

Modéliser ses fenêtres avec Qt Designer

A force d'écrire le code de vos fenêtres, vous devez peut-être commencer à trouver ça long et répétitif. C'est amusant au début, mais au bout d'un moment on en a un peu marre d'écrire des constructeurs de 3 kilomètres de long juste pour placer les widgets sur la fenêtre.

C'est là que Qt Designer vient vous sauver la vie. Il s'agit d'un programme livré avec Qt (vous l'avez donc déjà installé) qui permet de **dessiner vos fenêtres visuellement**. Mais plus encore, Qt Designer vous permet aussi de modifier les propriétés des widgets, d'utiliser des layouts, et d'effectuer la connexion entre signaux et slots.

Qt Designer n'est pas un programme magique qui va réfléchir à votre place. Il vous permet juste de gagner du temps et d'éviter les tâches répétitives d'écriture du code de génération de la fenêtre.

N'utilisez PAS Qt Designer et ne lisez PAS ce chapitre si vous ne savez pas coder vos fenêtres à la main. En clair, si vous avez voulu sauter les chapitres précédents et juste lire celui-ci parce que vous le trouvez attirant, vous allez vous planter. C'est dit. 🤪

Nous commencerons par apprendre à manipuler Qt Designer lui-même. Vous verrez que c'est un outil complexe mais qu'on s'y fait vite car il est assez intuitif.

Ensuite, nous apprendrons à utiliser les fenêtres générées avec Qt Designer dans notre code source. Comme vous le verrez, il y a plusieurs façons de faire en fonction de vos besoins.

C'est parti ! 😊

Sommaire du chapitre :



- [Présentation de Qt Designer](#)
- [Placer des widgets sur la fenêtre](#)
- [Configurer les signaux et les slots](#)
- [Utiliser la fenêtre dans votre application](#)

1. Présentation de Qt Designer

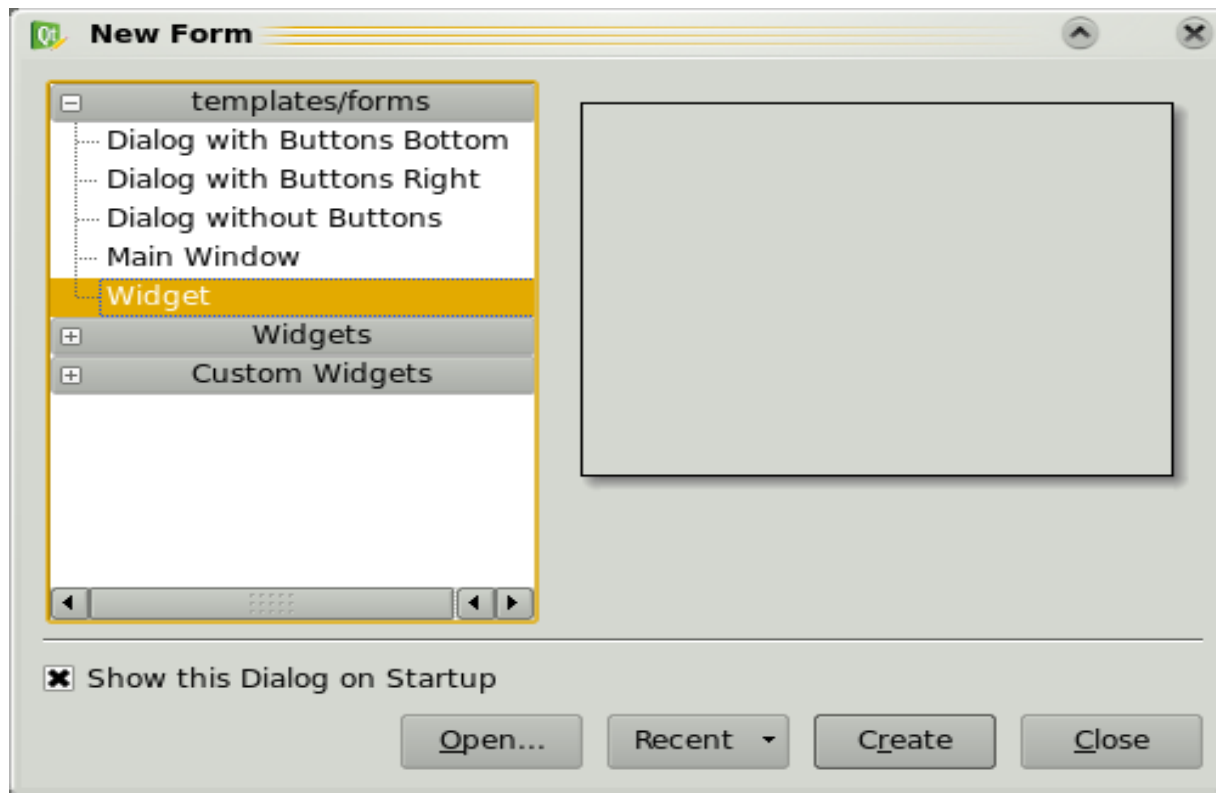


Nous allons commencer par démarrer directement Qt Designer.
Normalement, un raccourci a déjà été créé sur votre système (cf icône ci-contre).

Attention : si vous utilisez un thème personnalisé sur votre ordinateur (par exemple un thème Windows XP téléchargé sur internet), il se pourrait que Qt Designer rencontre des bugs d'affichage. Essayez de désactiver le thème personnalisé et de revenir au thème par défaut avant d'exécuter Qt Designer.

1.1. Choix du type de fenêtre à créer

Lorsque vous lancez Qt Designer, il vous propose de créer un nouveau projet. Vous avez le choix entre plusieurs types de fenêtres :



Les 3 premiers choix correspondent à des `QDialog`.

Vous pouvez aussi créer une `QMainWindow` si vous avez besoin de gérer des menus et des barres d'outils.

Enfin, le dernier choix correspond à une simple fenêtre de type `QWidget`.

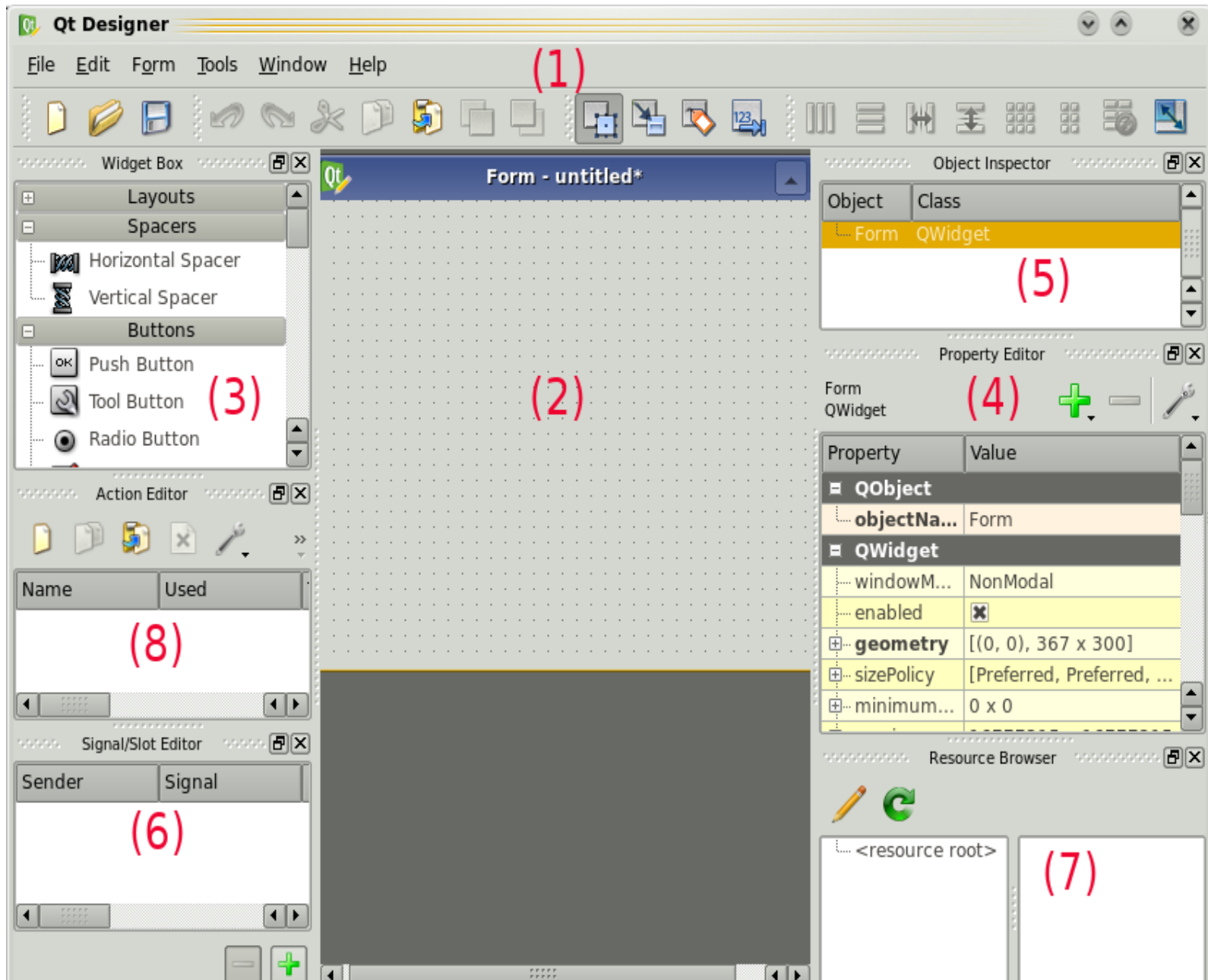
Pour nos exemples, nous allons choisir de créer une fenêtre simple de type `QWidget`. Sélectionnez donc le choix `Widget`.

