Pyside QMenuView Documentation

Release 0.1.4

David Zuber

August 11, 2015

Contents

PySide view for menues

CHAPTER 1

Features

- MenuView class which creates submenus/actions based on a model.
- Supports modelReset, rowsInserted, rowsRemoved and dataChanged signals of the model.
- Supports icon, text, iconText, toolTip, checked, whatsThis, statusTip, enabled
- Easy to extend and customize.

1.1 Welcome to Pyside QMenuView‘s documentation!

CHAPTER 2

Contents:

2.1 Installation

At the command line either via easy_install or pip:

```
$ easy_install qmenuview
$ pip install qmenuview
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv qmenuview
$ pip install qmenuview
```

2.2 Usage

To use QMenuView:

```
from PySide import QtGui
import qmenuview

app = QtGui.QApplication([])

view = qmenuview.MenuView('RootMenuTitle')

# create a model
m = QtGui.QStandardItemModel()
for i in range(10):
    m.appendRow(QtGui.QStandardItem("Delicious dish no. %s" % i))

view.model = m  # now the submenus are created

# insert new menus
m.appendRow(QtGui.QStandardItem("New fancy dish"))
# the view will also handle trees
rootitem = QtGui.QStandardItem("root")
lvl1item = QtGui.QStandardItem("Level 1")
lvl2item = QtGui.QStandardItem("Level 2")
rootitem.appendRow(lvl1item)
lvl1item.appendRow(lvl2item)
m.appendRow(rootitem)
```

```
# remove menus
m.removeRow(0)

# Menus are automatically updated
rootitem.setText("Newroot")

view.show()

app.exec_()
```

2.2.1 Signals

To handle signals connect to the views signals:

```
from PySide import QtGui
import qmenuview

app = QtGui.QApplication([])

view = qmenuview.MenuView('RootMenuTitle')

# create a model
m = QtGui.QStandardItemModel()
for i in range(10):
    m.appendRow(QtGui.QStandardItem("Delicious dish no. %s" % i))

view.model = m  # now the submenus are created

def triggered_cb(index, checked=False):
     item = index.model().itemForIndex(index)
     print('Item triggered', item)
menuview.action_triggered.connect(triggered_cb)
```

See `qmenuview.MenuView.action_triggered`, `qmenuview.MenuView.action_toggled`, `qmenuview.MenuView.action_hovered`.

2.2.2 Customization

There are multiple things you can customize.

Columns

You can specify which columns of the model you want to use for the data:

```
import qmenuview
view = qmenuview.MenuView()

view.text_column = 2  # use third column for action text
view.icon_column = 1  # use second column for icon
view.icontext_column = -1  # don't set the icontext
```

Advanced Data Control

If you want to have more control over how the data is applied to actions, have a look at the `qmenuview.MenuView.setdataargs`. It is a list of *qmenuview.view.SetDataArgs*. Each entry defines a column and a role for the data to query, a method to convert the data and a name of a action method to apply the data.

So you can remove or add entries to the list. The following example will make the view query data from column 0 with `PySide.QtCore.Qt.FontRole`. There is no need to convert the data, so the convertion function is None. `setFont` will specify that `PySide.QtGui.QAction.setFont()` will be used to apply the data:

```
from PySide import QtCore
import qmenuview
view = qmenuview.MenuView()
fontargs = qmenuview.SetDataArgs('setFont', 0, QtCore.Qt.FontRole, None)
view.setdataargs.append(fontargs)
```

Custom classes

If you want to use custom menu or action classes subclass the view and override `qmenuview.MenuView.create_menu()` or `qmenuview.MenuView.create_action()`:

```
from PySide import QtGui
import qmenuview

class SuperAction(QtGui.QAction): pass

class SuperMenuView(qmenuview.MenuView):
    def create_action(self, parent):
        return SuperAction(parent)
```

2.3 Reference

Automatic generated Documenation by apidoc and autodoc.

2.3.1 qmenuview

Submodules

qmenuview.view

MenuView([title, parent])	A view that creates submenus based on a model.
SetDataArgs(setfunc, column, role, convertfunc)	A container of arguments for setting attributes on an action.

