical separator to the panel.
RibbonPanel.addGallery([minimumWidth, ...])	Add a gallery to the panel.

3.5.4 Example

For example, using the following code,

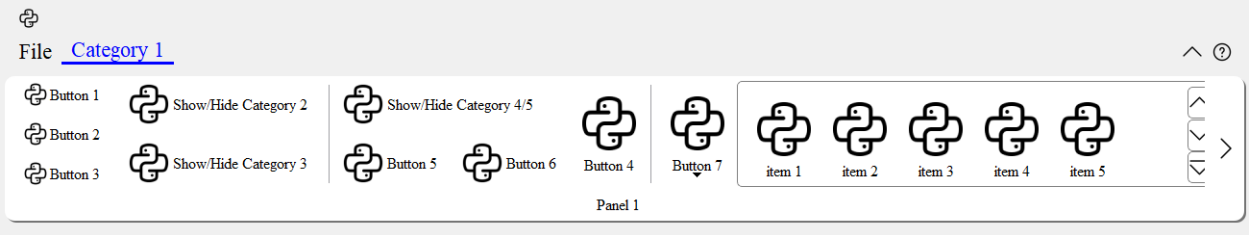
```
import sys

from qtpy import QtGui
from qtpy.QtWidgets import QApplication, QToolButton, QMenu, QLabel, QLineEdit
from qtpy.QtGui import QIcon
from qtpy.QtCore import Qt

from pyqtribbon import RibbonBar
from pyqtribbon.screenshotwindow import RibbonScreenShotWindow

if __name__ == '__main__':
    app = QApplication(sys.argv)
    app.setFont(QtGui.QFont("Times New Roman", 8))
    window = RibbonScreenShotWindow('panel.png')

    # Ribbon bar
    ribbonbar = RibbonBar()
    window.setMenuBar(ribbonbar)

    category1 = ribbonbar.addCategory("Category 1")
    panel = category1.addPanel("Panel 1", showPanelOptionButton=False)
    panel.addSmallButton("Button 1", icon=QIcon("python.png"))
    panel.addSmallButton("Button 2", icon=QIcon("python.png"))
    panel.addSmallButton("Button 3", icon=QIcon("python.png"))
    panel.addMediumToggleButton("Show/Hide Category 2", icon=QIcon("python.png"))
    panel.addVerticalSeparator()
    panel.addMediumToggleButton("Show/Hide Category 3", icon=QIcon("python.png"))
```

(continues on next page)

(continued from previous page)

```
    panel.addMediumToggleButton("Show/Hide Category 4/5", icon=QIcon("python.png"),
                                                          colSpan=2, alignment=Qt.AlignLeft)
    panel.addLargeButton("Button 4", icon=QIcon("python.png"))
    panel.addVerticalSeparator()
    panel.addMediumButton("Button 5", icon=QIcon("python.png"))
    panel.addMediumButton("Button 6", icon=QIcon("python.png"))

    button = panel.addLargeButton("Button 7", icon=QIcon("python.png"))
    menu = QMenu()
    menu.addAction(QIcon("python.png"), "Action 1")
    menu.addAction(QIcon("python.png"), "Action 2")
    menu.addAction(QIcon("python.png"), "Action 3")
    button.setMenu(menu)
    button.setPopupMode(QToolButton.InstantPopup)
    panel.addWidget(button, rowSpan=6)

    gallery = panel.addGallery(minimumWidth=500, popupHideOnClick=True)
    for i in range(100):
        gallery.addToggleButton(f'item {i+1}', QIcon("python.png"))
    popupMenu = gallery.popupMenu()
    submenu = popupMenu.addMenu(QIcon("python.png"), 'Submenu')
    submenu.addAction(QIcon("python.png"), "Action 4")
    popupMenu.addAction(QIcon("python.png"), "Action 1")
    popupMenu.addAction(QIcon("python.png"), "Action 2")
    popupMenu.addSeparator()
    popupMenu.addWidget(QLabel("This is a custom widget"))
    formLayout = popupMenu.addFormLayoutWidget()
    formLayout.addRow(QLabel("Row 1"), QLineEdit())

    # Show the window
    window.resize(1300, 250)
    window.show()

    sys.exit(app.exec_())
```

You can get a window like this:

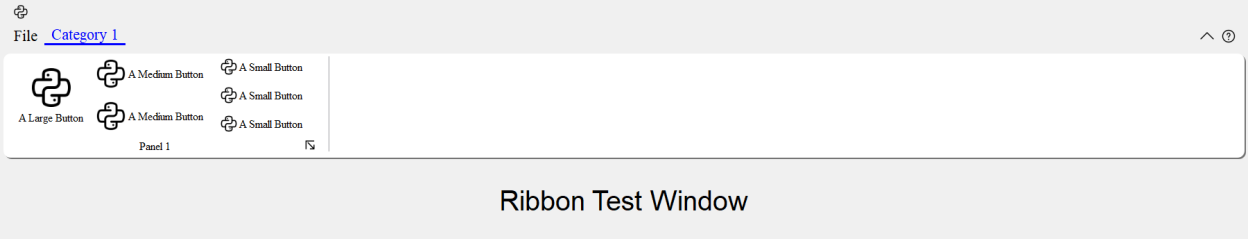


3.6 A Complete Example

The following code snippet is a complete example.

```
import sys

from PyQt5.QtWidgets import QApplication, QLabel, QWidget, QVBoxLayout
from PyQt5.QtGui import QIcon, QFont
from PyQt5.QtCore import Qt

from pyqtribbon import RibbonBar
from pyqtribbon.screenshotwindow import RibbonScreenShotWindow
from pyqtribbon.utils import data_file_path

if __name__ == "__main__":
    app = QApplication(sys.argv)
    app.setFont(QFont("Times New Roman", 8))

    # Central widget
    window = RibbonScreenShotWindow('tutorial-ribbonbar.png')
    window.setWindowIcon(QIcon(data_file_path("icons/python.png")))
    centralWidget = QWidget()
    window.setCentralWidget(centralWidget)
    layout = QVBoxLayout(centralWidget)

    # Ribbon bar
    ribbonbar = RibbonBar()
    window.setMenuBar(ribbonbar)
    category = ribbonbar.addCategory("Category 1")
    panel = category.addPanel("Panel 1")
    panel.addLargeButton("A Large Button", QIcon(data_file_path("icons/python.png")))
    panel.addMediumButton("A Medium Button", QIcon(data_file_path("icons/python.png")))
    panel.addMediumButton("A Medium Button", QIcon(data_file_path("icons/python.png")))
    panel.addSmallButton("A Small Button", QIcon(data_file_path("icons/python.png")))
    panel.addSmallButton("A Small Button", QIcon(data_file_path("icons/python.png")))
    panel.addSmallButton("A Small Button", QIcon(data_file_path("icons/python.png")))

    # Display a label in the main window
    label = QLabel("Ribbon Test Window")
    label.setFont(QFont("Arial", 20))
    label.setAlignment(Qt.AlignCenter)

    # Add the ribbon bar and label to the layout
    layout.addWidget(label, 1)

    # Show the window
    window.resize(1800, 350)
    window.show()
    sys.exit(app.exec_())
```

You can get a window like this:


File
Category 1
A Large Button
A Medium Button
A Medium Button
A Small Button
A Small Button
A Small Button
Panel 1
Ribbon Test Window

CHAPTER

FOUR

API REFERENCE

This page contains auto-generated API reference documentation[1].

4.1 pyqtribbon

4.1.1 Submodules

pyqtribbon.category

Module Contents

Classes

RibbonCategoryStyle	The button style of a category.
RibbonCategoryLayoutButton	Previous/Next buttons in the category when the
RibbonCategoryScrollArea	Scroll area for the gallery
RibbonCategoryScrollAreaContents	Scroll area contents for the gallery
RibbonCategoryLayoutWidget	The category layout widget's category scroll area to arrange the widgets in the category.
RibbonCategory	The RibbonCategory is the logical grouping that represents the contents of a ribbon tab.
RibbonNormalCategory	A normal category.
RibbonContextCategory	A context category.
RibbonContextCategories	A list of context categories.

Attributes

Normal
Context
contextColors

[1] Created with sphinx-autoapi

class pyqtribbon.category.**RibbonCategoryStyle**

Bases: enum.IntEnum

The button style of a category.

Normal = 0

Context = 1

pyqtribbon.category.**Normal**

pyqtribbon.category.**Context**

pyqtribbon.category.**contextColors**

class pyqtribbon.category.**RibbonCategoryLayoutButton**

Bases: qtpy.QtWidgets.QToolButton

Previous/Next buttons in the category when the size is not enough for the widgets.

class pyqtribbon.category.**RibbonCategoryScrollArea**

Bases: qtpy.QtWidgets.QScrollArea

Scroll area for the gallery

class pyqtribbon.category.**RibbonCategoryScrollAreaContents**

Bases: qtpy.QtWidgets.QFrame

Scroll area contents for the gallery

class pyqtribbon.category.**RibbonCategoryLayoutWidget**(*parent=None*)

Bases: qtpy.QtWidgets.QFrame

The category layout widget's category scroll area to arrange the widgets in the category.

displayOptionsButtonClicked

paintEvent(*a0: qtpy.QtGui.QPaintEvent*) → None

Override the paint event to draw the background.

resizeEvent(*a0: qtpy.QtGui.QResizeEvent*) → None

Override the resize event to resize the scroll area.

autoSetScrollButtonsVisible()

Set the visibility of the scroll buttons.

scrollPrevious()

Scroll the category to the previous widget.

scrollNext()

Scroll the category to the next widget.

addWidget(*widget: qtpy.QtWidgets.QWidget*)

Add a widget to the category layout.

Parameters

widget – The widget to add.

removeWidget(*widget: qtpy.QtWidgets.QWidget*)

Remove a widget from the category layout.

Parameters
widget – The widget to remove.

takeWidget(*widget: qtpy.QtWidgets.QWidget*) → qtpy.QtWidgets.QWidget

Remove and return a widget from the category layout.

Parameters
widget – The widget to remove.

Returns
The widget that was removed.

class pyqtribbon.category.**RibbonCategory**(*title: str = ''*, *style: RibbonCategoryStyle = RibbonCategoryStyle.Normal*, *color: qtpy.QtGui.QColor = None*, *parent=None) RibbonCategory(parent=None*)

Bases: qtpy.QtWidgets.QFrame

The RibbonCategory is the logical grouping that represents the contents of a ribbon tab.

_title: *str*

_ribbon: *pyqtribbon.typehints.RibbonType* | None

_style: *RibbonCategoryStyle*

_panels: Dict[str, *pyqtribbon.panel.RibbonPanel*]

_color: qtpy.QtGui.QColor | None

_maxRows: int = 6

setMaximumRows(*rows: int*)

Set the maximum number of rows.

Parameters
rows – The maximum number of rows.

title() → str

Return the title of the category.

setCategoryStyle(*style:* RibbonCategoryStyle)

Set the button style of the category.

Parameters
style – The button style.

categoryStyle() → *RibbonCategoryStyle*

Return the button style of the category.

Returns
The button style.

addPanelsBy(*data: Dict[str, Dict]*) → Dict[str, *pyqtribbon.panel.RibbonPanel*]

Add panels from a dictionary.