Il y a d'autres choix que je ne détaillerai pas ici, dans les sous-catégories "Widgets" et "Custom Widgets". Par exemple, on peut créer une fenêtre-`QGroupBox`.

Vous utiliserez très rarement ces choix.

1.2. Analyse de la fenêtre de Qt Designer

Lorsque vous avez créé un nouveau projet, la fenêtre de Qt Designer commence à s'animer et... comme vous pouvez le voir, c'est assez complet :



Wow ! Mais comment je vais faire pour m'y retrouver avec tous ces boutons ? 🤔

En y allant méthodiquement. 😊

Détaillons chacune des zones importantes dans l'ordre :

1. Sur la **barre d'outils** de Qt Designer, au moins 4 boutons méritent votre attention. Ce sont les 4 boutons situés sous la marque "(1)" rouge que j'ai placée sur la capture d'écran.



Ils permettent de passer d'un mode d'édition à un autre. Qt Designer propose 4 modes d'édition :

- **Edit Widgets** : le mode par défaut, que vous utiliserez le plus souvent. Il permet d'insérer des widgets sur la fenêtre et de modifier leurs propriétés.
- **Edit Signals/Slots** : permet de créer des connexions entre les signaux et les slots de vos widgets.
- **Edit Buddies** : permet d'associer des `QLabel` avec leurs champs respectifs. Lorsque vous faites un layout de type `QFormLayout`, ces associations sont automatiquement créées.
- **Edit Tab Order** : permet de modifier l'ordre de tabulation entre les champs de la fenêtre, pour ceux qui naviguent au clavier et passent d'un champ à l'autre en appuyant sur la touche "Tab".

Nous ne verrons dans ce chapitre que les 2 premiers modes (Edit Widgets et Edit Signals/Slots). Les autres modes sont peu importants et je vous laisse les découvrir par vous-mêmes.

2. Au **centre** de Qt Designer, vous avez la fenêtre que vous êtes en train de dessiner. Pour le moment celle-ci est vide. Si vous créez une `QMainWindow`, vous aurez en plus une barre de menus et une barre d'outils. Leur édition se fait à la souris, c'est très intuitif. Si vous créez une `QDialog`, vous aurez probablement des boutons "OK" et "Annuler" déjà disposés.
3. **Widget Box** : ce dock vous donne la possibilité de sélectionner un widget à placer sur la fenêtre. Vous pouvez constater qu'il y a un assez large choix ! Heureusement, ceux-ci sont organisés par groupes pour y voir plus clair. Pour placer un de ces widgets sur la fenêtre, il suffit de faire un glisser-déplacer. Simple et intuitif.

Les widgets en bas de la liste sont soit d'anciens widgets, soit des widgets modifiés non standards. Vous ne devriez pas avoir besoin d'y toucher.

4. **Property Editor** : lorsqu'un widget est sélectionné sur la fenêtre principale, vous pouvez éditer ses propriétés. Vous noterez que les widgets possèdent en général beaucoup de propriétés, et que celles-ci sont organisées en fonction de la classe dans laquelle elles ont été définies. On peut ainsi modifier toutes les propriétés dont un widget hérite, en plus des propriétés qui lui sont propres.

Comme toutes les classes héritent de `QObject`, vous aurez toujours la propriété `objectName`. C'est le nom de l'objet qui sera créé. N'hésitez pas à le personnaliser, afin d'y voir plus clair tout à l'heure dans votre code source (sinon vous aurez par exemple des boutons appelés `pushButton`, `pushButton_2`, `pushButton_3`, ce qui n'est pas très clair).

Si aucun widget n'est sélectionné, ce sont les propriétés de la fenêtre que vous éditez. Vous pourrez donc par exemple modifier son titre avec la propriété `windowTitle`, son icône avec `windowIcon`, etc.

5. **Object Inspector** : affiche la liste des widgets placés sur la fenêtre, en fonction de leur relation de parenté, sous forme d'arbre. Ça peut être pratique si vous avez une fenêtre complexe et que vous commencez à vous perdre dedans. Vous pouvez ainsi y voir par exemple que votre fenêtre contient un `QGroupBox` qui contient 3 cases à cocher.
6. **Signal / slot editor** : si vous avez associé des signaux et des slots, les connexions du widget sélectionné apparaissent ici. Nous verrons comment réaliser des connexions dans `Qt Designer` tout à l'heure.
7. **Resource Browser** : un petit utilitaire qui vous permet de naviguer à travers les fichiers de ressources de votre application. Ici, les fichiers de ressources portent l'extension `.qrc` et ont l'avantage d'être compatibles avec tous les OS.

Les fichiers de ressources servent empaqueter des fichiers (images, sons, texte...) au sein même de votre exécutable. Cela permet d'éviter d'avoir à placer ces fichiers dans le même dossier que votre programme, et cela évite donc le risque de les perdre (puisqu'ils se trouveront *toujours* dans votre exécutable).

C'est un peu hors-sujet, donc je n'en parlerai pas plus ici. Consultez la doc à propos des ressources si vous voulez en savoir plus.

8. **Action Editor** : permet de créer des `QObject`. C'est donc utile lorsque vous créez une `QMainWindow` avec des menus et une barre d'outils.

Voilà qui devrait suffire pour une présentation générale de `Qt Designer`. Maintenant, pratiquons un peu. 😊

2. Placer des widgets sur la fenêtre

Placer des widgets sur la fenêtre est en fait très simple : vous prenez le widget que vous voulez dans la liste à gauche, et vous le faites glisser où vous voulez sur la fenêtre.

Ce qui est très important à savoir, c'est qu'on peut placer ses widgets de 2 manières différentes :

- **De manière absolue** : vos widgets seront disposés au pixel près sur la fenêtre. C'est la méthode par défaut, la plus précise, mais la moins flexible aussi. Je vous avais parlé de ses défauts dans le chapitre sur les layouts.
- **Avec des layouts** (recommandé pour les fenêtres complexes) : vous pouvez utiliser tous les layouts que vous connaissez. Verticaux, horizontaux, en grille, en formulaire... Grâce à cette technique, les widgets s'adapteront automatiquement à la taille de votre fenêtre.