Classes

qmenuview.view.MenuView

class `qmenuview.view.`**`MenuView`**(*title=''*, *parent=None*)

Bases: `Mock`

A view that creates submenus based on a model.

The model can be a list, table or treemodel. Each row equals to one submenu/action. In a treemodel, the leaves are plain actions, the rest also have menus on top.

The view listens to the following signals:

- PySide.QtCore.QAbstractItemModel.modelReset
- PySide.QtCore.QAbstractItemModel.rowsInserted
- PySide.QtCore.QAbstractItemModel.rowsAboutToBeRemoved
- PySide.QtCore.QAbstractItemModel.dataChanged

So the view is quite dynamic. If all child rows of an index are removed, the menu gets removed from the action. If rows are inserted to a parent index, which had no children, the action will get a menu.

If an action emits a signal, the view will emit a signal too. The signal will contain the index and any arguments of the action's signal. You can get the action by using *MenuView.get_action()*. See *MenuView.action_triggered*, *MenuView.action_hovered*, *MenuView.action_toggled*.

Note: At the moment PySide.QtGui.QAction.changed will not be handled. There currently is a bug in MenuView._get_parents(), which causes an infinite loop.

You can set which column to use for each attribute. See *MenuView.text_column*, *MenuView.icon_column*, *MenuView.icontext_column*, *MenuView.tooltip_column*, *MenuView.checked_column*, *MenuView.whatsthis_column*, *MenuView.statustip_column*.

For more control on how the data gets applied to the action, change *MenuView.setdataargs*. It is a list of *SetDataArgs* containers. One container defines the functionname to use for setting the attribute, the column to use, the PySide.QtCore.Qt.ItemDataRole, and a data conversion function.

If you want custom menu and action classes, override *MenuView.create_menu()*, *MenuView.create_action()*.

__init__(*title=''*, *parent=None*)
Initialize a new menu view with the given title

Parameters

- **title** (str) – title of the top menu
- **parent** (PySide.QtGui.QWidget) – the parent widget

Raises None

action_hovered
Signal for when an action gets hovered

action_triggered
Signal for when an action gets triggered

action_toggled
Signal for when an action gets toggled

text_column = None
The column for the action text. Default 0

icon_column = None
The column for the action icon. Default 0

icontext_column = None
The column for the action icon text. Default -1

tooltip_column = None
The column for the tooltip data. Default 0

checked_column = None
The column for the checked data. Has to be checkable. Default 0

whatsthis_column = None
The column for the whatsThis text. Default 0

statustip_column = None
The column for the statustip text. Default 0

setdataargs = None
A list of *SetDataArgs* containers. Defines how the data from the model is applied to the action

model
Get the model

Returns the current model

Return type PySide.QtCore.QAbstractItemModel

Raises None

reset()
Delete and recreate all menus

Returns None

Return type None

Raises None

create_all_menus()
Create all menus according to the model

Returns None

Return type None

Raises None

create_menu_for_index(*index*)

create_menu(*parent*)
Create a menu and return the menus action.

The parent of the menu has to be set to `parent`

Parameters **parent** (PySide.QtGui.QMenu) – The parent menu

Returns The menu action

Return type PySide.QtGui.QAction

Raises None

create_action(*parent*)
Create and return a new action

The parent of the action has to be set to `parent`

Parameters **parent** (PySide.QtGui.QMenu) – The parent menu

Returns The created action

Return type PySide.QtGui.QAction

Raises None

insert_menus(*parent*, *first*, *last*)
Create menus for rows first til last under the given parent

Parameters

- **parent** (PySide.QtCore.QModelIndex) – The parent index
- **first** (int) – the first row
- **last** (int) – the last row

Returns None

Return type None

Raises None

remove_menus(*parent*, *first*, *last*)
Remove the menus under the given parent

Parameters

- **parent** (PySide.QtCore.QModelIndex) – the parent of the menus
- **first** (int) – the first row
- **last** (int) – the last row

Returns None

Return type None

Raises None

update_menus(*topLeft*, *bottomRight*)
Update the menus from topleft index to bottomright index