Parameters
data – The dictionary. The keys are the titles of the panels. The value is a dictionary of arguments. the argument showPanelOptionButton is a boolean to decide whether to show

the panel option button, the rest arguments are passed to the RibbonPanel.addWidgetsBy() method. The dict is of the form:

```
{
    "panel-title": {
        "showPanelOptionButton": True,
        "widgets": {
            "widget-name": {
                "type": "Button",
                "arguments": {
                    "key1": "value1",
                    "key2": "value2"
                }
            },
        }
    },
}
```

Returns
A dictionary of the newly created panels.

addPanel(*title: str*, *showPanelOptionButton=True*) → *pyqtribbon.panel.RibbonPanel*

Add a new panel to the category.

Parameters

- **title** – The title of the panel.
- **showPanelOptionButton** – Whether to show the panel option button.

Returns
The newly created panel.

removePanel(*title: str*)

Remove a panel from the category.

Parameters
title – The title of the panel.

takePanel(*title: str*) → *pyqtribbon.panel.RibbonPanel*

Remove and return a panel from the category.

Parameters
title – The title of the panel.

Returns
The removed panel.

panel(*title: str*) → *pyqtribbon.panel.RibbonPanel*

Return a panel from the category.

Parameters
title – The title of the panel.

Returns
The panel.

panels() → Dict[str, *pyqtribbon.panel.RibbonPanel*]

Return all panels in the category.

Returns
The panels.

class pyqtribbon.category.**RibbonNormalCategory**(*title: str*, *parent: qtpy.QtWidgets.QWidget*)

Bases: *RibbonCategory*

A normal category.

setCategoryStyle(*style:* RibbonCategoryStyle)

Set the button style of the category.

Parameters
style – The button style.

class pyqtribbon.category.**RibbonContextCategory**(*title: str*, *color: qtpy.QtGui.QColor*, *parent: qtpy.QtWidgets.QWidget*)

Bases: *RibbonCategory*

A context category.

setCategoryStyle(*style:* RibbonCategoryStyle)

Set the button style of the category.

Parameters
style – The button style.

color() → qtpy.QtGui.QColor

Return the color of the context category.

Returns
The color of the context category.

setColor(*color: qtpy.QtGui.QColor*)

Set the color of the context category.

Parameters
color – The color of the context category.

showContextCategory()

Show the given category, if it is not a context category, nothing happens.

hideContextCategory()

Hide the given category, if it is not a context category, nothing happens.

categoryVisible() → bool

Return whether the category is shown.

Returns
Whether the category is shown.

setCategoryVisible(*visible: bool*)

Set the state of the category.

Parameters
visible – The state.

class pyqtribbon.category.**RibbonContextCategories**(*name: str*, *color: qtpy.QtGui.QColor*, *categories: Dict[str,* RibbonContextCategory*]*, *ribbon*)

Bases: Dict[str, *RibbonContextCategory*]

A list of context categories.

_ribbon: *pyqtribbon.typehints.RibbonType*

name() → str

Return the name of the context categories.

setName(*name: str*)

Set the name of the context categories.

color() → qtpy.QtGui.QColor

Return the color of the context categories.

setColor(*color: qtpy.QtGui.QColor*)

Set the color of the context categories.

showContextCategories()

Show the categories

hideContextCategories()

Hide the categories

categoriesVisible() → bool

Return whether the categories are shown.

setCategoriesVisible(*visible: bool*)

Set the state of the categories.

pyqtribbon.gallery

Module Contents

Classes

RibbonPopupWidget	The popup widget for the gallery widget.
RibbonGalleryListWidget	Gallery list widget.
RibbonGalleryButton	Gallery button.
RibbonGalleryPopupListWidget	Gallery popup list widget.
RibbonGallery	A widget that displays a gallery of buttons.

class pyqtribbon.gallery.**RibbonPopupWidget**

Bases: qtpy.QtWidgets.QFrame

The popup widget for the gallery widget.

class pyqtribbon.gallery.**RibbonGalleryListWidget**(*parent=None*)

Bases: qtpy.QtWidgets.QListWidget

Gallery list widget.

resizeEvent(*e: qtpy.QtGui.QResizeEvent*) → None

Resize the list widget.

scrollToNextRow() → None

Scroll to the next row.

scrollToPreviousRow() → None

Scroll to the previous row.

class pyqtribbon.gallery.**RibbonGalleryButton**

Bases: qtpy.QtWidgets.QToolButton

Gallery button.

class pyqtribbon.gallery.**RibbonGalleryPopupListWidget**(*parent=None*)

Bases: *RibbonGalleryListWidget*

Gallery popup list widget.

class pyqtribbon.gallery.**RibbonGallery**(*minimumWidth=800*, *popupHideOnClick=False*, *parent=None*)
RibbonGallery(parent=None)

Bases: qtpy.QtWidgets.QFrame

A widget that displays a gallery of buttons.

_popupWindowSize

_buttons: List[*pyqtribbon.toolbutton.RibbonToolButton*] = []

_popupButtons: List[*pyqtribbon.toolbutton.RibbonToolButton*] = []

_popupHideOnClick = False

_handlePopupAction(*action: qtpy.QtWidgets.QAction*) → None

Handle a popup action.

resizeEvent(*a0: qtpy.QtGui.QResizeEvent*) → None

Resize the gallery.

popupMenu() → *pyqtribbon.menu.RibbonPermanentMenu*

Return the popup menu.

showPopup()

Show the popup window

hidePopupWidget()

Hide the popup window

popupWindowSize()

Return the size of the popup window

Returns
size of the popup window

setPopupWindowSize(*size: qtpy.QtCore.QSize*)

Set the size of the popup window

Parameters
size – size of the popup window

setSelectedButton()

Set the selected button

_addWidget(*widget: qtpy.QtWidgets.QWidget*)

Add a widget to the gallery

Parameters
widget – widget to add

_addPopupWidget(*widget: qtpy.QtWidgets.QWidget*)

Add a widget to the popup gallery

Parameters
widget – widget to add

setPopupHideOnClick(*popupHideOnClick: bool*)

Set the hide on click flag

Parameters
popupHideOnClick – hide on click flag

addButton(*text: str = None*, *icon: qtpy.QtGui.QIcon = None*, *slot=None*, *shortcut=None*, *tooltip=None*, *statusTip=None*, *checkable=False*) → *pyqtribbon.toolbutton.RibbonToolButton*

Add a button to the gallery

Parameters

- **text** – text of the button
- **icon** – icon of the button
- **slot** – slot to call when the button is clicked
- **shortcut** – shortcut of the button
- **tooltip** – tooltip of the button
- **statusTip** – status tip of the button
- **checkable** – checkable flag of the button

Returns
the button added

addToggleButton(*text: str = None*, *icon: qtpy.QtGui.QIcon = None*, *slot=None*, *shortcut=None*, *tooltip=None*, *statusTip=None*) → *pyqtribbon.toolbutton.RibbonToolButton*

Add a toggle button to the gallery

Parameters

- **text** – text of the button
- **icon** – icon of the button
- **slot** – slot to call when the button is clicked
- **shortcut** – shortcut of the button
- **tooltip** – tooltip of the button
- **statusTip** – status tip of the button

Returns
the button added

pyqtribbon.menu

Module Contents

Classes

RibbonMenu	
RibbonPermanentMenu	A permanent menu.

class pyqtribbon.menu.**RibbonMenu**(*title: str = ''*, *parent=None) RibbonMenu(parent=None*)

Bases: qtpy.QtWidgets.QMenu

addWidget(*widget: qtpy.QtWidgets.QWidget*)

Add a widget to the menu.

Parameters
widget – The widget to add.

addHorizontalLayoutWidget() → qtpy.QtWidgets.QHBoxLayout

Add a horizontal layout widget to the menu.

Returns
The horizontal layout.

addVerticalLayoutWidget() → qtpy.QtWidgets.QVBoxLayout

Add a vertical layout widget to the menu.

Returns
The vertical layout.

addGridLayoutWidget() → qtpy.QtWidgets.QGridLayout

Add a grid layout widget to the menu.

Returns
The grid layout.

addFormLayoutWidget() → qtpy.QtWidgets.QFormLayout

Add a form layout widget to the menu.

Returns
The form layout.

addSpacing(*spacing: int = 5*)

Add spacing to the menu.

Parameters
spacing – The spacing.

addLabel(*text: str = ''*, *alignment: qtpy.QtCore.Qt.Alignment = QtCore.Qt.AlignLeft*)

Add a label to the menu.

Parameters

- **text** – The text of the label.
- **alignment** – The alignment of the label.

class pyqtribbon.menu.**RibbonPermanentMenu**(*title: str = ''*, *parent=None*)
RibbonPermanentMenu(parent=None)

Bases: *RibbonMenu*

A permanent menu.

actionAdded

hideEvent(*a0: QHideEvent*) → None

actionEvent(*a0: QActionEvent*) → None

pyqtribbon.panel

Module Contents

Classes

RibbonPanelTitle	Widget to display the title of a panel.
RibbonSpaceFindMode	Mode to find available space in a grid layout, ColumnWise or RowWise.
RibbonGridLayoutManager	Grid Layout Manager.
RibbonPanelItemWidget	Widget to display a panel item.
RibbonPanelOptionButton	Button to display the options of a panel.
RibbonPanel	Panel in the ribbon category.

Attributes

ColumnWise	
RowWise	

class pyqtribbon.panel.**RibbonPanelTitle**

Bases: qtpy.QtWidgets.QLabel

Widget to display the title of a panel.

class pyqtribbon.panel.**RibbonSpaceFindMode**

Bases: enum.IntEnum

Mode to find available space in a grid layout, ColumnWise or RowWise.

ColumnWise = 0

RowWise = 1

pyqtribbon.panel.**ColumnWise**

pyqtribbon.panel.**RowWise**

class pyqtribbon.panel.**RibbonGridLayoutManager**(*rows: int*)

Bases: object

Grid Layout Manager.

request_cells(*rowSpan: int = 1*, *colSpan: int = 1*, *mode=RibbonSpaceFindMode.ColumnWise*)

Request a number of available cells from the grid.

Parameters

- **rowSpan** – The number of rows the cell should span.
- **colSpan** – The number of columns the cell should span.
- **mode** – The mode of the grid.

Returns

row, col, the row and column of the requested cell.

class pyqtribbon.panel.**RibbonPanelItemWidget**(*parent=None*)

Bases: qtpy.QtWidgets.QFrame

Widget to display a panel item.

addWidget(*widget*)

Add a widget to the panel item.

Parameters

widget – The widget to add.

class pyqtribbon.panel.**RibbonPanelOptionButton**

Bases: qtpy.QtWidgets.QToolButton

Button to display the options of a panel.

class pyqtribbon.panel.**RibbonPanel**(*title: str = ''*, *maxRows: int = 6*, *showPanelOptionButton=True*, *parent=None) RibbonPanel(parent=None*)

Bases: qtpy.QtWidgets.QFrame

Panel in the ribbon category.

_maxRows: int = 6

_largeRows: int = 6

_mediumRows: int = 3

_smallRows: int = 2

_gridLayoutManager: *RibbonGridLayoutManager*

_showPanelOptionButton: bool

_widgets: List[qtpy.QtWidgets.QWidget] = []

_titleHeight: int = 20

panelOptionClicked

maximumRows() → int

Return the maximal number of rows in the panel.