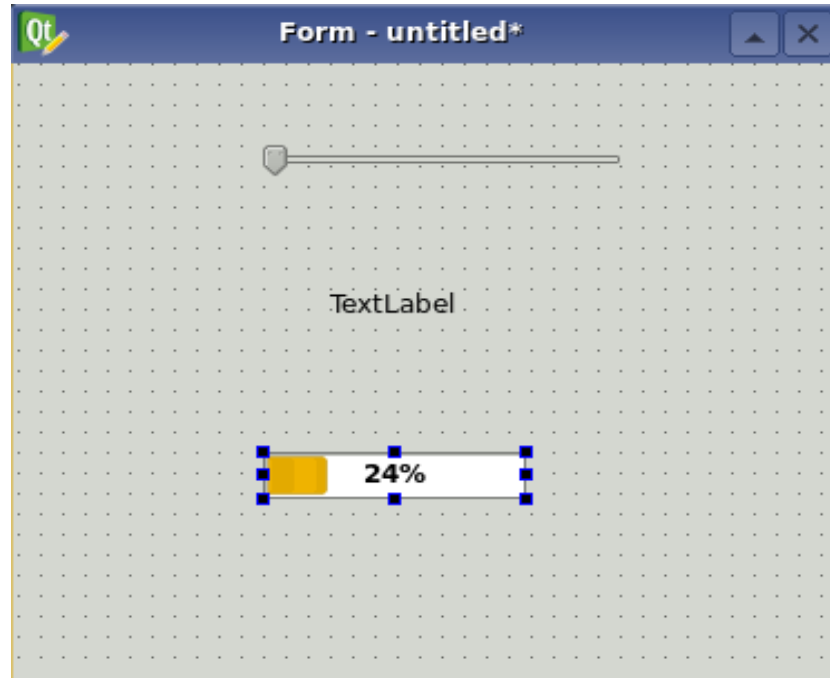
Commençons par les placer de manière absolue, puis nous verrons comment utiliser les layouts dans Qt Designer.

2.1. Placer les widgets de manière absolue

Je vous propose pour vous entraîner de faire une petite fenêtre simple composée de 3 widgets :

- `QSlider`
- `QLabel`
- `QProgressBar`

Votre fenêtre devrait à peu près ressembler à ceci maintenant :



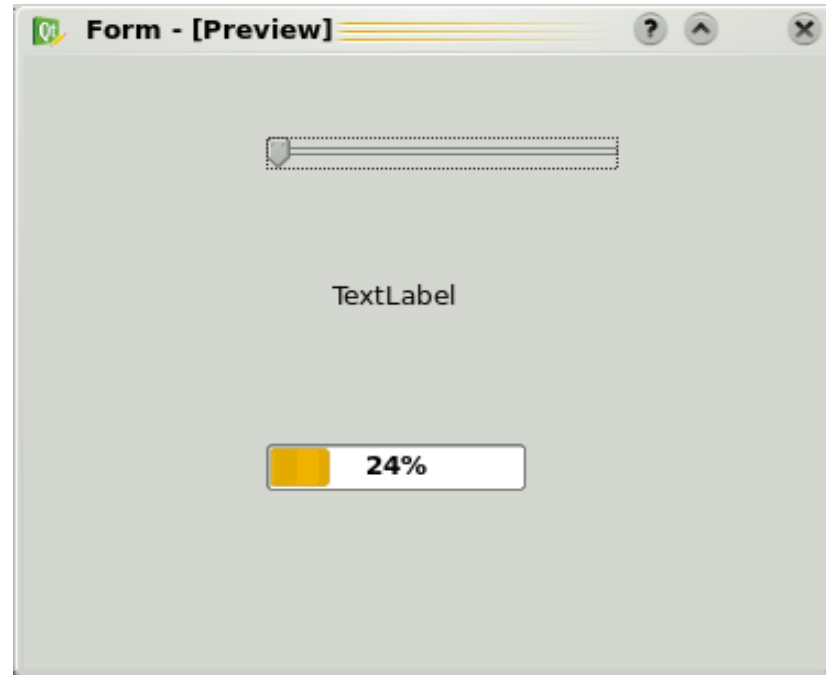
Vous pouvez déplacer ces widgets comme bon vous semble sur la fenêtre.
Vous pouvez les agrandir ou les rétrécir.

Quelques raccourcis à connaître :

- En maintenant la touche **Shift** appuyée, vous pouvez sélectionner plusieurs widgets en même temps.
- Faites **Suppr** pour supprimer les widgets sélectionnés.
- Si vous maintenez la touche **Ctrl** enfoncée lorsque vous déplacez un widget, celui-ci sera copié.
- Vous pouvez **double-cliquer** sur un widget pour modifier son nom (il vaut mieux donner un nom personnalisé plutôt que laisser le nom par défaut).
Sur certains widgets complexes, comme la `QComboBox` (liste déroulante), le double clic a pour effet de vous permettre d'éditer la liste des éléments contenus dans la liste déroulante.
- Pensez aussi à faire un **clic droit** sur les widgets pour modifier certaines propriétés, comme la bulle d'aide (`toolTip`).

Vous pouvez prévisualiser la fenêtre en faisant **Ctrl + R**, ou encore en allant dans le menu "Form / Preview".

Voici notre fenêtre en mode "Preview" :



Ce mode nous permet de tester la fenêtre telle qu'elle apparaîtra à la fin, de manipuler les widgets, etc. Sortez du mode Preview et revenez à l'édition, nous avons encore des choses à voir.

2.2. Utiliser les layouts

Pour le moment, nous n'utilisons aucun layout. Si vous essayez de redimensionner la fenêtre, vous verrez que les widgets ne s'adaptent pas à la nouvelle taille et qu'ils peuvent même disparaître si on réduit trop la taille de la fenêtre !

Il y a 2 façons d'utiliser des layouts :

- Utiliser la barre d'outils en haut.
- Glisser-déplacer des layouts depuis le dock de sélection de widgets ("Widget Box").

Pour une fenêtre simple comme celle-là, nous n'aurons besoin que d'un layout principal. Pour définir ce layout principal, le mieux est de passer par la barre d'outils :

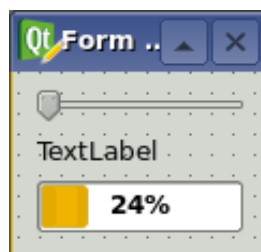


Cliquez sur une zone vide de la fenêtre (en clair, il faut que ce soit la fenêtre qui soit sélectionnée et non un de ses widgets). Vous devriez alors voir les boutons de la barre d'outils des layouts s'activer, comme sur l'image ci-dessus.

Cliquez sur le bouton correspondant au layout de grille pour organiser automatiquement la fenêtre selon un layout de grille. 😊

Vous pouvez aussi demander à ce que la fenêtre soit réduite à la taille minimale acceptable, en cliquant sur le bouton tout à droite de la barre d'outils, intitulé "Adjust Size".

Vous devriez alors voir vos widgets s'organiser comme ceci :



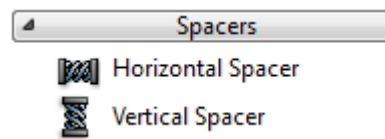
Maintenant que vous avez défini le layout principal de la fenêtre, sachez que vous pouvez insérer un sous-layout en plaçant par exemple un des layouts proposés dans la Widget Box.

2.3. Insérer des spacers

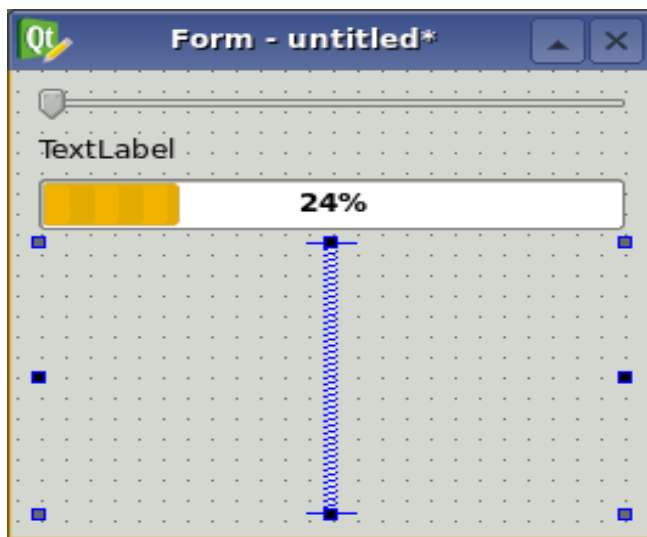
Vous trouvez que la fenêtre est un peu moche si on l'agrandit trop ?
Moi aussi. Les widgets sont trop espacés, ça ne me convient pas.

