PyQt : concevoir visuellement des interfaces avec Qt Designer

Par Harsh - Traducteur : Thibaut Cuvelier

Date de publication : 2 juin 2011

Dernière mise à jour : 12 juillet 2011

Concevoir une interface graphique pour une application peut être une tâche éreintante. Il y a quelques lignes directrices dont il faut tenir compte, des dispositions à maintenir et bien d'autres choses. Dans les exemples PyQt que l'on a vus jusqu'à présent, on a écrit ces interfaces directement en code. C'est facile et amusant à faire quand il n'y a que cinq à dix widgets, cela n'en vaut pas la peine pour des interfaces d'applications plus complètes.

Commentez

I - Qt Designer....3
II - Une visionneuse d'images....3
II-A - Concevoir la GUI....3
II-B - Utiliser pyuic4....5
II-C - Lancer la GUI....5
II-D - Ajouter la logique de l'application....7
III - Exercice....8
IV - Remerciements....8

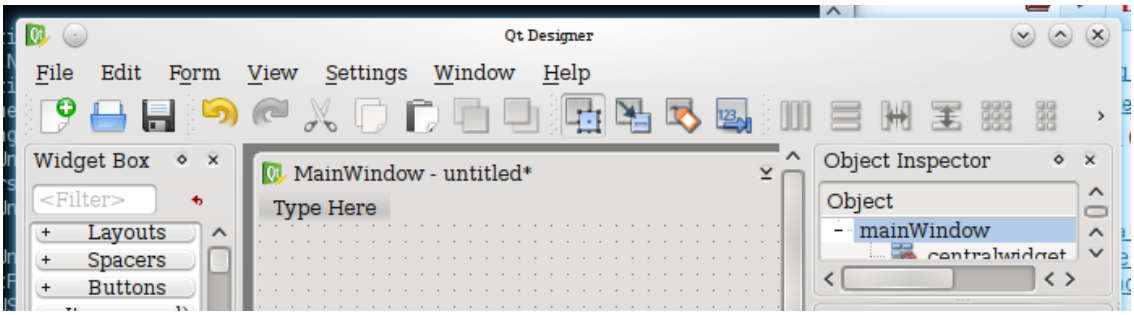
I - Qt Designer

Heureusement, Qt fournit un outil pour concevoir des interfaces et en faire du code utilisable automatiquement. Cet outil s'appelle Qt Designer et il est installé en même temps que le bundle Qt. En plus, on a besoin d'un convertisseur pour le XML des fichiers `.ui` du designer en fichiers Python, il a été installé avec PyQt4.

Ainsi, pour concevoir des interfaces pour PyQt, les outils suivants sont nécessaires :

- Qt Designer, de Nokia ;
- `pyuic4`, de Riverbank.

L'interface pourrait sembler familière à ceux qui ont déjà utilisé Visual Studio, Glade ou d'autres outils du même genre. Pour le reste, il est assez intuitif pour être appris sans difficulté. Le **guide de Qt Designer** donne une aide plus élaborée au besoin.



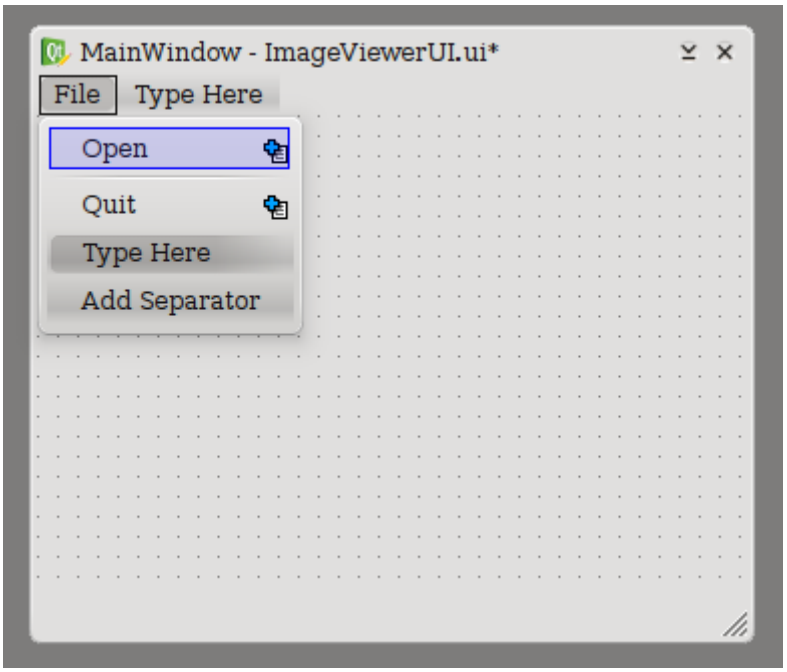
Références : **Qt Designer**, **la documentation de PyQt et de Qt Designer**.