Returns
The maximal number of rows in the panel.

largeRows() → int

Return the number of span rows for large widgets.

Returns
The number of span rows for large widgets.

mediumRows() → int

Return the number of span rows for medium widgets.

Returns
The number of span rows for medium widgets.

smallRows() → int

Return the number of span rows for small widgets.

Returns
The number of span rows for small widgets.

setMaximumRows(*maxRows: int*)

Set the maximal number of rows in the panel.

Parameters
maxRows – The maximal number of rows in the panel.

setLargeRows(*rows: int*)

Set the number of span rows for large widgets.

Parameters
rows – The number of span rows for large widgets.

setMediumRows(*rows: int*)

Set the number of span rows for medium widgets.

Parameters
rows – The number of span rows for medium widgets.

setSmallRows(*rows: int*)

Set the number of span rows for small widgets.

Parameters
rows – The number of span rows for small widgets.

defaultRowSpan(*rowSpan: int | pyqtribbon.toolbutton.RibbonButtonStyle*)

Return the number of span rows for the given widget type.

Parameters
rowSpan – row span or type.

Returns
The number of span rows for the given widget type.

panelOptionButton() → *RibbonPanelOptionButton*

Return the panel option button.

Returns
The panel option button.

setPanelOptionToolTip(*text: str*)

Set the tooltip of the panel option button.

Parameters
text – The tooltip text.

rowHeight() → int

Return the height of a row.

addWidgetsBy(*data: Dict[str, Dict]*) → Dict[str, qtpy.QtWidgets.QWidget]

Add widgets to the panel.

Parameters
data – The data to add. The dict is of the form:

```
{
    "widget-name": {
        "type": "Button",
        "arguments": {
            "key1": "value1",
            "key2": "value2"
        }
    },
}
```

Possible types are: Button, SmallButton, MediumButton, LargeButton, ToggleButton, SmallToggleButton, MediumToggleButton, LargeToggleButton, ComboBox, FontComboBox, LineEdit, TextEdit, PlainTextEdit, Label, ProgressBar, SpinBox, DoubleSpinBox, DataEdit, TimeEdit, DateTimeEdit, TableWidget, TreeWidget, ListWidget, CalendarWidget, Separator, HorizontalSeparator, VerticalSeparator, Gallery.

Returns
A dictionary of the added widgets.

addWidget(*widget: qtpy.QtWidgets.QWidget*, *rowSpan: int | pyqtribbon.toolbutton.RibbonButtonStyle = Small*, *colSpan: int = 1*, *mode=RibbonSpaceFindMode.ColumnWise*, *alignment=QtCore.Qt.AlignCenter*, *fixedHeight: bool | float = False*)

Add a widget to the panel.

Parameters

- **widget** – The widget to add.
- **rowSpan** – The number of rows the widget should span, 2: small, 3: medium, 6: large.
- **colSpan** – The number of columns the widget should span.
- **mode** – The mode to find spaces.
- **alignment** – The alignment of the widget.
- **fixedHeight** – Whether to fix the height of the widget, it can be a boolean, a percentage or a fixed height, when a boolean is given, the height is fixed to the maximum height allowed if the value is True, when a percentage is given (0 < percentage < 1) the height is calculated from the height of the maximum height allowed, depends on the number of rows to span. The minimum height is 40% of the maximum height allowed.

addSmallWidget(*widget: qtpy.QtWidgets.QWidget*, *mode=RibbonSpaceFindMode.ColumnWise*, *alignment=QtCore.Qt.AlignCenter*, *fixedHeight: bool | float = False*)

Add a small widget to the panel.

Parameters

- **widget** – The widget to add.
- **mode** – The mode to find spaces.
- **alignment** – The alignment of the widget.
- **fixedHeight** – Whether to fix the height of the widget, it can be a boolean, a percentage or a fixed height, when a boolean is given, the height is fixed to the maximum height allowed if the value is True, when a percentage is given (0 < percentage < 1) the height is calculated from the height of the maximum height allowed, depends on the number of rows to span. The minimum height is 40% of the maximum height allowed.

Returns
The widget that was added.

addMediumWidget(*widget: qtpy.QtWidgets.QWidget*, *mode=RibbonSpaceFindMode.ColumnWise*, *alignment=QtCore.Qt.AlignCenter*, *fixedHeight: bool | float = False*)

Add a medium widget to the panel.

Parameters

- **widget** – The widget to add.
- **mode** – The mode to find spaces.
- **alignment** – The alignment of the widget.
- **fixedHeight** – Whether to fix the height of the widget, it can be a boolean, a percentage or a fixed height, when a boolean is given, the height is fixed to the maximum height allowed if the value is True, when a percentage is given (0 < percentage < 1) the height is calculated from the height of the maximum height allowed, depends on the number of rows to span. The minimum height is 40% of the maximum height allowed.

addLargeWidget(*widget: qtpy.QtWidgets.QWidget*, *mode=RibbonSpaceFindMode.ColumnWise*, *alignment=QtCore.Qt.AlignCenter*, *fixedHeight: bool | float = False*)

Add a large widget to the panel.

Parameters

- **widget** – The widget to add.
- **mode** – The mode to find spaces.
- **alignment** – The alignment of the widget.
- **fixedHeight** – Whether to fix the height of the widget, it can be a boolean, a percentage or a fixed height, when a boolean is given, the height is fixed to the maximum height allowed if the value is True, when a percentage is given (0 < percentage < 1) the height is calculated from the height of the maximum height allowed, depends on the number of rows to span. The minimum height is 40% of the maximum height allowed.

removeWidget(*widget: qtpy.QtWidgets.QWidget*)

Remove a widget from the panel.

widget(*index: int*) → qtpy.QtWidgets.QWidget

Get the widget at the given index.

Parameters
index – The index of the widget, starting from 0.

Returns
The widget at the given index.

widgets() → List[qtpy.QtWidgets.QWidget]

Get all the widgets in the panel.

Returns
A list of all the widgets in the panel.

addButton(*text: str = None*, *icon: qtpy.QtGui.QIcon = None*, *style: pyqtribbon.toolbutton.RibbonButtonStyle = RibbonButtonStyle.Large*, *showText: bool = True*, *colSpan: int = 1*, *slot=None*, *shortcut=None*, *tooltip=None*, *statusTip=None*, *mode=RibbonSpaceFindMode.ColumnWise*, *alignment=QtCore.Qt.AlignCenter*, *fixedHeight: bool | float = False*) → *pyqtribbon.toolbutton.RibbonToolButton*

Add a button to the panel.

Parameters

- **text** – The text of the button.
- **icon** – The icon of the button.
- **style** – The style of the button.
- **showText** – Whether to show the text of the button.
- **colSpan** – The number of columns the button should span.
- **slot** – The slot to call when the button is clicked.
- **shortcut** – The shortcut of the button.
- **tooltip** – The tooltip of the button.
- **statusTip** – The status tip of the button.
- **mode** – The mode to find spaces.
- **alignment** – The alignment of the button.
- **fixedHeight** – Whether to fix the height of the widget, it can be a boolean, a percentage or a fixed height, when a boolean is given, the height is fixed to the maximum height allowed if the value is True, when a percentage is given (0 < percentage < 1) the height is calculated from the height of the maximum height allowed, depends on the number of rows to span. The minimum height is 40% of the maximum height allowed.

Returns
The button that was added.

addSmallButton(*text: str = None*, *icon: qtpy.QtGui.QIcon = None*, *showText: bool = True*, *colSpan: int = 1*, *slot=None*, *shortcut=None*, *tooltip=None*, *statusTip=None*, *mode=RibbonSpaceFindMode.ColumnWise*, *alignment=QtCore.Qt.AlignCenter*, *fixedHeight: bool | float = False*) → *pyqtribbon.toolbutton.RibbonToolButton*

Add a small button to the panel.

Parameters

- **text** – The text of the button.
- **icon** – The icon of the button.
- **showText** – Whether to show the text of the button.
- **colSpan** – The number of columns the button should span.

- **slot** – The slot to call when the button is clicked.
- **shortcut** – The shortcut of the button.
- **tooltip** – The tooltip of the button.
- **statusTip** – The status tip of the button.
- **mode** – The mode to find spaces.
- **alignment** – The alignment of the button.
- **fixedHeight** – Whether to fix the height of the widget, it can be a boolean, a percentage or a fixed height, when a boolean is given, the height is fixed to the maximum height allowed if the value is True, when a percentage is given (0 < percentage < 1) the height is calculated from the height of the maximum height allowed, depends on the number of rows to span. The minimum height is 40% of the maximum height allowed.

Returns
The button that was added.

addMediumButton(*text: str = None*, *icon: qtpy.QtGui.QIcon = None*, *showText: bool = True*, *colSpan: int = 1*, *slot=None*, *shortcut=None*, *tooltip=None*, *statusTip=None*, *mode=RibbonSpaceFindMode.ColumnWise*, *alignment=QtCore.Qt.AlignCenter*, *fixedHeight: bool | float = False*) → *pyqtribbon.toolbutton.RibbonToolButton*

Add a medium button to the panel.

Parameters

- **text** – The text of the button.
- **icon** – The icon of the button.
- **showText** – Whether to show the text of the button.
- **colSpan** – The number of columns the button should span.
- **slot** – The slot to call when the button is clicked.
- **shortcut** – The shortcut of the button.
- **tooltip** – The tooltip of the button.
- **statusTip** – The status tip of the button.
- **mode** – The mode to find spaces.
- **alignment** – The alignment of the button.
- **fixedHeight** – Whether to fix the height of the widget, it can be a boolean, a percentage or a fixed height, when a boolean is given, the height is fixed to the maximum height allowed if the value is True, when a percentage is given (0 < percentage < 1) the height is calculated from the height of the maximum height allowed, depends on the number of rows to span. The minimum height is 40% of the maximum height allowed.

Returns
The button that was added.

addLargeButton(*text: str = None*, *icon: qtpy.QtGui.QIcon = None*, *showText: bool = True*, *colSpan: int = 1*, *slot=None*, *shortcut=None*, *tooltip=None*, *statusTip=None*, *mode=RibbonSpaceFindMode.ColumnWise*, *alignment=QtCore.Qt.AlignCenter*, *fixedHeight: bool | float = False*) → *pyqtribbon.toolbutton.RibbonToolButton*

Add a large button to the panel.

Parameters

- **text** – The text of the button.
- **icon** – The icon of the button.
- **showText** – Whether to show the text of the button.
- **colSpan** – The number of columns the button should span.
- **slot** – The slot to call when the button is clicked.
- **shortcut** – The shortcut of the button.
- **tooltip** – The tooltip of the button.
- **statusTip** – The status tip of the button.
- **mode** – The mode to find spaces.
- **alignment** – The alignment of the button.
- **fixedHeight** – Whether to fix the height of the widget, it can be a boolean, a percentage or a fixed height, when a boolean is given, the height is fixed to the maximum height allowed if the value is True, when a percentage is given (0 < percentage < 1) the height is calculated from the height of the maximum height allowed, depends on the number of rows to span. The minimum height is 40% of the maximum height allowed.