Pour changer la position des widgets *tout en conservant le layout*, on peut insérer un spacer. Il s'agit d'un widget invisible qui sert à créer de l'espace sur la fenêtre.

Le mieux est encore d'essayer pour comprendre ce que ça fait. Dans la Widget Box, vous devriez avoir une section "Spacers" :



Prenez un "Vertical Spacer", et insérez-le tout en bas de la fenêtre. Vous devriez alors voir ceci :

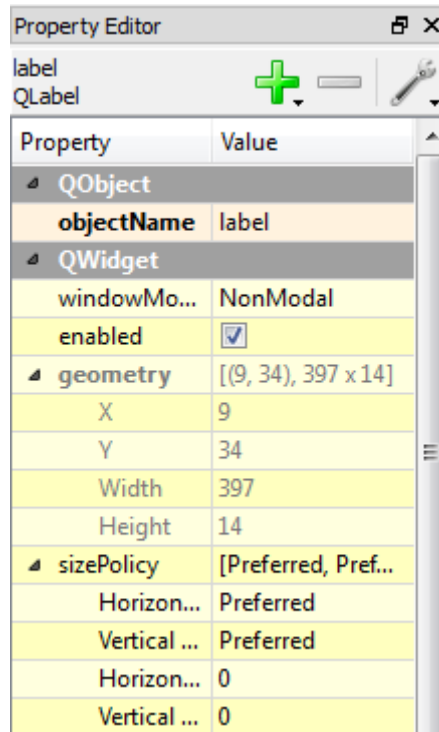


Le spacer va forcer les autres widgets à se coller tout en haut. Ils sont toujours organisés selon un layout, mais au moins maintenant nos widgets sont plus rapprochés les uns des autres.
Essayez de déplacer le spacer sur la fenêtre pour voir. Placez-le entre le libellé et la barre de progression. Vous devriez voir que la barre de progression se colle maintenant tout en bas.

Le comportement du spacer est assez logique, mais il faut l'essayer pour bien comprendre. 😊

3. Editer les propriétés des widgets

Il nous reste une chose très importante à voir : l'édition des propriétés des widgets.
Sélectionnez par exemple le libellé (`QLabel`). Regardez le dock intitulé "Property Editor". Il affiche maintenant les propriétés du `QLabel` :



Ces propriétés sont organisées en fonction de la classe dans laquelle elles ont été définies, et c'est une bonne chose. Je m'explique. Vous savez peut-être qu'un `QLabel` hérite de `QFrame`, qui hérite de `QWidget`, qui hérite lui-même de `QObject` ?

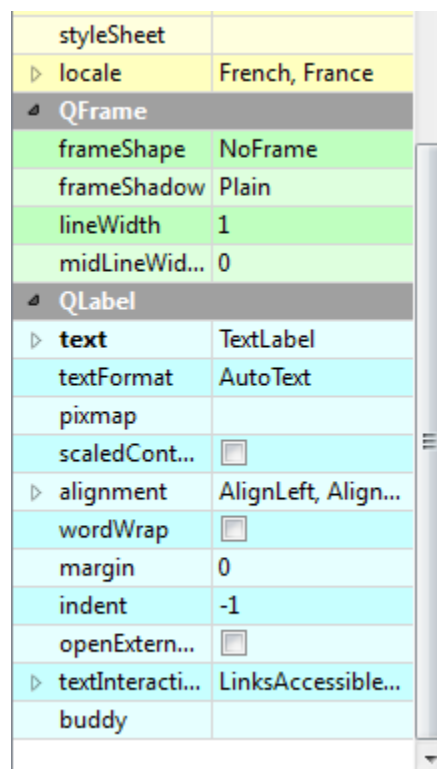
Chacune de ces classes définit des propriétés. `QLabel` hérite donc des propriétés de `QFrame`, `QWidget` et `QObject`, mais a aussi des propriétés qui lui sont propres.

Sur ma capture d'écran ci-dessus, on peut voir une propriété de `QObject` : `objectName`. C'est le nom de l'objet qui sera créé dans le code. Je vous conseille de le personnaliser pour que vous puissiez vous y retrouver dans le code source ensuite.

La plupart du temps, on peut éditer le nom d'un widget en double-cliquant dessus sur la fenêtre.

Si vous descendez un peu plus bas dans la liste, vous devriez vous rendre compte qu'un grand nombre de propriétés sont proposées par

QWidget (notamment la police, le style de curseur de la souris, etc.). Descendez encore plus bas. Vous devriez arriver sur les propriétés héritées de QFrame, puis celles propres à QLabel :



styleSheet	
▸ locale	French, France
▾ QWidget	
frameShape	NoFrame
frameShadow	Plain
lineWidth	1
midLineWidth	0
▾ QLabel	
▸ text	TextLabel
textFormat	AutoText
pixmap	
scaledContent	<input type="checkbox"/>
▸ alignment	AlignLeft, Align...
wordWrap	<input type="checkbox"/>
margin	0
indent	-1
openExternally	<input type="checkbox"/>
▸ textInteractions	LinksAccessible...
buddy	

Comme vous pouvez le voir, ces propriétés ont été mises en valeur : elles sont en vert. Je trouve que c'est très bien d'avoir organisé les propriétés comme ça. Ainsi, on voit bien où elles sont définies.

Vous devriez modifier la propriété *text*, pour changer le texte affiché dans le QLabel. Mettez par exemple "0". Amusez-vous à changer la police (propriété *font* issue de QWidget) ou encore à mettre une bordure (propriété *frameShape* issue de QFrame).

Vous remarquerez que lorsque vous éditez une propriété, son nom s'affiche en gras pour être mis en valeur. Cela vous permet par la suite de repérer du premier coup d'œil les propriétés que vous avez modifiées.

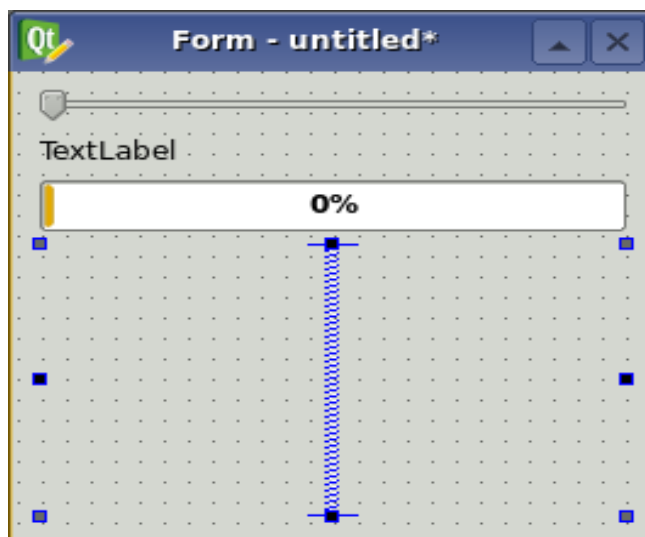
Certaines propriétés, comme *alignement* de `QLabel`, possèdent des sous-propriétés. Cliquez sur la petite flèche à gauche pour afficher et modifier ces sous-propriétés. Essayez de faire en sorte que le texte de notre libellé soit centré horizontalement par exemple.

Modifiez aussi les propriétés de la `QProgressBar` pour qu'elle affiche 0% pour défaut (propriété *value*).

Vous pouvez aussi modifier les propriétés de la fenêtre. Cliquez sur une zone vide de la fenêtre afin qu'aucun widget ne soit sélectionné. Le dock "Property Editor" vous affichera alors les propriétés de la fenêtre (ici, notre fenêtre est un `QWidget`, donc vous aurez juste les propriétés de `QWidget`).

Astuce : si vous ne comprenez pas à quoi sert une propriété, cliquez dessus puis appuyez sur la touche F1. Qt Designer lancera automatiquement Qt Assistant pour afficher l'aide sur la propriété sélectionnée.

Essayez d'avoir une fenêtre qui ressemble au final grosso modo à la mienne :



Le libellé et la barre de progression doivent afficher 0 par défaut.