Parameters

- **topLeft** (PySide.QtCore.QModelIndex) – The top left index to update
- **bottomRight** (PySide.QtCore.QModelIndex) – the bottom right index to update

Returns None

Return type None

Raises None

get_index(*action*, *column=0*)
Return the index for the given action

Parameters

- **action** (PySide.QtGui.QAction) – the action to query
- **column** (int) – The column of the index

Returns the index of the action

Return type PySide.QtCore.QModelIndex

Rasies None

get_action(*index*)
Return the action for the given index

Parameters **index** (PySide.QtCore.QModelIndex) – the index to query

Returns the action for the given index

Return type PySide.QtGui.QAction

Raises None

set_action_data(*action*, *index*)
Set the data of the action for the given index

Note: The column of the index does not matter. The columns for the data are specified in *MenuView.setdataargs*.

The arguments to used are defined in *MenuView.setdataargs*.

Parameters

- **action** (PySide.QtGui.QAction) – The action to update
- **index** (PySide.QtCore.QModelIndex) – The index with the data

Returns None

Return type None

Raises None

static get_data(*index*, *role*, *column=None*)
Get data of the given index

If the column is is not None and different from the index column, will get the data from a sibling index with the same row.

Parameters

- **index** (PySide.QtCore.QModelIndex) – The index to query for data
- **role** (PySide.QtCore.Qt.ItemDataRole) – the data role
- **column** (int | None) – the column of the row to query for data.

Returns The data retrieved

Raises None

qmenuview.view.SetDataArgs

class qmenuview.view.**SetDataArgs**(*setfunc*, *column*, *role*, *convertfunc*)
Bases: object

A container of arguments for setting attributes on an action.

The data is queried from the model with `role`. Then converted with `convertfunc`. Then `setfunc` is used for setting the attribute on the action. `convertfunc` can be `None`.

If column is a string, the attribute of the view with that name will be used as column.

__init__(*setfunc*, *column*, *role*, *convertfunc*)
Initialize a new container

Raises None

Data
qmenuview.view.**QtCore** = **<Mock id='140698019332880'>**
qmenuview.view.**QtGui** = **<Mock id='140698019333072'>**

Module contents

Data

---

qmenuview.**absolute_import** = **_Feature((2, 5, 0, 'alpha', 1), (3, 0, 0, 'alpha', 0), 16384)**

2.4 Contributing

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

You can contribute in many ways:

2.4.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/storax/qmenuview/issues.

If you are reporting a bug, please include:

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

Fix Bugs

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

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

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

Submit Feedback

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

If you are proposing a feature:

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

2.4.2 Get Started!

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

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

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

3. Create a branch for local development:

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

Now you can make your changes locally.

4. When you're done making changes, check that your changes pass style and unit tests, including testing other Python versions with tox:

```
$ tox
```

To get tox, just pip install it.

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

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

6. Submit a pull request through the GitHub website.

2.4.3 Pull Request Guidelines

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

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, and 3.3, and for PyPy. Check https://travis-ci.org/storax/qmenuview under pull requests for active pull requests or run the `tox` command and make sure that the tests pass for all supported Python versions.

2.4.4 Tips

To run a subset of tests:

```
$ py.test test/test_qmenuview.py
```

2.5 Credits

2.5.1 Development Lead

- David Zuber <zuber.david@gmx.de>

2.5.2 Contributors

None yet. Why not be the first?

2.6 History

2.6.1 0.1.0 (2015-08-08)

- First release on PyPI.

2.6.2 0.1.1 (2015-08-11)

- Fix getting parents of action
- Fix changing columns

2.6.3 0.1.4 (2015-08-11)

- Fix removing all rows below root.

CHAPTER 3

Feedback

If you have any suggestions or questions about **Pyside QMenuView** feel free to email me at zuber.david@gmx.de.

If you encounter any errors or problems with **Pyside QMenuView**, please let me know! Open an Issue at the GitHub https://github.com/storax/qmenuview main repository.

3.1 Indices and tables

- genindex
- modindex
- search

q

Symbols

A

C

G

I

M

Q

R

S

T

U

W