Returns

The button that was added.

addToggleButton(*text: str = None*, *icon: qtpy.QtGui.QIcon = None*, *style:* pyqtribbon.toolbutton.RibbonButtonStyle *= RibbonButtonStyle.Large*, *showText: bool = True*, *colSpan: int = 1*, *slot=None*, *shortcut=None*, *tooltip=None*, *statusTip=None*, *mode=RibbonSpaceFindMode.ColumnWise*, *alignment=QtCore.Qt.AlignCenter*, *fixedHeight: bool | float = False*) → *pyqtribbon.toolbutton.RibbonToolButton*

Add a toggle button to the panel.

Parameters

- **text** – The text of the button.
- **icon** – The icon of the button.
- **style** – The style of the button.
- **showText** – Whether to show the text of the button.
- **colSpan** – The number of columns the button should span.
- **slot** – The slot to call when the button is clicked.
- **shortcut** – The shortcut of the button.
- **tooltip** – The tooltip of the button.
- **statusTip** – The status tip of the button.
- **mode** – The mode to find spaces.
- **alignment** – The alignment of the button.
- **fixedHeight** – Whether to fix the height of the widget, it can be a boolean, a percentage or a fixed height, when a boolean is given, the height is fixed to the maximum height allowed if the value is True, when a percentage is given (0 < percentage < 1) the height is calculated from the height of the maximum height allowed, depends on the number of rows to span. The minimum height is 40% of the maximum height allowed.

Returns
The button that was added.

addSmallToggleButton(*text: str = None*, *icon: qtpy.QtGui.QIcon = None*, *showText: bool = True*, *colSpan: int = 1*, *slot=None*, *shortcut=None*, *tooltip=None*, *statusTip=None*, *mode=RibbonSpaceFindMode.ColumnWise*, *alignment=QtCore.Qt.AlignCenter*, *fixedHeight: bool | float = False*) → *pyqtribbon.toolbutton.RibbonToolButton*

Add a small toggle button to the panel.

Parameters

- **text** – The text of the button.
- **icon** – The icon of the button.
- **showText** – Whether to show the text of the button.
- **colSpan** – The number of columns the button should span.
- **slot** – The slot to call when the button is clicked.
- **shortcut** – The shortcut of the button.
- **tooltip** – The tooltip of the button.
- **statusTip** – The status tip of the button.
- **mode** – The mode to find spaces.
- **alignment** – The alignment of the button.
- **fixedHeight** – Whether to fix the height of the widget, it can be a boolean, a percentage or a fixed height, when a boolean is given, the height is fixed to the maximum height allowed if the value is True, when a percentage is given (0 < percentage < 1) the height is calculated from the height of the maximum height allowed, depends on the number of rows to span. The minimum height is 40% of the maximum height allowed.

Returns
The button that was added.

addMediumToggleButton(*text: str = None*, *icon: qtpy.QtGui.QIcon = None*, *showText: bool = True*, *colSpan: int = 1*, *slot=None*, *shortcut=None*, *tooltip=None*, *statusTip=None*, *mode=RibbonSpaceFindMode.ColumnWise*, *alignment=QtCore.Qt.AlignCenter*, *fixedHeight: bool | float = False*) → *pyqtribbon.toolbutton.RibbonToolButton*

Add a medium toggle button to the panel.

Parameters

- **text** – The text of the button.
- **icon** – The icon of the button.
- **showText** – Whether to show the text of the button.
- **colSpan** – The number of columns the button should span.
- **slot** – The slot to call when the button is clicked.
- **shortcut** – The shortcut of the button.
- **tooltip** – The tooltip of the button.
- **statusTip** – The status tip of the button.
- **mode** – The mode to find spaces.

- **alignment** – The alignment of the button.
- **fixedHeight** – Whether to fix the height of the widget, it can be a boolean, a percentage or a fixed height, when a boolean is given, the height is fixed to the maximum height allowed if the value is True, when a percentage is given (0 < percentage < 1) the height is calculated from the height of the maximum height allowed, depends on the number of rows to span. The minimum height is 40% of the maximum height allowed.

Returns
The button that was added.

addLargeToggleButton(*text: str = None*, *icon: qtpy.QtGui.QIcon = None*, *showText: bool = True*, *colSpan: int = 1*, *slot=None*, *shortcut=None*, *tooltip=None*, *statusTip=None*, *mode=RibbonSpaceFindMode.ColumnWise*, *alignment=QtCore.Qt.AlignCenter*, *fixedHeight: bool | float = False*) → *pyqtribbon.toolbutton.RibbonToolButton*

Add a large toggle button to the panel.

Parameters

- **text** – The text of the button.
- **icon** – The icon of the button.
- **showText** – Whether to show the text of the button.
- **colSpan** – The number of columns the button should span.
- **slot** – The slot to call when the button is clicked.
- **shortcut** – The shortcut of the button.
- **tooltip** – The tooltip of the button.
- **statusTip** – The status tip of the button.
- **mode** – The mode to find spaces.
- **alignment** – The alignment of the button.
- **fixedHeight** – Whether to fix the height of the widget, it can be a boolean, a percentage or a fixed height, when a boolean is given, the height is fixed to the maximum height allowed if the value is True, when a percentage is given (0 < percentage < 1) the height is calculated from the height of the maximum height allowed, depends on the number of rows to span. The minimum height is 40% of the maximum height allowed.

Returns
The button that was added.

addComboBox(*items: List[str]*, *rowSpan: int | pyqtribbon.toolbutton.RibbonButtonStyle = Small*, *colSpan: int = 1*, *mode=RibbonSpaceFindMode.ColumnWise*, *alignment=QtCore.Qt.AlignCenter*, *fixedHeight: bool | float = False*) → qtpy.QtWidgets.QComboBox

Add a combo box to the panel.

Parameters

- **items** – The items of the combo box.
- **rowSpan** – The number of rows the combo box should span.
- **colSpan** – The number of columns the combo box should span.
- **mode** – The mode to find spaces.
- **alignment** – The alignment of the combo box.

- **fixedHeight** – Whether to fix the height of the widget, it can be a boolean, a percentage or a fixed height, when a boolean is given, the height is fixed to the maximum height allowed if the value is True, when a percentage is given (0 < percentage < 1) the height is calculated from the height of the maximum height allowed, depends on the number of rows to span. The minimum height is 40% of the maximum height allowed.

Returns
The combo box that was added.

addFontComboBox(*rowSpan: int | pyqtribbon.toolbutton.RibbonButtonStyle = Small*, *colSpan: int = 1*, *mode=RibbonSpaceFindMode.ColumnWise*, *alignment=QtCore.Qt.AlignCenter*, *fixedHeight: bool | float = False*) → qtpy.QtWidgets.QFontComboBox

Add a font combo box to the panel.

Parameters

- **rowSpan** – The number of rows the combo box should span.
- **colSpan** – The number of columns the combo box should span.
- **mode** – The mode to find spaces.
- **alignment** – The alignment of the combo box.
- **fixedHeight** – Whether to fix the height of the widget, it can be a boolean, a percentage or a fixed height, when a boolean is given, the height is fixed to the maximum height allowed if the value is True, when a percentage is given (0 < percentage < 1) the height is calculated from the height of the maximum height allowed, depends on the number of rows to span. The minimum height is 40% of the maximum height allowed.

Returns
The combo box that was added.

addLineEdit(*rowSpan: int | pyqtribbon.toolbutton.RibbonButtonStyle = Small*, *colSpan: int = 1*, *mode=RibbonSpaceFindMode.ColumnWise*, *alignment=QtCore.Qt.AlignCenter*, *fixedHeight: bool | float = False*) → qtpy.QtWidgets.QLineEdit

Add a line edit to the panel.

Parameters

- **rowSpan** – The number of rows the line edit should span.
- **colSpan** – The number of columns the line edit should span.
- **mode** – The mode to find spaces.
- **alignment** – The alignment of the line edit.
- **fixedHeight** – Whether to fix the height of the widget, it can be a boolean, a percentage or a fixed height, when a boolean is given, the height is fixed to the maximum height allowed if the value is True, when a percentage is given (0 < percentage < 1) the height is calculated from the height of the maximum height allowed, depends on the number of rows to span. The minimum height is 40% of the maximum height allowed.

Returns
The line edit that was added.

addTextEdit(*rowSpan: int | pyqtribbon.toolbutton.RibbonButtonStyle = Small*, *colSpan: int = 1*, *mode=RibbonSpaceFindMode.ColumnWise*, *alignment=QtCore.Qt.AlignCenter*, *fixedHeight: bool | float = False*) → qtpy.QtWidgets.QTextEdit

Add a text edit to the panel.

Parameters

- **rowSpan** – The number of rows the text edit should span.
- **colSpan** – The number of columns the text edit should span.
- **mode** – The mode to find spaces.
- **alignment** – The alignment of the text edit.
- **fixedHeight** – Whether to fix the height of the widget, it can be a boolean, a percentage or a fixed height, when a boolean is given, the height is fixed to the maximum height allowed if the value is True, when a percentage is given (0 < percentage < 1) the height is calculated from the height of the maximum height allowed, depends on the number of rows to span. The minimum height is 40% of the maximum height allowed.

Returns

The text edit that was added.

addPlainTextEdit(*rowSpan: int | pyqtribbon.toolbutton.RibbonButtonStyle = Small*, *colSpan: int = 1*, *mode=RibbonSpaceFindMode.ColumnWise*, *alignment=QtCore.Qt.AlignCenter*, *fixedHeight: bool | float = False*) → qtpy.QtWidgets.QPlainTextEdit

Add a plain text edit to the panel.

Parameters

- **rowSpan** – The number of rows the text edit should span.
- **colSpan** – The number of columns the text edit should span.
- **mode** – The mode to find spaces.
- **alignment** – The alignment of the text edit.
- **fixedHeight** – Whether to fix the height of the widget, it can be a boolean, a percentage or a fixed height, when a boolean is given, the height is fixed to the maximum height allowed if the value is True, when a percentage is given (0 < percentage < 1) the height is calculated from the height of the maximum height allowed, depends on the number of rows to span. The minimum height is 40% of the maximum height allowed.

Returns

The text edit that was added.

addLabel(*text: str*, *rowSpan: int | pyqtribbon.toolbutton.RibbonButtonStyle = Small*, *colSpan: int = 1*, *mode=RibbonSpaceFindMode.ColumnWise*, *alignment=QtCore.Qt.AlignCenter*, *fixedHeight: bool | float = False*) → qtpy.QtWidgets.QLabel

Add a label to the panel.

Parameters

- **text** – The text of the label.
- **rowSpan** – The number of rows the label should span.
- **colSpan** – The number of columns the label should span.
- **mode** – The mode to find spaces.
- **alignment** – The alignment of the label.
- **fixedHeight** – Whether to fix the height of the widget, it can be a boolean, a percentage or a fixed height, when a boolean is given, the height is fixed to the maximum height allowed if the value is True, when a percentage is given (0 < percentage < 1) the height is calculated

from the height of the maximum height allowed, depends on the number of rows to span. The minimum height is 40% of the maximum height allowed.

Returns
The label that was added.

addProgressBar(*rowSpan: int | pyqtribbon.toolbutton.RibbonButtonStyle = Small*, *colSpan: int = 1*, *mode=RibbonSpaceFindMode.ColumnWise*, *alignment=QtCore.Qt.AlignCenter*, *fixedHeight: bool | float = False*) → qtpy.QtWidgets.QProgressBar

Add a progress bar to the panel.