Bravo, vous savez maintenant insérer des widgets, les organiser selon un layout et personnaliser leurs propriétés dans Qt Designer ! 😊
Nous n'avons utilisé pour le moment que le mode "Edit Widgets". Il nous reste à étudier le mode "Edit Signals/Slots"...

4. Configurer les signaux et les slots

Passez en mode "Edit Signals/Slots" en cliquant sur le second bouton de la barre d'outils :

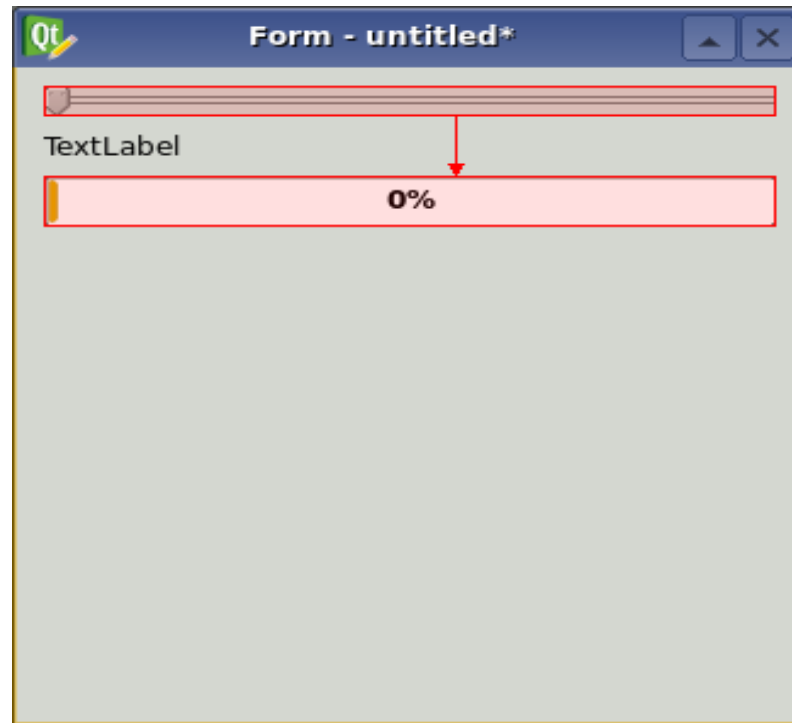


Vous pouvez aussi appuyer sur la touche F4. Vous pourrez faire F3 pour revenir au mode d'édition des widgets.

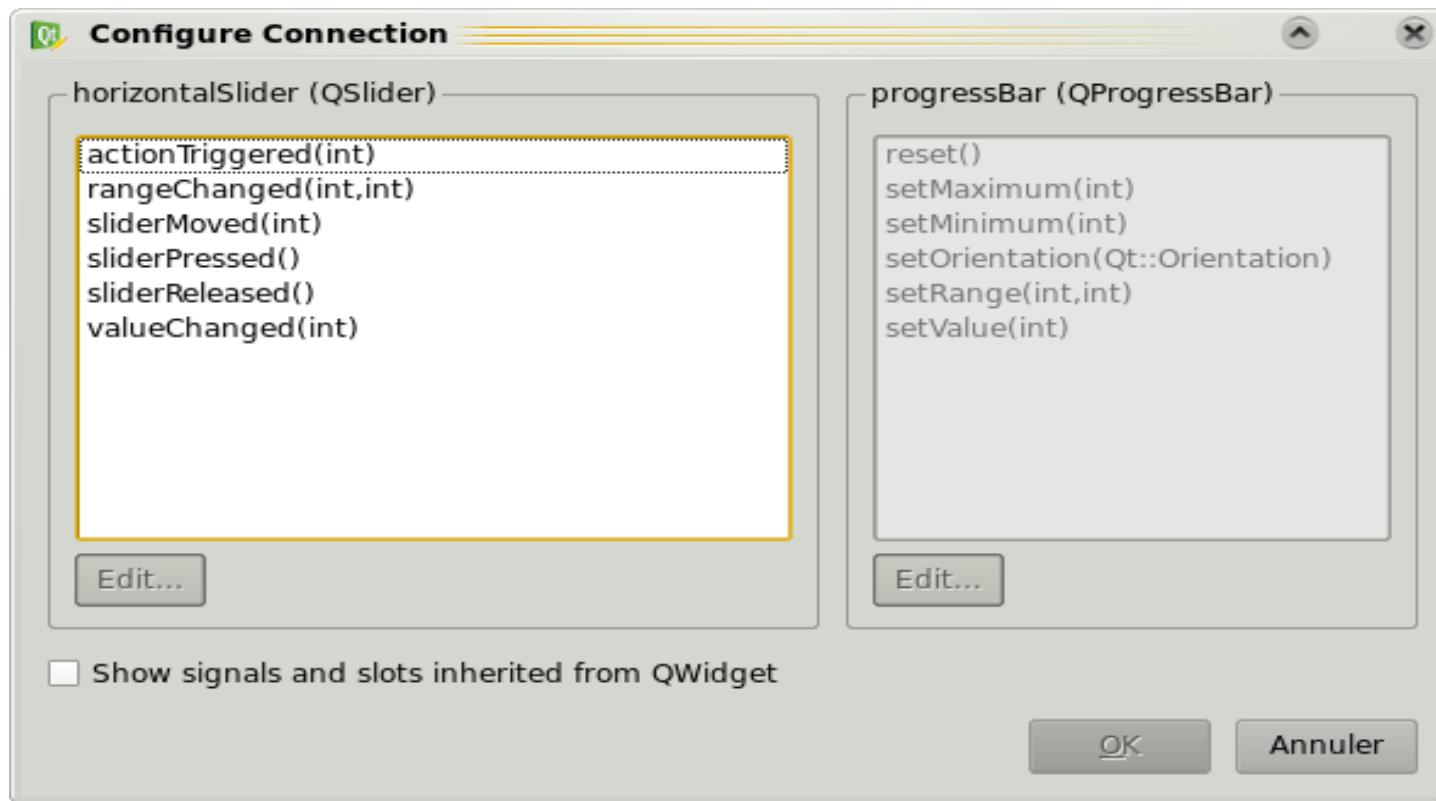
Dans ce mode, on ne peut pas ajouter, modifier, supprimer, ni déplacer de widgets. Par contre, si vous pointez sur les widgets de votre fenêtre, vous devriez voir un cadre rouge autour d'eux.

Vous pouvez, de manière très intuitive, associer les widgets entre eux pour créer des connexions simples entre leurs signaux et slots. Je vous propose par exemple d'associer le `QSlider` avec notre `QProgressBar`.

Pour cela, cliquez sur le `QSlider` et maintenez le bouton gauche de la souris enfoncé. Pointez sur la `QProgressBar` et relâchez le bouton. La connexion que vous allez faire devrait ressembler à ceci :



Une fenêtre apparaît alors pour que vous puissiez choisir le signal et le slot à connecter :



A gauche : les signaux disponibles dans le `QSlider`.

A droite : les slots *compatibles* disponibles dans la `QProgressBar`.

Sélectionnez un signal à gauche, par exemple `sliderMoved(int)`. Ce signal est envoyé dès que l'on déplace un peu le slider.

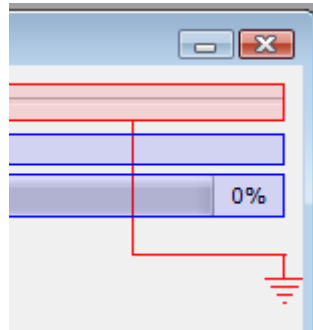
Vous verrez que la liste des signaux compatibles apparaît à droite.

En fonction du signal choisi, Qt Designer ne vous affiche que les slots de destination compatibles. Par exemple, `sliderMoved(int)` s'accorde bien avec `setValue(int)`. On peut aussi le connecter à `reset()`, dans ce cas le nombre envoyé en paramètre sera perdu. Par contre, on ne peut pas connecter le signal `sliderMoved(int)` au slot `setRange(int, int)` car le signal n'envoie pas assez de paramètres. D'ailleurs, vous ne devriez pas voir ce slot disponible dans la liste des slots si vous avez choisi le signal `sliderMoved(int)`, ce qui vous empêche de créer une connexion incompatible.