II - Une visionneuse d'images

Cette application sera très simple, avec une option pour ouvrir une image d'un format répandu (JPG, PNG, GIF...) et une autre pour fermer. À l'ouverture d'une image, on affiche aussi ses dimensions dans la barre de statut.

II-A - Concevoir la GUI

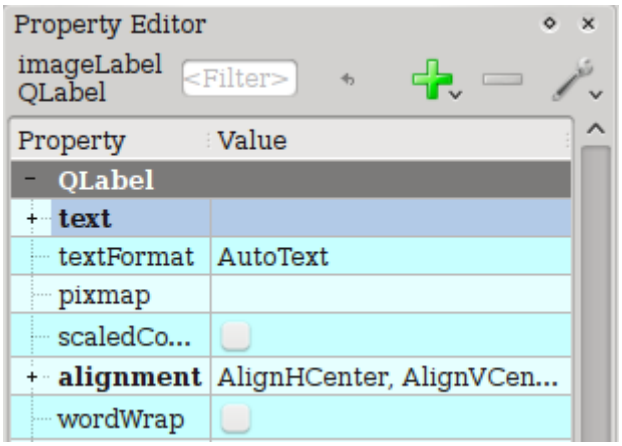
Dans Qt Designer (la commande `designer` sous Linux), choisissez *Fichier* > *Nouveau* > *Fenêtre principale* sous *templates*. Ceci crée un nouveau widget QMainWindows. Il s'agit simplement du rassemblement d'un widget pour la zone principale, d'une barre de menu et d'une barre d'état en bas, soit la base commune d'une application.



Depuis la boîte des widgets, déposez un label sur la zone principale. Un clic droit dans l'espace vide de la même zone ouvrira un menu avec un sous-menu *Disposer* et l'option *Disposer horizontalement*. Le designer va alors automatiquement élargir le label sur toute la zone disponible. On doit utiliser une disposition (peu importe laquelle, ici), puisqu'on veut que le label s'agrandisse pour héberger toute l'image.

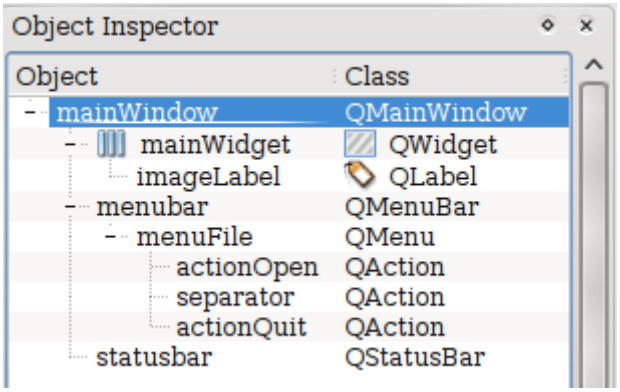
Maintenant, les menus. Ajoutez un menu *Fichier* et des options *Ouvrir* et *Fermer*.

Dans l'éditeur de propriétés, sélectionnez et définissez la propriété *text* du label à une chaîne vide, son alignement à *AlignHCenter* et *AlignVCenter*. Ceci termine la conception de la GUI dans le designer. Il ne faut pas le fermer pour autant, on va explorer d'autres propriétés des widgets utilisés et voir leurs utilisations possibles.



Par exemple, regardez ce que font les propriétés *geometry*, *font*, *tooltip*... Si vous êtes de bonne humeur, regardez aussi la documentation des **feuilles de style**, elle explique beaucoup de choses sur le construction de chaque widget et comment le personnaliser à l'envi.