Parameters

- **rowSpan** – The number of rows the progress bar should span.
- **colSpan** – The number of columns the progress bar should span.
- **mode** – The mode to find spaces.
- **alignment** – The alignment of the progress bar.
- **fixedHeight** – Whether to fix the height of the widget, it can be a boolean, a percentage or a fixed height, when a boolean is given, the height is fixed to the maximum height allowed if the value is True, when a percentage is given (0 < percentage < 1) the height is calculated from the height of the maximum height allowed, depends on the number of rows to span. The minimum height is 40% of the maximum height allowed.

Returns
The progress bar that was added.

addSlider(*rowSpan: int | pyqtribbon.toolbutton.RibbonButtonStyle = Small*, *colSpan: int = 1*, *mode=RibbonSpaceFindMode.ColumnWise*, *alignment=QtCore.Qt.AlignCenter*, *fixedHeight: bool | float = False*) → qtpy.QtWidgets.QSlider

Add a slider to the panel.

Parameters

- **rowSpan** – The number of rows the slider should span.
- **colSpan** – The number of columns the slider should span.
- **mode** – The mode to find spaces.
- **alignment** – The alignment of the slider.
- **fixedHeight** – Whether to fix the height of the widget, it can be a boolean, a percentage or a fixed height, when a boolean is given, the height is fixed to the maximum height allowed if the value is True, when a percentage is given (0 < percentage < 1) the height is calculated from the height of the maximum height allowed, depends on the number of rows to span. The minimum height is 40% of the maximum height allowed.

Returns
The slider that was added.

addSpinBox(*rowSpan: int | pyqtribbon.toolbutton.RibbonButtonStyle = Small*, *colSpan: int = 1*, *mode=RibbonSpaceFindMode.ColumnWise*, *alignment=QtCore.Qt.AlignCenter*, *fixedHeight: bool | float = False*) → qtpy.QtWidgets.QSpinBox

Add a spin box to the panel.

Parameters

- **rowSpan** – The number of rows the spin box should span.

- **colSpan** – The number of columns the spin box should span.
- **mode** – The mode to find spaces.
- **alignment** – The alignment of the spin box.
- **fixedHeight** – Whether to fix the height of the widget, it can be a boolean, a percentage or a fixed height, when a boolean is given, the height is fixed to the maximum height allowed if the value is True, when a percentage is given (0 < percentage < 1) the height is calculated from the height of the maximum height allowed, depends on the number of rows to span. The minimum height is 40% of the maximum height allowed.

Returns
The spin box that was added.

addDoubleSpinBox(*rowSpan: int | pyqtribbon.toolbutton.RibbonButtonStyle = Small*, *colSpan: int = 1*, *mode=RibbonSpaceFindMode.ColumnWise*, *alignment=QtCore.Qt.AlignCenter*, *fixedHeight: bool | float = False*) → qtpy.QtWidgets.QDoubleSpinBox

Add a double spin box to the panel.

Parameters

- **rowSpan** – The number of rows the double spin box should span.
- **colSpan** – The number of columns the double spin box should span.
- **mode** – The mode to find spaces.
- **alignment** – The alignment of the double spin box.
- **fixedHeight** – Whether to fix the height of the widget, it can be a boolean, a percentage or a fixed height, when a boolean is given, the height is fixed to the maximum height allowed if the value is True, when a percentage is given (0 < percentage < 1) the height is calculated from the height of the maximum height allowed, depends on the number of rows to span. The minimum height is 40% of the maximum height allowed.

Returns
The double spin box that was added.

addDateEdit(*rowSpan: int | pyqtribbon.toolbutton.RibbonButtonStyle = Small*, *colSpan: int = 1*, *mode=RibbonSpaceFindMode.ColumnWise*, *alignment=QtCore.Qt.AlignCenter*, *fixedHeight: bool | float = False*) → qtpy.QtWidgets.QDateEdit

Add a date edit to the panel.

Parameters

- **rowSpan** – The number of rows the date edit should span.
- **colSpan** – The number of columns the date edit should span.
- **mode** – The mode to find spaces.
- **alignment** – The alignment of the date edit.
- **fixedHeight** – Whether to fix the height of the widget, it can be a boolean, a percentage or a fixed height, when a boolean is given, the height is fixed to the maximum height allowed if the value is True, when a percentage is given (0 < percentage < 1) the height is calculated from the height of the maximum height allowed, depends on the number of rows to span. The minimum height is 40% of the maximum height allowed.

Returns
The date edit that was added.

addTimeEdit(*rowSpan: int | pyqtribbon.toolbutton.RibbonButtonStyle = Small*, *colSpan: int = 1*, *mode=RibbonSpaceFindMode.ColumnWise*, *alignment=QtCore.Qt.AlignCenter*, *fixedHeight: bool | float = False*) → qtpy.QtWidgets.QTimeEdit

Add a time edit to the panel.

Parameters

- **rowSpan** – The number of rows the time edit should span.
- **colSpan** – The number of columns the time edit should span.
- **mode** – The mode to find spaces.
- **alignment** – The alignment of the time edit.
- **fixedHeight** – Whether to fix the height of the widget, it can be a boolean, a percentage or a fixed height, when a boolean is given, the height is fixed to the maximum height allowed if the value is True, when a percentage is given (0 < percentage < 1) the height is calculated from the height of the maximum height allowed, depends on the number of rows to span. The minimum height is 40% of the maximum height allowed.

Returns

The time edit that was added.

addDateTimeEdit(*rowSpan: int | pyqtribbon.toolbutton.RibbonButtonStyle = Small*, *colSpan: int = 1*, *mode=RibbonSpaceFindMode.ColumnWise*, *alignment=QtCore.Qt.AlignCenter*, *fixedHeight: bool | float = False*) → qtpy.QtWidgets.QDateTimeEdit

Add a date time edit to the panel.

Parameters

- **rowSpan** – The number of rows the date time edit should span.
- **colSpan** – The number of columns the date time edit should span.
- **mode** – The mode to find spaces.
- **alignment** – The alignment of the date time edit.
- **fixedHeight** – Whether to fix the height of the widget, it can be a boolean, a percentage or a fixed height, when a boolean is given, the height is fixed to the maximum height allowed if the value is True, when a percentage is given (0 < percentage < 1) the height is calculated from the height of the maximum height allowed, depends on the number of rows to span. The minimum height is 40% of the maximum height allowed.

Returns

The date time edit that was added.

addTableWidget(*rowSpan: int | pyqtribbon.toolbutton.RibbonButtonStyle = Large*, *colSpan: int = 1*, *mode=RibbonSpaceFindMode.ColumnWise*, *alignment=QtCore.Qt.AlignCenter*, *fixedHeight: bool | float = False*) → qtpy.QtWidgets.QTableWidget

Add a table widget to the panel.

Parameters

- **rowSpan** – The number of rows the table widget should span.
- **colSpan** – The number of columns the table widget should span.
- **mode** – The mode to find spaces.
- **alignment** – The alignment of the table widget.

- **fixedHeight** – Whether to fix the height of the widget, it can be a boolean, a percentage or a fixed height, when a boolean is given, the height is fixed to the maximum height allowed if the value is True, when a percentage is given (0 < percentage < 1) the height is calculated from the height of the maximum height allowed, depends on the number of rows to span. The minimum height is 40% of the maximum height allowed.

Returns
The table widget that was added.

addTreeWidget(*rowSpan: int | pyqtribbon.toolbutton.RibbonButtonStyle = Large, colSpan: int = 1, mode=RibbonSpaceFindMode.ColumnWise, alignment=QtCore.Qt.AlignCenter, fixedHeight: bool | float = False*) → qtpy.QtWidgets.QTreeWidget

Add a tree widget to the panel.

Parameters

- **rowSpan** – The number of rows the tree widget should span.
- **colSpan** – The number of columns the tree widget should span.
- **mode** – The mode to find spaces.
- **alignment** – The alignment of the tree widget.
- **fixedHeight** – Whether to fix the height of the widget, it can be a boolean, a percentage or a fixed height, when a boolean is given, the height is fixed to the maximum height allowed if the value is True, when a percentage is given (0 < percentage < 1) the height is calculated from the height of the maximum height allowed, depends on the number of rows to span. The minimum height is 40% of the maximum height allowed.

Returns
The tree widget that was added.

addListWidget(*rowSpan: int | pyqtribbon.toolbutton.RibbonButtonStyle = Large, colSpan: int = 1, mode=RibbonSpaceFindMode.ColumnWise, alignment=QtCore.Qt.AlignCenter, fixedHeight: bool | float = False*) → qtpy.QtWidgets.QListWidget

Add a list widget to the panel.

Parameters

- **rowSpan** – The number of rows the list widget should span.
- **colSpan** – The number of columns the list widget should span.
- **mode** – The mode to find spaces.
- **alignment** – The alignment of the list widget.
- **fixedHeight** – Whether to fix the height of the widget, it can be a boolean, a percentage or a fixed height, when a boolean is given, the height is fixed to the maximum height allowed if the value is True, when a percentage is given (0 < percentage < 1) the height is calculated from the height of the maximum height allowed, depends on the number of rows to span. The minimum height is 40% of the maximum height allowed.

Returns
The list widget that was added.

addCalendarWidget(*rowSpan: int | pyqtribbon.toolbutton.RibbonButtonStyle = Large, colSpan: int = 1, mode=RibbonSpaceFindMode.ColumnWise, alignment=QtCore.Qt.AlignCenter, fixedHeight: bool | float = False*) → qtpy.QtWidgets.QCalendarWidget

Add a calendar widget to the panel.

Parameters

- **rowSpan** – The number of rows the calendar widget should span.
- **colSpan** – The number of columns the calendar widget should span.
- **mode** – The mode to find spaces.
- **alignment** – The alignment of the calendar widget.
- **fixedHeight** – Whether to fix the height of the widget, it can be a boolean, a percentage or a fixed height, when a boolean is given, the height is fixed to the maximum height allowed if the value is True, when a percentage is given (0 < percentage < 1) the height is calculated from the height of the maximum height allowed, depends on the number of rows to span. The minimum height is 40% of the maximum height allowed.

Returns

The calendar widget that was added.

addSeparator(*orientation=QtCore.Qt.Vertical*, *width=6*, *rowSpan: int* | pyqtribbon.toolbutton.RibbonButtonStyle = *Large*, *colSpan: int = 1*, *mode=RibbonSpaceFindMode.ColumnWise*, *alignment=QtCore.Qt.AlignCenter*, *fixedHeight: bool* | *float = False*) → *pyqtribbon.separator.RibbonHorizontalSeparator* | *pyqtribbon.separator.RibbonVerticalSeparator*

Add a separator to the panel.

Parameters

- **orientation** – The orientation of the separator.
- **width** – The width of the separator.
- **rowSpan** – The number of rows the separator spans.
- **colSpan** – The number of columns the separator spans.
- **mode** – The mode to find spaces.
- **alignment** – The alignment of the separator.
- **fixedHeight** – Whether to fix the height of the widget, it can be a boolean, a percentage or a fixed height, when a boolean is given, the height is fixed to the maximum height allowed if the value is True, when a percentage is given (0 < percentage < 1) the height is calculated from the height of the maximum height allowed, depends on the number of rows to span. The minimum height is 40% of the maximum height allowed.