Nous allons connecter `sliderMoved(int)` du `QSlider` avec `setValue(int)` de la `QProgressBar`.
Faites OK pour valider une fois le signal et le slot choisis. C'est bon, la connexion est créée. 😊

Faites de même pour associer `sliderMoved(int)` du `QSlider` à `setNum(int)` du `QLabel`.

Notez que vous pouvez aussi connecter un widget à la fenêtre. Dans ce cas, visez une zone vide de la fenêtre. La flèche devrait se transformer en symbole de masse (bien connu par ceux qui font de l'électricité ou de l'électronique) :



Cela vous permet d'associer un signal du widget à un slot de la fenêtre, ce qui peut vous être utile si vous voulez créer un bouton "Fermer la fenêtre" par exemple.

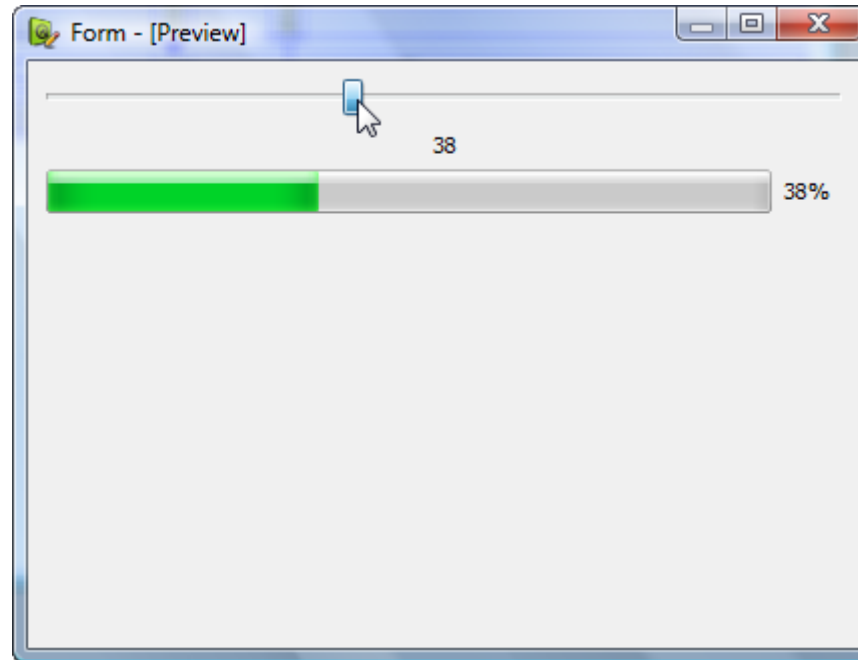
Attention : si dans la fenêtre du choix du signal et du slot vous ne voyez aucun slot s'afficher pour la fenêtre, c'est normal. Qt les masque par défaut car ils sont nombreux. Si on les affichait pour chaque connexion entre 2 widgets, on en aurait beaucoup trop (puisque tous les widgets héritent de `QWidget`).

Pour afficher quand même les signaux et slots issus de `QWidget`, cochez la case "*Show signals and slots inherited from QWidget*".

Passez maintenant en mode preview (**Ctrl + R**) pour tester vos connexions.

Essayez de déplacer le slider. Si vous avez fait les choses correctement, vous devriez voir le libellé et la barre de progression changer de

valeur en même temps ! 😊



Pour des connexions simples entre les signaux et les slots des widgets, Qt Designer est donc très intuitif et convient parfaitement. Eh, mais si je veux créer un slot personnalisé pour faire des manipulations un peu plus complexes, comment je fais ?

Qt Designer ne peut pas vous aider pour ça. Si vous voulez créer un signal ou un slot personnalisé, il faudra le faire tout à l'heure dans le code source. Comme vous pourrez le voir néanmoins, c'est très simple à faire.

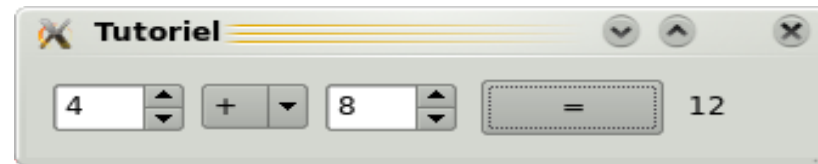
En y réfléchissant bien, c'est même d'ailleurs la seule chose que vous aurez à coder ! En effet, tout le reste est automatiquement géré par Qt Designer. Vous n'avez plus qu'à vous concentrer sur la partie "réflexion" de votre code source. Qt Designer vous permet donc de gagner du temps en vous épargnant les tâches répétitives et basiques qu'on fait à chaque fois que l'on crée une fenêtre.

5. Utiliser la fenêtre dans votre application

Il reste une dernière étape, et pas des moindres : apprendre à utiliser la fenêtre ainsi créée dans votre application.

5.1. Notre nouvel exemple

Je vous propose de créer une nouvelle fenêtre (parce que l'exemple de tout à l'heure était bien joli, mais pas très intéressant à part pour tester les signaux et slots 🤪). On va créer une mini-calculatrice :



Essayez de reproduire à peu près la même fenêtre que moi.
Un layout principal horizontal suffira à organiser les widgets.

La fenêtre est constituée des widgets suivants, de gauche à droite :

Widget	Nom de l'objet
QSpinBox	spinBoxNombre1
QComboBox	comboBoxOperation
QSpinBox	spinBoxnombre2
QPushButton	pushButtonEgal
QLabel	labelResultat

Pour la liste déroulante du choix de l'opération, je l'ai déjà pré-remplie avec 4 valeurs : +, -, * et /.

Double-cliquez sur la liste déroulante pour ajouter / supprimer des valeurs.

Pensez à bien renommer les widgets afin que vous puissiez vous y retrouver dans votre code source ensuite. 😊

Enregistrer le fichier sous le nom `calculatrice.ui` (l'extension `.ui` est rajoutée automatiquement à l'enregistrement par designer).

Tous les fichiers de fenêtres créés avec Qt Designer portent l'extension `.ui` (comme User Interface, "Interface Utilisateur" en français).

5.2. Le principe de la génération du code source

Essayons maintenant de récupérer le code de la fenêtre dans notre application et d'ouvrir cette fenêtre.

Le code ? Quel code ? Je ne vois pas de code moi ?

Qt Designer est censé générer un code source ?

Non, Qt Designer ne fait que produire un fichier `.ui`.

Pour produire du code python, il faut utiliser le programme `pyuic` ou `pyuic4` ou `pyuic-qt4` (le même programme qui porte un nom différent suivant la distribution Linux utilisée)

la ligne de commande

```
$ pyuic maFenetre.ui > maFenetre.py
```

Vous dessinez la fenêtre avec Qt Designer qui produit un fichier `.ui`, disons `maFenetre.ui`

Ce fichier est transformé automatiquement en code source par le petit programme en ligne de commande `pyuic`.

Celui-ci génèrera un fichier que je vous recommande de nommer `maFenetre.py`. Ce fichier contient une classe python, du nom de `Ui_nomDeVotrefenetre.py`.

5.3. Utiliser la fenêtre dans notre application

Il nous reste une importante étape : modifier le code source de notre application pour ouvrir la fenêtre créée sous Qt Designer. Et là, nous avons le choix. Nous pouvons utiliser la fenêtre de 3 manières différentes, de la plus simple à la plus compliquée (la plus compliquée étant la meilleure bien sûr 😊) :

- Utilisation directe
- Utilisation avec un héritage simple
- Utilisation avec un héritage multiple

Je vais vous décrire chacune de ces 3 méthodes. 😊
Vous verrez que la dernière, bien que plus complexe, est la plus pratique et la plus souple.

i) Utilisation directe

Avantages : technique très simple à mettre en oeuvre, à peine quelques lignes à écrire.

Défauts : pas de possibilité de personnaliser la fenêtre, ni d'écrire des slots personnalisés. La fenêtre est "figée".