Finalement, renommer les variables des éléments pourrait être une bonne idée, puisque cela va aider à coder l'application de manière plus lisible. Voici comment j'ai décidé de les nommer, mais vous pouvez choisir une autre convention de nommage :



Il suffit de double-cliquer sur le nom de l'objet et de les éditer en ligne. Une fois cela fait, enregistrez le fichier (ImageViewerUI.ui).

Références : **QLabel**, **QAction**, **QMenuBar**, **Qt Designer et dispositions**.

II-B - Utiliser pyuic4

La prochaine étape est d'utiliser l'outil déjà mentionné, pyuic4. On l'appelle ainsi :

```
pyuic4 entree.ui -o sortie.py
```

On peut lui passer un paramètre -x pour rendre le code généré exécutable.

Ainsi, pour le fichier juste créé :

```
pyuic4 ImageViewerUI.ui -o ImageViewerUI.py
```

Ceci va créer un fichier ImageViewerUI.py que l'on pourra utiliser. En regardant son code, on s'aperçoit que ce n'est qu'une longue liste de construction de widgets et d'application de propriétés. Il n'y a aucun besoin d'éditer ce fichier, il est même recommandé de ne pas le faire, puisque pyuic4 écrasera tous les changements si on le relance.

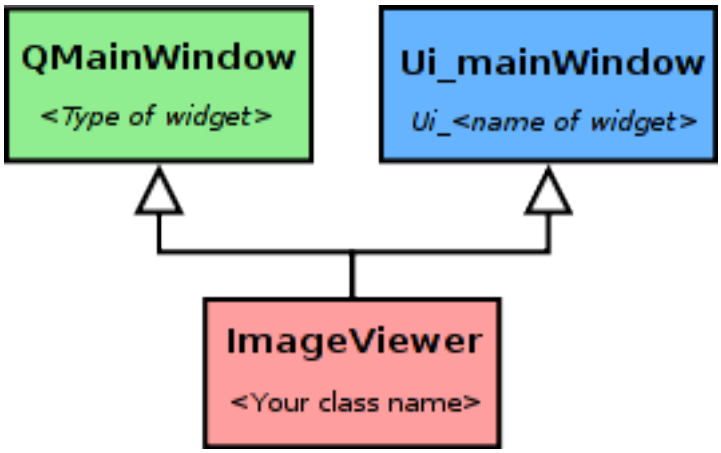
Lors de l'édition de fichiers .ui, il faut s'assurer que l'outil pyuic4 est appelé, afin de le convertir en son équivalent Python (ou de mettre à jour ce fichier pour qu'il reflète les derniers changements).

Pour plus d'informations sur les options que l'on peut passer à pyuic4,

```
pyuic4 &#150;help
```

II-C - Lancer la GUI

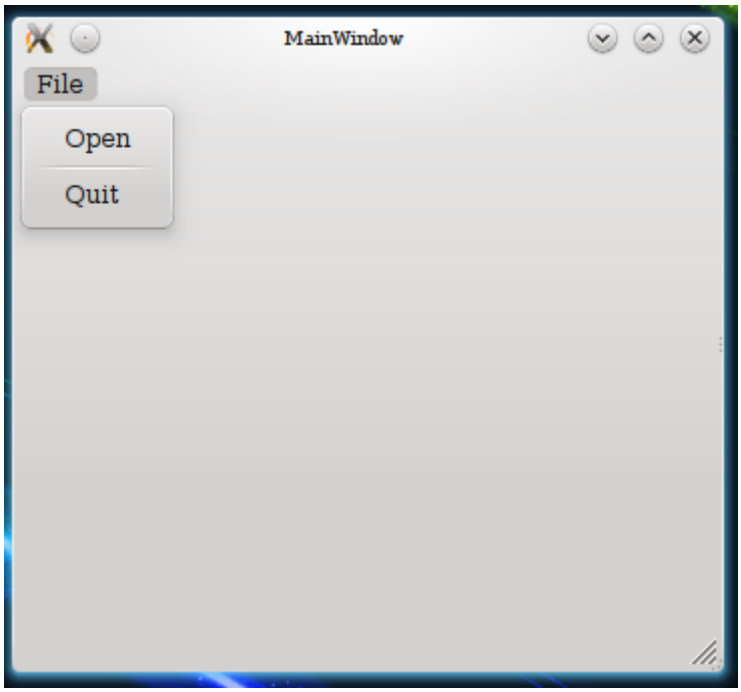
Il faut créer un fichier ImageViewer.py pour finalement ajouter la logique de l'application. Avant que l'on code cette partie, il faut expliquer l'approche de Qt pour les classes dérivées. Un fichier généré par Qt Designer hérite de la classe QMainWindow, l'interface est configurée avec la méthode setupUi(). Ceci crée tous les objets et widgets pour l'interface en tant qu'attributs du dérivé de QMainWindow, cette classe est donc prête pour affichage. Ensuite, on doit ajouter la logique de l'application à cette classe en tant que méthodes normales. Puisqu'on a accès à tous les widgets utilisés dans l'interface par le biais de la classe, on peut faire comme désiré avec les fonctionnalités disponibles. Le diagramme suivant explique la hiérarchie que l'on doit suivre à chaque fois que l'on implémente une classe d'interface :