Returns

The separator.

addHorizontalSeparator(*width=6*, *rowSpan: int* | pyqtribbon.toolbutton.RibbonButtonStyle = *Small*, *colSpan: int = 2*, *mode=RibbonSpaceFindMode.ColumnWise*, *alignment=QtCore.Qt.AlignCenter*, *fixedHeight: bool* | *float = False*) → *pyqtribbon.separator.RibbonHorizontalSeparator*

Add a horizontal separator to the panel.

Parameters

- **width** – The width of the separator.
- **rowSpan** – The number of rows the separator spans.
- **colSpan** – The number of columns the separator spans.
- **mode** – The mode to find spaces.

- **alignment** – The alignment of the separator.
- **fixedHeight** – Whether to fix the height of the widget, it can be a boolean, a percentage or a fixed height, when a boolean is given, the height is fixed to the maximum height allowed if the value is True, when a percentage is given (0 < percentage < 1) the height is calculated from the height of the maximum height allowed, depends on the number of rows to span. The minimum height is 40% of the maximum height allowed.

Returns
The separator.

addVerticalSeparator(*width=6*, *rowSpan: int | pyqtribbon.toolbutton.RibbonButtonStyle = Large*, *colSpan: int = 1*, *mode=RibbonSpaceFindMode.ColumnWise*, *alignment=QtCore.Qt.AlignCenter*, *fixedHeight: bool | float = False*) → *pyqtribbon.separator.RibbonVerticalSeparator*

Add a vertical separator to the panel.

Parameters

- **width** – The width of the separator.
- **rowSpan** – The number of rows the separator spans.
- **colSpan** – The number of columns the separator spans.
- **mode** – The mode to find spaces.
- **alignment** – The alignment of the separator.
- **fixedHeight** – Whether to fix the height of the widget, it can be a boolean, a percentage or a fixed height, when a boolean is given, the height is fixed to the maximum height allowed if the value is True, when a percentage is given (0 < percentage < 1) the height is calculated from the height of the maximum height allowed, depends on the number of rows to span. The minimum height is 40% of the maximum height allowed.

Returns
The separator.

addGallery(*minimumWidth=800*, *popupHideOnClick=False*, *rowSpan: int | pyqtribbon.toolbutton.RibbonButtonStyle = Large*, *colSpan: int = 1*, *mode=RibbonSpaceFindMode.ColumnWise*, *alignment=QtCore.Qt.AlignCenter*, *fixedHeight: bool | float = False*) → *pyqtribbon.gallery.RibbonGallery*

Add a gallery to the panel.

Parameters

- **minimumWidth** – The minimum width of the gallery.
- **popupHideOnClick** – Whether the gallery popup should be hidden when a user clicks on it.
- **rowSpan** – The number of rows the gallery spans.
- **colSpan** – The number of columns the gallery spans.
- **mode** – The mode of the gallery.
- **alignment** – The alignment of the gallery.
- **fixedHeight** – Whether to fix the height of the widget, it can be a boolean, a percentage or a fixed height, when a boolean is given, the height is fixed to the maximum height allowed if the value is True, when a percentage is given (0 < percentage < 1) the height is calculated

from the height of the maximum height allowed, depends on the number of rows to span. The minimum height is 40% of the maximum height allowed.

Returns
The gallery.

`setTitle`(*title: str*)

Set the title of the panel.

Parameters
title – The title to set.

`title`()

Get the title of the panel.

Returns
The title.

pyqtribbon.ribbonbar

Module Contents

Classes

RibbonStyle	Enum where members are also (and must be) ints
RibbonStackedWidget	Stacked widget that is used to display the ribbon.
RibbonBar	The RibbonBar class is the top level widget that contains the ribbon.

Attributes

Debug
Default

class pyqtribbon.ribbonbar.**RibbonStyle**

Bases: enum.IntEnum

Enum where members are also (and must be) ints

Default = 0

Debug = 1

pyqtribbon.ribbonbar.**Debug**

pyqtribbon.ribbonbar.**Default**

class pyqtribbon.ribbonbar.**RibbonStackedWidget**(*parent=None*)

Bases: qtpy.QtWidgets.QStackedWidget

Stacked widget that is used to display the ribbon.

class pyqtribbon.ribbonbar.**RibbonBar**(*title: str = ''*, *maxRows=6*, *parent=None) RibbonBar(parent=None)*

Bases: qtpy.QtWidgets.QMenuBar

The RibbonBar class is the top level widget that contains the ribbon.

helpButtonClicked

fileButtonClicked

_categories: Dict[str, *pyqtribbon.category.RibbonCategory*]

_contextCategoryCount = 0

_maxRows = 6

_ribbonVisible = True

_ribbonHeight = 200

_currentTabIndex = 0

abstract actionAt(*QPoint*)

abstract actionGeometry(*QAction*)

abstract activeAction()

abstract addMenu(**__args*)

abstract addAction(**__args*)

abstract addSeparator()

abstract clear()

abstract cornerWidget(*corner=None*)

abstract insertMenu(*QAction*, *QMenu*)

abstract insertSeparator(*QAction*)

abstract isDefaultUp()

abstract isNativeMenuBar()

abstract setActiveAction(*QAction*)

abstract setCornerWidget(*QWidget*, *corner=None*)

abstract setDefaultUp(*up*)

abstract setNativeMenuBar(*bar*)

setRibbonStyle(*style:* RibbonStyle)

Set the style of the ribbon.

Parameters

style – The style to set.

applicationOptionButton() → *pyqtribbon.titlewidget.RibbonApplicationButton*

Return the application button.

setApplicationIcon(*icon: qtpy.QtGui.QIcon*)

Set the application icon.

Parameters
icon – The icon to set.

addTitleWidget(*widget: qtpy.QtWidgets.QWidget*)

Add a widget to the title widget.

Parameters
widget – The widget to add.

removeTitleWidget(*widget: qtpy.QtWidgets.QWidget*)

Remove a widget from the title widget.

Parameters
widget – The widget to remove.

insertTitleWidget(*index: int*, *widget: qtpy.QtWidgets.QWidget*)

Insert a widget to the title widget.

Parameters

- **index** – The index to insert the widget.
- **widget** – The widget to insert.

addFileMenu() → *pyqtribbon.menu.RibbonMenu*

Add a file menu to the ribbon.

ribbonHeight() → int

Get the total height of the ribbon.

Returns
The height of the ribbon.

setRibbonHeight(*height: int*)

Set the total height of the ribbon.

Parameters
height – The height to set.

tabBar() → *pyqtribbon.tabbar.RibbonTabBar*

Return the tab bar of the ribbon.

Returns
The tab bar of the ribbon.

quickAccessToolBar() → qtpy.QtWidgets.QToolBar

Return the quick access toolbar of the ribbon.

Returns
The quick access toolbar of the ribbon.

addQuickAccessButton(*button: qtpy.QtWidgets.QToolButton*)

Add a button to the quick access bar.

Parameters
button – The button to add.

setQuickAccessButtonHeight(*height: int = 30*)

Set the height of the quick access buttons.

Parameters
height – The height to set.

title() → str

Return the title of the ribbon.

Returns
The title of the ribbon.

setTitle(*title: str*)

Set the title of the ribbon.

Parameters
title – The title to set.

rightToolBar() → qtpy.QtWidgets.QToolBar

Return the right toolbar of the ribbon.

Returns
The right toolbar of the ribbon.

addRightToolButton(*button: qtpy.QtWidgets.QToolButton*)

Add a widget to the right button bar.

Parameters
button – The button to add.

setRightToolBarHeight(*height: int = 24*)

Set the height of the right buttons.

Parameters
height – The height to set.

helpRibbonButton() → qtpy.QtWidgets.QToolButton

Return the help button of the ribbon.

Returns
The help button of the ribbon.

setHelpButtonIcon(*icon: qtpy.QtGui.QIcon*)

Set the icon of the help button.

Parameters
icon – The icon to set.

removeHelpButton()

Remove the help button from the ribbon.

collapseRibbonButton() → qtpy.QtWidgets.QToolButton

Return the collapse ribbon button.

Returns
The collapse ribbon button.

setCollapseButtonIcon(*icon: qtpy.QtGui.QIcon*)

Set the icon of the min button.

Parameters
icon – The icon to set.

removeCollapseButton()

Remove the min button from the ribbon.

category(*name: str*) → *pyqtribbon.category.RibbonCategory*

Return the category with the given name.

Parameters

name – The name of the category.

Returns

The category with the given name.

categories() → Dict[str, *pyqtribbon.category.RibbonCategory*]

Return a list of categories of the ribbon.

Returns

A dict of categories of the ribbon.

addCategoriesBy(*data: Dict[str, Dict]*) → Dict[str, *pyqtribbon.category.RibbonCategory*]

Add categories from a dict.

Parameters

data – The dict of categories. The dict is of the form:

```
{
    "category-title": {
        "style": RibbonCategoryStyle.Normal,
        "color": QtCore.Qt.red,
        "panels": {
            "panel-title": {
                "showPanelOptionButton": True,
                "widgets": {
                    "widget-name": {
                        "type": "Button",
                        "arguments": {
                            "key1": "value1",
                            "key2": "value2"
                        }
                    },
                }
            },
        }
    }
}
```

Returns

A dict of categories of the ribbon.

addCategory(*title: str*, *style=RibbonCategoryStyle.Normal*, *color: qtpy.QtGui.QColor = None*) → *pyqtribbon.category.RibbonNormalCategory* | *pyqtribbon.category.RibbonContextCategory*

Add a new category to the ribbon.

Parameters

- **title** – The title of the category.
- **style** – The button style of the category.

- **color** – The color of the context category, only used if style is Context, if None, the default color will be used.

Returns
The newly created category.

addNormalCategory(*title: str*) → *pyqtribbon.category.RibbonNormalCategory*

Add a new category to the ribbon.

Parameters
title – The title of the category.

Returns
The newly created category.

addContextCategory(*title: str*, *color: qtpy.QtGui.QColor | qtpy.QtCore.Qt.GlobalColor = QtCore.Qt.blue*) → *pyqtribbon.category.RibbonContextCategory*

Add a new context category to the ribbon.

Parameters

- **title** – The title of the category.
- **color** – The color of the context category, if None, the default color will be used.

Returns
The newly created category.

addContextCategories(*name: str*, *titles: List[str]*, *color: qtpy.QtGui.QColor | qtpy.QtCore.Qt.GlobalColor = QtCore.Qt.blue*) → *pyqtribbon.category.RibbonContextCategories*

Add a group of context categories with the same tab color to the ribbon.

Parameters

- **name** – The name of the context categories.
- **titles** – The title of the category.
- **color** – The color of the context category, if None, the default color will be used.

Returns
The newly created category.

showCategoryByIndex(*index: int*)

Show category by tab index

Parameters
index – tab index

showContextCategory(*category:* pyqtribbon.category.RibbonContextCategory | pyqtribbon.category.RibbonContextCategories)

Show the given category or categories, if it is not a context category, nothing happens.

Parameters
category – The category to show.

hideContextCategory(*category:* pyqtribbon.category.RibbonContextCategory | pyqtribbon.category.RibbonContextCategories)

Hide the given category or categories, if it is not a context category, nothing happens.

Parameters
category – The category to hide.

categoryVisible(*category:* pyqtribbon.category.RibbonCategory) → bool

Return whether the category is shown.

Parameters
category – The category to check.

Returns
Whether the category is shown.

removeCategory(*category:* pyqtribbon.category.RibbonCategory)

Remove a category from the ribbon.