La technique la plus simple, mais la moins puissante, consiste à utiliser directement la fenêtre générée. On va supposer que votre programme n'est constitué que d'un `main()`. Ajoutez les lignes surlignées :


```

#!/usr/bin/python
#-*-coding: utf-8 -*-
from PyQt4.QtGui import *
from PyQt4.QtCore import *
import os,sys
from calculatrice import *

def main(args):
    a=QApplication(args)
    f=QWidget()
    c=Ui_calculatrice()
    c.setupUi(f)
    f.show()
    r=a.exec_()
    return r

if __name__=="__main__":
    main(sys.argv)#include <QApplication>

```

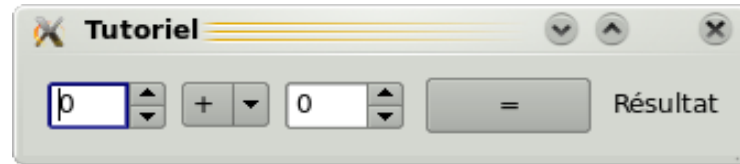
Dans un premier temps, on inclut le fichier `calculatrice.py` qui sera généré par `pyuic4` à l'étape précédente.

Ensuite, on fait comme si on créait une nouvelle fenêtre en créant un nouvel objet de type `QWidget`.

Au lieu d'afficher cette fenêtre directement, on la précharge avec le contenu que l'on a dessiné dans Qt Designer. Pour cela, on crée un objet de type `Ui_calculatrice` (où "calculatrice" est le nom que vous avez donné à votre fenêtre dans Qt Designer). On appelle `setupUi(fenetre)` pour dessiner le contenu de la fenêtre avec l'interface réalisée sous Qt Designer.

On peut ensuite ouvrir la fenêtre avec `fenetre.show()`; comme d'habitude. 😊

Admirez ensuite le programme ainsi généré :



Ca marche ! 😊

Vous noterez toutefois qu'il y a un défaut : notre fenêtre s'affiche, c'est bien beau, mais elle ne réagit au clic sur le bouton "=". En effet, la méthode que nous venons de voir est très simple, mais elle a un énorme défaut : nous ne pouvons pas créer nos propres slots pour personnaliser un peu le code de la fenêtre.

Les techniques suivantes que nous allons voir nous permettent de le faire, et sont donc bien plus souples.

ii) Utilisation avec un héritage simple

Avantages : on peut personnaliser la fenêtre et écrire nos propres slots.

Défauts : il faut utiliser le préfixe "ui" devant les noms de tous les widgets pour pouvoir les utiliser.

Nous allons hériter de la fenêtre créée avec Qt Designer. Pour faire cela, nous allons créer une nouvelle classe dans notre projet intitulée "calculatrice" (du même nom que la fenêtre créée sous Qt Designer, oui oui).

Pour cela, nous reprenons notre fichier `calculatriceMain.py`, et nous y introduisons l'implémentation de la classe `calculatrice`.

Le programme principal sera situé dans le même fichier que la classe `calculatrice`, ce qui du point de vue purement informatique est une hérésie, mais que nous justifierons « à la python » : le fichier contenant la classe `calculatrice` contient également la batterie de programmes (ici le `main()`) permettant de tester la classe `calculatrice`

Au final, votre projet devrait comporter les fichiers suivants :

- calculatriceMain.py
- calculatrice.py

Définissez le fichier **calculatriceMain.py** comme ceci :

```
#!/usr/bin/python
#-*-coding: utf-8 -*-
from PyQt4.QtGui import *
from PyQt4.QtCore import *
import os,sys
from calculatrice import *
class calculatrice(QWidget):
    def __init__(self, parent=None):
        QWidget.__init__(self)
        self.ui=Ui_calculatrice()
        self.ui.setupUi(parent)
        #Ici, personnalisez vos widgets si nécessaire
        #Réalisez les connexions supplémentaires entre signaux et slots
def main(args):
    a=QApplication(args)
    f=QWidget()
    c=calculatrice(f)
    f.show()
    r=a.exec_()
    return r

if __name__=="__main__":
    main(sys.argv)
```

On importe "calculatrice" pour pouvoir utiliser la fenêtre créée avec Qt Designer.

On crée une classe `calculatrice` héritant de `QWidget`. Ehhh oui, il faut hériter du même type que la fenêtre créée sous Qt Designer (qui

était un `QWidget` si vous vous souvenez bien).

On crée un constructeur classique.

On déclare un objet "ui" de type `Ui_calculatrice`. Ça c'est la particularité. L'objet `ui` contiendra tous les widgets de la fenêtre, vous allez voir.

Tout ce que vous avez à faire, c'est un `ui.setupUi(parent)` pour créer le contenu de la fenêtre.

Il faut faire cela en premier dans le constructeur. Ensuite, libre à vous de personnaliser les widgets et de créer des connexions supplémentaires entre des signaux et des slots.

Particularité : tous les widgets sont accessibles en faisant `self.ui.nomDuWidget`.

Par exemple, on peut changer le texte du bouton comme ceci :

```
class calculatrice(QWidget):
    def __init__(self, parent=None):
        QWidget.__init__(self)
        self.ui=Ui_calculatrice()
        self.ui.setupUi(parent)
        self.ui.pushButtonEgal.setText("Egal");
```

Le nom du bouton "boutonEgal", nous l'avons défini dans Qt Designer tout à l'heure (propriété `objectName` de `QObject`). Retournez voir le petit tableau un peu plus haut pour vous souvenir de la liste des noms des widgets de la fenêtre.

Bon en général vous n'aurez pas besoin de personnaliser vos widgets, vu que vous avez tout fait sous Qt Designer. Mais si vous avez besoin d'adapter leur contenu à l'exécution (pour afficher le nom de l'utilisateur par exemple), il faudra passer par là.

Maintenant ce qui est intéressant surtout, c'est d'effectuer une connexion :

```
class calculatrice(QWidget):
    def __init__(self, parent=None):
        QWidget.__init__(self)
        self.ui=Ui_calculatrice()
        self.ui.setupUi(parent)
        self.connect(self.ui.pushButtonEgale,
                    SIGNAL("clicked()"),
                    self.calcul)
```

N'oubliez pas à chaque fois de mettre le préfixe "self.ui" devant chaque nom de widget !

Ce code nous permet de faire en sorte que le slot `calcul()` de la fenêtre soit appelé à chaque fois que l'on clique sur le bouton.

La méthode que nous venons de voir est très pratique et on peut faire tout ce qu'on veut avec, mais il faut écrire le préfixe "self.ui" devant le nom du widget à chaque fois.

Si vous voulez évitez d'avoir à écrire "self.ui", il va falloir faire un héritage multiple...

iii) Utilisation avec un héritage multiple

Avantages : on peut personnaliser la fenêtre, écrire nos propres slots, et on n'a pas besoin de mettre le préfixe "ui" devant chaque nom de widget.

Défauts : il faut faire un héritage multiple, une technique un peu plus complexe que l'héritage classique.

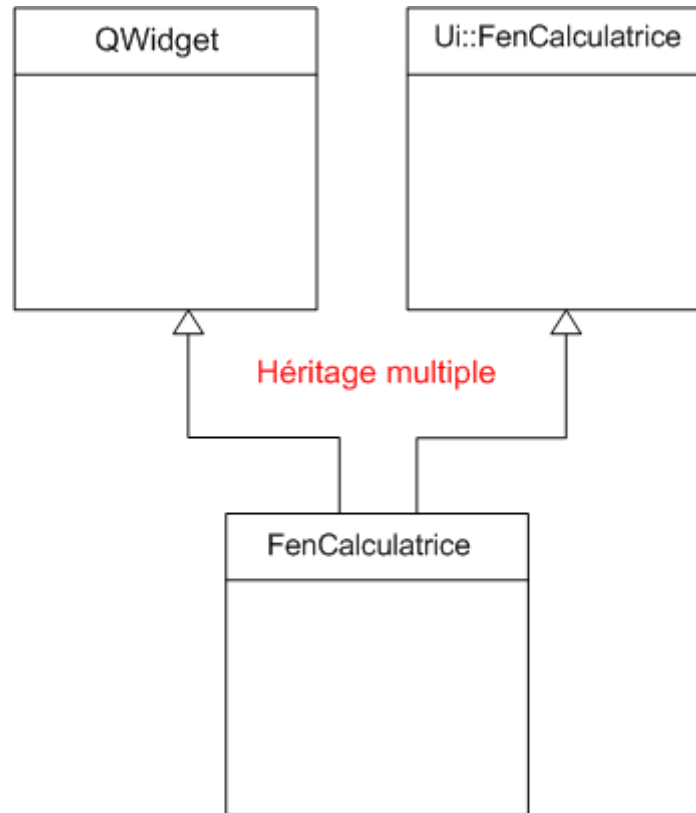
L'héritage multiple est une technique complexe du C++ et de python. Il faut dire qu'on l'utilise rarement et, bien que cette technique soit puissante, elle est considérée comme trop complexes par certains nouveaux langages (Java, Ruby...) qui ont décidé de ne pas gérer l'héritage multiple.