Ainsi, une classe de base pourrait ressembler à ceci :

```
#!/usr/bin/python

from PyQt4 import QtGui, QtCore
import sys

import ImageViewerUI

class ImageViewer(QtGui.QMainWindow, ImageViewerUI.Ui_mainWindow):
    def __init__(self, parent=None):
        super(ImageViewer, self).__init__(parent)
        self.setupUi(self)

    def main(self):
        self.show()

if __name__=='__main__':
    app = QtGui.QApplication(sys.argv)
    imageViewer = ImageViewer()
    imageViewer.main()
    app.exec_()
```

En lançant ce code, on obtient la fenêtre que l'on vient de créer :



Il est impératif d'appeler self.setupUi(self) pour que l'interface s'initialise.

Références : **QMainWindow**.

II-D - Ajouter la logique de l'application

Maintenant, il faut ajouter la fonctionnalité d'ouverture d'image, on définit donc quelques méthodes dans la classe :

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

from PyQt4 import QtGui, QtCore
import sys

import ImageViewerUI

class ImageViewer(QtGui.QMainWindow, ImageViewerUI.Ui_mainWindow):
    def __init__(self, parent=None):
        super(ImageViewer, self).__init__(parent)
        self.setupUi(self)
        self.connectActions()

    def connectActions(self):
        self.actionQuit.triggered.connect(QtGui.qApp.quit)
        self.actionOpen.triggered.connect(self.openImage)

    def openImage(self):
        fileName = QtGui.QFileDialog.getOpenFileName(
                        self,
                        "Ouvrir un fichier d'image",
                        QtCore.QDir.homePath(),
                        "Fichiers d'image (*.jpg, *.jpeg, *.gif, *.png)"
                    )
        if fileName:
            self.imageLabel.setPixmap(QtGui.QPixmap(fileName))

    def main(self):
        self.show()

if __name__=='__main__':
    app = QtGui.QApplication(sys.argv)
    imageViewer = ImageViewer()
    imageViewer.main()
    app.exec_()
```

Notez les deux nouvelles méthodes : connectActions() et openImage(). Elles complètent la logique de l'application. Essayez de mieux les comprendre en regardant les classes qu'elles utilisent et en vous référant à leur documentation pour de plus amples informations.

Maintenant, on peut charger des images par *Fichier* > *Ouvrir* et fermer la fenêtre avec *Fichier* > *Fermer*.

Voici le résultat final de tout ce travail :

Références : **QPixmap**, **QFileDialog**, **QApplication**.

III - Exercice

Essayez de construire une application simple d'évaluation de formules mathématiques sous forme textuelle qui contient deux choses :

- un champ d'entrée des expressions mathématiques ;
- un label de sortie pour en afficher le résultat.

Les classes potentiellement à utiliser sont **QLabel**, **QLineEdit** et **QTextEdit**. Un autre indice ? La fonction Python **eval()**. En tant que bonus, tentez d'effectuer l'évaluation en exécution Python restreinte.

IV - Remerciements

Merci à Harsh pour l'autorisation de traduire son article, **PyQt - Creating interfaces visually with Designer** !

Merci à **Jean-Philippe André** pour sa relecture orthographique !