Parameters
category – The category to remove.

removeCategories(*categories:* pyqtribbon.category.RibbonContextCategories)

Remove a list of categories from the ribbon.

Parameters
categories – The categories to remove.

setCurrentCategory(*category:* pyqtribbon.category.RibbonCategory)

Set the current category.

Parameters
category – The category to set.

currentCategory() → *pyqtribbon.category.RibbonCategory*

Return the current category.

Returns
The current category.

minimumSizeHint() → qtpy.QtCore.QSize

Return the minimum size hint of the widget.

Returns
The minimum size hint.

_collapseButtonClicked()

showRibbon()

Show the ribbon.

hideRibbon()

Hide the ribbon.

ribbonVisible() → bool

Get the visibility of the ribbon.

Returns
True if the ribbon is visible, False otherwise.

setRibbonVisible(*visible: bool*)

Set the visibility of the ribbon.

Parameters
visible – True to show the ribbon, False to hide it.

pyqtribbon.screenshotwindow

Module Contents

Classes

RibbonScreenShotWindow	This class is just for taking a screenshot of the window, the window will be closed 0.1s after it is shown.

class pyqtribbon.screenshotwindow.**RibbonScreenShotWindow**(*fileName: str = 'shot.jpg'*, **args*, ***kwargs*)

Bases: qtpy.QtWidgets.QMainWindow

This class is just for taking a screenshot of the window, the window will be closed 0.1s after it is shown.

_fileName = 'shot.jpg'

setScreenShotFileName(*fileName: str*)

Set the file name for the screenshot.

Parameters
fileName – The file name for the screenshot.

takeScreenShot()

Take a screenshot of the window.

pyqtribbon.separator

Module Contents

Classes

RibbonSeparator	The RibbonSeparator is a separator that can be used to separate widgets in a ribbon.
RibbonHorizontalSeparator	Horizontal separator.
RibbonVerticalSeparator	Vertical separator.

class pyqtribbon.separator.**RibbonSeparator**(*orientation=QtCore.Qt.Vertical*, *width=6*, *parent=None)*
RibbonSeparator(parent=None)

Bases: qtpy.QtWidgets.QFrame

The RibbonSeparator is a separator that can be used to separate widgets in a ribbon.

_topMargins: int = 4

_bottomMargins: int = 4

_leftMargins: int = 4

_rightMargins: int = 4

_orientation: qtpy.QtCore.Qt.Orientation

sizeHint() → qtpy.QtCore.QSize

Return the size hint.

setTopBottomMargins(*top: int*, *bottom: int*) → None

Set the top and bottom margins.

paintEvent(*event: qtpy.QtGui.QPaintEvent*) → None

Paint the separator.

class pyqtribbon.separator.**RibbonHorizontalSeparator**(*width: int = 6*, *parent=None*)

Bases: *RibbonSeparator*

Horizontal separator.

class pyqtribbon.separator.**RibbonVerticalSeparator**(*width: int = 6*, *parent=None*)

Bases: *RibbonSeparator*

Vertical separator.

pyqtribbon.tabbar

Module Contents

Classes

RibbonTabBar	The TabBar for the title widget.

class pyqtribbon.tabbar.**RibbonTabBar**(*parent=None*)

Bases: qtpy.QtWidgets.QTabBar

The TabBar for the title widget.

_contextCategoryTopMargin = 0

_contextCategoryDarkColorHeight = 5

_tabColors: Dict[str, qtpy.QtCore.Qt.GlobalColor | qtpy.QtGui.QColor]

_associated_tabs

indexOf(*tabName: str*) → int

Return the index of the tab with the given name.

Parameters
tabName – The name of the tab.

Returns
The index of the tab.

tabTitles() → List[str]

Return the titles of all tabs.

Returns
The titles of all tabs.

addTab(*text: str*, *color: qtpy.QtGui.QColor = None*) → int

Add a new tab to the tab bar.

Parameters

- **text** – The text of the tab.
- **color** – The color of the tab.

Returns
The index of the tab.

addAssociatedTabs(*name: str*, *texts: List[str]*, *color: qtpy.QtGui.QColor*) → List[int]

Add associated multiple tabs which have the same color to the tab bar.

Parameters

- **name** – The name of the context category.
- **texts** – The texts of the tabs.
- **color** – The color of the tabs.

Returns
The indices of the tabs.

removeAssociatedTabs(*titles: List[str]*) → None

Remove tabs with the given titles.

Parameters
titles – The titles of the tabs to remove.

currentTabColor() → qtpy.QtGui.QColor

Current tab color

Returns
Current tab color

paintEvent(*a0: qtpy.QtGui.QPaintEvent*) → None

Paint the tab bar.

pyqtribbon.titlewidget

Module Contents

Classes

RibbonApplicationButton	Application button in the ribbon bar.
RibbonTitleLabel	Title label in the ribbon bar.
RibbonTitleWidget	The title widget of the ribbon.

class pyqtribbon.titlewidget.**RibbonApplicationButton**

Bases: qtpy.QtWidgets.QToolButton

Application button in the ribbon bar.

addFileMenu() → *pyqtribbon.menu.RibbonMenu*

Add a new ribbon menu to the application button.

Returns
The new ribbon menu.

class pyqtribbon.titlewidget.**RibbonTitleLabel**

Bases: qtpy.QtWidgets.QLabel

Title label in the ribbon bar.

class pyqtribbon.titlewidget.**RibbonTitleWidget**(*title='PyQtRibbon'*, *parent=None*)
RibbonTitleWidget(parent=None)

Bases: qtpy.QtWidgets.QFrame

The title widget of the ribbon.

helpButtonClicked

collapseRibbonButtonClicked

_quickAccessButtons = []

_rightToolButtons = []

_quickAccessButtonHeight = 30

_rightButtonHeight = 24

applicationButton() → *RibbonApplicationButton*

Return the application button.

setApplicationIcon(*icon: qtpy.QtGui.QIcon*)

Set the application icon.

Parameters
icon – The icon to set.

addTitleWidget(*widget: qtpy.QtWidgets.QWidget*)

Add a widget to the title layout.

Parameters
widget – The widget to add.

insertTitleWidget(*index: int*, *widget: qtpy.QtWidgets.QWidget*)

Insert a widget to the title layout.

Parameters

- **index** – The index to insert the widget.
- **widget** – The widget to insert.

removeTitleWidget(*widget: qtpy.QtWidgets.QWidget*)

Remove a widget from the title layout.

Parameters
widget – The widget to remove.

tabBar() → *pyqtribbon.tabbar.RibbonTabBar*

Return the tab bar of the ribbon.

Returns
The tab bar of the ribbon.

quickAccessToolBar() → qtpy.QtWidgets.QToolBar

Return the quick access toolbar of the ribbon.

Returns
The quick access toolbar of the ribbon.

quickAccessButtons() → List[qtpy.QtWidgets.QToolButton]

Return the quick access buttons of the ribbon.

Returns
The quick access buttons of the ribbon.

addQuickAccessButton(*button: qtpy.QtWidgets.QToolButton*)

Add a widget to the quick access bar.

Parameters
button – The button to add.

setQuickAccessButtonHeight(*height: int = 30*)

Set the height of the quick access buttons.

Parameters
height – The height to set.

title() → str

Return the title of the ribbon.

Returns
The title of the ribbon.

setTitle(*title: str*)

Set the title of the ribbon.

Parameters
title – The title to set.

rightToolBar() → qtpy.QtWidgets.QToolBar

Return the right toolbar of the ribbon.

Returns
The right toolbar of the ribbon.

addRightToolButton(*button: qtpy.QtWidgets.QToolButton*)

Add a widget to the right button bar.

Parameters
button – The button to add.

setRightToolBarHeight(*height: int = 24*)

Set the height of the right buttons.

Parameters
height – The height to set.

helpRibbonButton() → qtpy.QtWidgets.QToolButton

Return the help ribbon button.

Returns
The help ribbon button.

setHelpButtonIcon(*icon: qtpy.QtGui.QIcon*)

Set the icon of the help button.

Parameters
icon – The icon to set.

removeHelpButton()

Remove the help button from the ribbon.

setCollapseButtonIcon(*icon: qtpy.QtGui.QIcon*)

Set the icon of the min button.

Parameters
icon – The icon to set.

removeCollapseButton()

Remove the min button from the ribbon.

collapseRibbonButton() → qtpy.QtWidgets.QToolButton

Return the collapse ribbon button.

Returns
The collapse ribbon button.

pyqtribbon.toolbutton

Module Contents

Classes

RibbonButtonStyle	Button style, Small, Medium, or Large.
RibbonToolButton	Tool button that is showed in the ribbon.

Attributes

Small
Medium
Large

class pyqtribbon.toolbutton.**RibbonButtonStyle**

Bases: enum.IntEnum

Button style, Small, Medium, or Large.

Small = 0

Medium = 1

Large = 2

pyqtribbon.toolbutton.**Small**

pyqtribbon.toolbutton.**Medium**

pyqtribbon.toolbutton.**Large**

class pyqtribbon.toolbutton.**RibbonToolButton**(*parent=None*)

Bases: qtpy.QtWidgets.QToolButton

Tool button that is showed in the ribbon.

_buttonStyle: *RibbonButtonStyle*

_largeButtonIconSize = 64

_mediumButtonIconSize = 48

_smallButtonIconSize = 32

_maximumIconSize = 64

setMaximumIconSize(*size: int*)

Set the maximum icon size of the button.

Parameters
size – The maximum icon size of the button.

maximumIconSize() → int

Get the maximum icon size of the button.

Returns
The maximum icon size of the button.

setButtonStyle(*style:* RibbonButtonStyle)

Set the button style of the button.

Parameters
style – The button style of the button.

buttonStyle() → *RibbonButtonStyle*

Get the button style of the button.

Returns
The button style of the button.

addRibbonMenu() → *pyqtribbon.menu.RibbonMenu*

Add a ribbon menu for the button.

Returns
The added ribbon menu.

pyqtribbon.typehints

Module Contents

Classes

PyQtSignalType	This is a protocol for the pyqt signal type.
PyQtActionType	This is a protocol for the pyqt action type.
RibbonType	This is a protocol for the ribbon type for type hints in categories for getting the tabRect.

class pyqtribbon.typehints.**PyQtSignalType**

This is a protocol for the pyqt signal type.

connect(*slot*)

disconnect(*slot*)

emit(**args*)

class pyqtribbon.typehints.**PyQtActionType**

This is a protocol for the pyqt action type.

triggered: *PyQtSignalType*

class pyqtribbon.typehints.**RibbonType**

This is a protocol for the ribbon type for type hints in categories for getting the tabRect.

tabBar() → qtpy.QtWidgets.QTabBar

showContextCategory(*category*)

hideContextCategory(*category*)

setCategoryState(*category*, *state*)

categoryVisible(*category*)

categories() → Dict

repaint()

pyqtribbon.utils

Module Contents

Functions

data_file_path(filename)	Return the path to a data file.

pyqtribbon.utils.**data_file_path**(*filename*)

Return the path to a data file.

Parameters
filename – The filename of the data file.

Returns
The path to the data file.