Python l'implémente, mais tous les cas ne sont pas autorisés. A retenir : en PyQt4, une classe peut hériter d'une classe **PyQt4** au **maximum**. Mais elle peut hériter d'une classe PyQt4 et d'une classe python ordinaire (une de vos classes, donc). Ce principe permet d'implémenter la

méthode qui suit :

Bon, le principe c'est quoi ? A priori c'est tout bête : c'est **une classe qui hérite de 2 classes** (ou plus !).

Dans notre cas, il faut que l'on hérite à la fois de `QWidget` (le type de la fenêtre) et de `Ui::calculatrice` (la fenêtre créée sous Qt Designer) !



```
class calculatrice(QWidget,Ui_calculatrice):
    def __init__(self, parent=None):
        QWidget.__init__(self)
        self.setupUi(parent)
        self.connect(self.pushButtonEgal,
                    SIGNAL("clicked()"),
                    self.calcul)
```

La seule ligne qui change a été surlignée, c'est celle de déclaration de la classe. On hérite de `QWidget` et de `Ui_calculatrice` à la fois. L'instanciation de `ui` (la ligne `self.ui=Ui_calculatrice()`) a été supprimée on n'a plus besoin de définir un objet "ui" de type `Ui_calculatrice` cette fois.

Pour compléter, donnons le code de la méthode `calcul()` connectée au `pushButtonEgal`

```
def calcul(self):
    n1=self.spinBoxNombre1.value()
    n2=self.spinBoxNombre2.value()
    op=self.comboBoxOperation.currentText()
    if op=='+' :self.labelResultat.setText(str(n1+n2))
    else : self.labelResultat.setText(self.trUtf8("non
implémenté"))
```

On récupère la valeur des nombres `n1` et `n2` dans les `spinBox`
On récupère la valeur de l'opérateur (`+`, `-`, `*` ou `/`) dans la `comboBox`
on calcul suivant la valeur de l'opérateur
on insère le résultat dans le `labelResultat`

Voici le résultat :

ça semble fonctionner

5.4. Personnaliser le code et utiliser les Auto-Connect

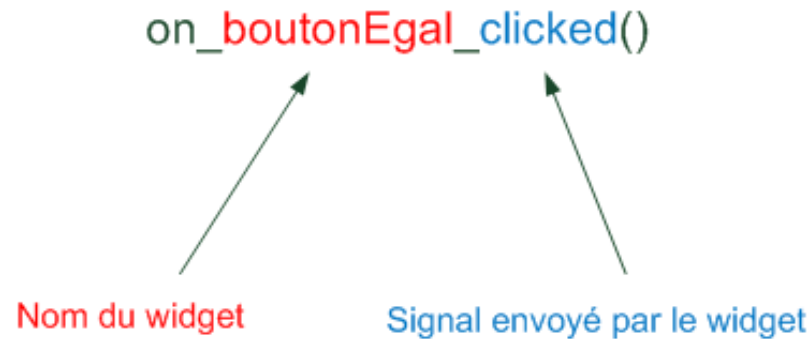
Les fenêtres créées avec Qt Designer bénéficient du système "Auto-Connect" de Qt. C'est un système qui crée les connexions tout seul.

Par quelle magie ?

Il vous suffit en fait de créer des slots en leur donnant un nom qui respecte une convention.

Prenons le widget `pushButtonEgal` et son signal `clicked()`. Si vous créez un slot appelé `on_pushButtonEgal_clicked()` dans votre fenêtre, ce slot sera automatiquement appelé lors d'un clic sur le bouton.

La convention à respecter est représentée sur le schéma ci-dessous :



Essayons d'utiliser l'Auto-Connect dans notre programme. Je me base ici sur un héritage multiple.

Pour le moment, en PyQt4, ça ne fonctionne pas

...

Voici le .h :


```
1 #ifndef HEADER_FENCALCULATRICE
2 #define HEADER_FENCALCULATRICE
3
4 #include <QtGui>
5 #include "ui_calculatrice.h"
6
7 class FenCalculatrice : public QWidget, private Ui::FenCalculatrice
8 {
9     Q_OBJECT
10
11     public:
12         FenCalculatrice(QWidget *parent = 0);
13
14     private slots:
15         void on_boutonEgal_clicked();
16 };
17
18
19 #endif
```

... et le .cpp :

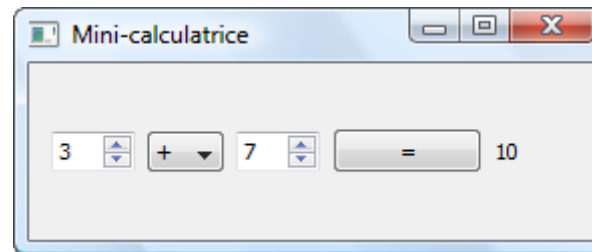
```

1 #include "FenCalculatrice.h"
2
3 FenCalculatrice::FenCalculatrice(QWidget *parent) : QWidget(parent)
4 {
5     setupUi(this);
6 }
7
8 void FenCalculatrice::on_boutonEgal_clicked()
9 {
10     int somme = nombre1->value() + nombre2->value();
11     resultat->setNum(somme);
12 }

```

Vous noterez qu'on n'a plus besoin de faire de connexion dans le constructeur. Ben oui, c'est le principe de l'Auto-Connect. 🤖
Comme vous le voyez, il suffit de créer un slot avec un nom particulier, et tout roule comme sur des roulettes !

Vous pouvez tester le programme, ça marche !



Bon, j'avoue, je n'ai géré ici que l'addition. Mais je vais pas tout vous faire non plus hein. 🤖

Exercice (me dites pas que vous l'avez pas vu venir 🤖) : complétez le code de la calculatrice pour effectuer la bonne opération en fonction de l'élément sélectionné dans la liste déroulante.

L'Auto-Connect est activé par défaut dans les fenêtres créées avec Qt Designer, mais vous pouvez aussi vous en servir dans vos autres fenêtres "faites main".

Il suffira d'ajouter la ligne suivante dans le constructeur de la fenêtre pour bénéficier de toute la puissance de l'Auto-Connect :

QObject::connectSlotsByName(this);

Ceux qui croyaient que Qt Designer était un "*programme magique qui allait réaliser des fenêtres tout seul sans avoir besoin de coder*" en ont été pour leurs frais ! 🤖

Pourtant, comme avec Qt Linguist, le processus de création de fenêtres de Qt Designer a été très bien pensé. Tout est logique et s'enchaîne de bout en bout, mais encore faut-il comprendre cette logique. J'espère vous y avoir aidé à travers ce chapitre.

J'ai profité de ce chapitre pour survoler la notion d'**héritage multiple**. Il y aurait à redire sur cette notion, mais c'est un peu complexe et on peut se contenter de retenir ce qui a été dit dans ce chapitre si on a juste besoin d'utiliser des fenêtres créées avec Qt Designer. Si vous voulez en savoir plus (et je vous y encourage), il vous faudra chercher d'autres cours sur le web traitant de l'héritage multiple en C++.

Entraînez-vous à utiliser quelques fenêtres créées avec Qt Designer, et en particulier à créer des slots personnalisés.

Tant qu'à faire, je vous conseille de vous servir de l'Auto-Connect. Une fois qu'on y a goûté on ne peut plus s'en passer. 🤖