4.1.2 Package Contents

Classes

RibbonCategoryStyle	The button style of a category.
RibbonMenu	
RibbonPermanentMenu	A permanent menu.
RibbonSpaceFindMode	Mode to find available space in a grid layout, ColumnWise or RowWise.
RibbonBar	The RibbonBar class is the top level widget that contains the ribbon.
RibbonStyle	Enum where members are also (and must be) ints
RibbonButtonStyle	Button style, Small, Medium, or Large.

Attributes

Normal
Context
ColumnWise
RowWise
Default
Debug
Small
Medium
Large

class pyqtribbon.**RibbonCategoryStyle**

Bases: enum.IntEnum

The button style of a category.

Normal = 0

Context = 1

pyqtribbon.**Normal**

pyqtribbon.**Context**

class pyqtribbon.**RibbonMenu**(*title: str = ''*, *parent=None) RibbonMenu(parent=None*)

Bases: qtpy.QtWidgets.QMenu

addWidget(*widget: qtpy.QtWidgets.QWidget*)

Add a widget to the menu.

Parameters
widget – The widget to add.

addHorizontalLayoutWidget() → qtpy.QtWidgets.QHBoxLayout

Add a horizontal layout widget to the menu.

Returns
The horizontal layout.

addVerticalLayoutWidget() → qtpy.QtWidgets.QVBoxLayout

Add a vertical layout widget to the menu.

Returns
The vertical layout.

addGridLayoutWidget() → qtpy.QtWidgets.QGridLayout

Add a grid layout widget to the menu.

Returns
The grid layout.

addFormLayoutWidget() → qtpy.QtWidgets.QFormLayout

Add a form layout widget to the menu.

Returns
The form layout.

addSpacing(*spacing: int = 5*)

Add spacing to the menu.

Parameters
spacing – The spacing.

addLabel(*text: str = ''*, *alignment: qtpy.QtCore.Qt.Alignment = QtCore.Qt.AlignLeft*)

Add a label to the menu.

Parameters

- **text** – The text of the label.
- **alignment** – The alignment of the label.

class pyqtribbon.**RibbonPermanentMenu**(*title: str = ''*, *parent=None) RibbonPermanentMenu(parent=None*)

Bases: *RibbonMenu*

A permanent menu.

actionAdded

hideEvent(*a0: QHideEvent*) → None

actionEvent(*a0: QActionEvent*) → None

class pyqtribbon.**RibbonSpaceFindMode**

Bases: enum.IntEnum

Mode to find available space in a grid layout, ColumnWise or RowWise.

ColumnWise = 0

RowWise = 1

pyqtribbon.**ColumnWise**

pyqtribbon.**RowWise**

class pyqtribbon.**RibbonBar**(*title: str = ''*, *maxRows=6*, *parent=None) RibbonBar(parent=None*)

Bases: qtpy.QtWidgets.QMenuBar

The RibbonBar class is the top level widget that contains the ribbon.

helpButtonClicked

fileButtonClicked

_categories: Dict[str, *pyqtribbon.category.RibbonCategory*]

_contextCategoryCount = 0

_maxRows = 6

_ribbonVisible = True

_ribbonHeight = 200

_currentTabIndex = 0

abstract actionAt(*QPoint*)

abstract actionGeometry(*QAction*)

abstract activeAction()

abstract addMenu(**__args*)

abstract addAction(**__args*)

abstract addSeparator()

abstract clear()

abstract cornerWidget(*corner=None*)

abstract insertMenu(*QAction*, *QMenu*)

abstract insertSeparator(*QAction*)

abstract isDefaultUp()

abstract isNativeMenuBar()

abstract setActiveAction(*QAction*)

abstract setCornerWidget(*QWidget*, *corner=None*)

abstract setDefaultUp(*up*)

abstract setNativeMenuBar(*bar*)

setRibbonStyle(*style:* RibbonStyle)

Set the style of the ribbon.

Parameters
style – The style to set.

applicationOptionButton() → *pyqtribbon.titlewidget.RibbonApplicationButton*

Return the application button.

setApplicationIcon(*icon: qtpy.QtGui.QIcon*)

Set the application icon.

Parameters
icon – The icon to set.

addTitleWidget(*widget: qtpy.QtWidgets.QWidget*)

Add a widget to the title widget.

Parameters
widget – The widget to add.

removeTitleWidget(*widget: qtpy.QtWidgets.QWidget*)

Remove a widget from the title widget.

Parameters
widget – The widget to remove.

insertTitleWidget(*index: int*, *widget: qtpy.QtWidgets.QWidget*)

Insert a widget to the title widget.

Parameters

- **index** – The index to insert the widget.
- **widget** – The widget to insert.

addFileMenu() → *pyqtribbon.menu.RibbonMenu*

Add a file menu to the ribbon.

ribbonHeight() → int

Get the total height of the ribbon.

Returns
The height of the ribbon.

setRibbonHeight(*height: int*)

Set the total height of the ribbon.

Parameters
height – The height to set.

tabBar() → *pyqtribbon.tabbar.RibbonTabBar*

Return the tab bar of the ribbon.

Returns
The tab bar of the ribbon.

quickAccessToolBar() → qtpy.QtWidgets.QToolBar

Return the quick access toolbar of the ribbon.

Returns
The quick access toolbar of the ribbon.

addQuickAccessButton(*button: qtpy.QtWidgets.QToolButton*)

Add a button to the quick access bar.

Parameters
button – The button to add.

setQuickAccessButtonHeight(*height: int = 30*)

Set the height of the quick access buttons.

Parameters
height – The height to set.

title() → str

Return the title of the ribbon.

Returns
The title of the ribbon.

setTitle(*title: str*)

Set the title of the ribbon.

Parameters
title – The title to set.

rightToolBar() → qtpy.QtWidgets.QToolBar

Return the right toolbar of the ribbon.

Returns
The right toolbar of the ribbon.

addRightToolButton(*button: qtpy.QtWidgets.QToolButton*)

Add a widget to the right button bar.

Parameters
button – The button to add.

setRightToolBarHeight(*height: int = 24*)

Set the height of the right buttons.

Parameters
height – The height to set.

helpRibbonButton() → qtpy.QtWidgets.QToolButton

Return the help button of the ribbon.

Returns
The help button of the ribbon.

setHelpButtonIcon(*icon: qtpy.QtGui.QIcon*)

Set the icon of the help button.

Parameters
icon – The icon to set.

removeHelpButton()

Remove the help button from the ribbon.

collapseRibbonButton() → qtpy.QtWidgets.QToolButton

Return the collapse ribbon button.

Returns
The collapse ribbon button.

setCollapseButtonIcon(*icon: qtpy.QtGui.QIcon*)

Set the icon of the min button.

Parameters
icon – The icon to set.

removeCollapseButton()

Remove the min button from the ribbon.

category(*name: str*) → *pyqtribbon.category.RibbonCategory*

Return the category with the given name.

Parameters
name – The name of the category.

Returns
The category with the given name.

categories() → Dict[str, *pyqtribbon.category.RibbonCategory*]

Return a list of categories of the ribbon.

Returns
A dict of categories of the ribbon.

addCategoriesBy(*data: Dict[str, Dict]*) → Dict[str, *pyqtribbon.category.RibbonCategory*]

Add categories from a dict.

Parameters
data – The dict of categories. The dict is of the form:

```
{
    "category-title": {
        "style": RibbonCategoryStyle.Normal,
        "color": QtCore.Qt.red,
        "panels": {
            "panel-title": {
                "showPanelOptionButton": True,
                "widgets": {
                    "widget-name": {
                        "type": "Button",
                        "arguments": {
                            "key1": "value1",
                            "key2": "value2"
```

(continues on next page)

(continued from previous page)

```
                        }
                    },
                }
            },
        },
    }
}
```

Returns
A dict of categories of the ribbon.

addCategory(*title: str*, *style=RibbonCategoryStyle.Normal*, *color: qtpy.QtGui.QColor = None*) → *pyqtribbon.category.RibbonNormalCategory* | *pyqtribbon.category.RibbonContextCategory*

Add a new category to the ribbon.

Parameters

- **title** – The title of the category.
- **style** – The button style of the category.
- **color** – The color of the context category, only used if style is Context, if None, the default color will be used.

Returns
The newly created category.

addNormalCategory(*title: str*) → *pyqtribbon.category.RibbonNormalCategory*

Add a new category to the ribbon.

Parameters
title – The title of the category.

Returns
The newly created category.

addContextCategory(*title: str*, *color: qtpy.QtGui.QColor | qtpy.QtCore.Qt.GlobalColor = QtCore.Qt.blue*) → *pyqtribbon.category.RibbonContextCategory*

Add a new context category to the ribbon.

Parameters

- **title** – The title of the category.
- **color** – The color of the context category, if None, the default color will be used.

Returns
The newly created category.

addContextCategories(*name: str*, *titles: List[str]*, *color: qtpy.QtGui.QColor | qtpy.QtCore.Qt.GlobalColor = QtCore.Qt.blue*) → *pyqtribbon.category.RibbonContextCategories*

Add a group of context categories with the same tab color to the ribbon.

Parameters

- **name** – The name of the context categories.
- **titles** – The title of the category.
- **color** – The color of the context category, if None, the default color will be used.

Returns
The newly created category.

showCategoryByIndex(*index: int*)

Show category by tab index

Parameters
index – tab index

showContextCategory(*category:* pyqtribbon.category.RibbonContextCategory | pyqtribbon.category.RibbonContextCategories)

Show the given category or categories, if it is not a context category, nothing happens.

Parameters
category – The category to show.

hideContextCategory(*category:* pyqtribbon.category.RibbonContextCategory | pyqtribbon.category.RibbonContextCategories)

Hide the given category or categories, if it is not a context category, nothing happens.

Parameters
category – The category to hide.

categoryVisible(*category:* pyqtribbon.category.RibbonCategory) → bool

Return whether the category is shown.

Parameters
category – The category to check.

Returns
Whether the category is shown.

removeCategory(*category:* pyqtribbon.category.RibbonCategory)

Remove a category from the ribbon.

Parameters
category – The category to remove.

removeCategories(*categories:* pyqtribbon.category.RibbonContextCategories)

Remove a list of categories from the ribbon.

Parameters
categories – The categories to remove.

setCurrentCategory(*category:* pyqtribbon.category.RibbonCategory)

Set the current category.

Parameters
category – The category to set.

currentCategory() → *pyqtribbon.category.RibbonCategory*

Return the current category.

Returns
The current category.

minimumSizeHint() → qtpy.QtCore.QSize

Return the minimum size hint of the widget.

Returns
The minimum size hint.

_collapseButtonClicked()

showRibbon()

Show the ribbon.

hideRibbon()

Hide the ribbon.

ribbonVisible() → bool

Get the visibility of the ribbon.

Returns
True if the ribbon is visible, False otherwise.

setRibbonVisible(*visible: bool*)

Set the visibility of the ribbon.

Parameters
visible – True to show the ribbon, False to hide it.

class pyqtribbon.RibbonStyle

Bases: enum.IntEnum

Enum where members are also (and must be) ints

Default = 0

Debug = 1

pyqtribbon.Default

pyqtribbon.Debug

class pyqtribbon.RibbonButtonStyle

Bases: enum.IntEnum

Button style, Small, Medium, or Large.

Small = 0

Medium = 1

Large = 2

pyqtribbon.Small

pyqtribbon.Medium

pyqtribbon.Large

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

Symbols

A

B

C

D

E

F

H

I

L

M

N

P

Q

R

S

T

W