Qt3 -> Qt4

Modifications à faire pour passer de qt3 à qt4 en Python

On trouve une doc présentant les différences sur le site de Qt : http://doc.trolltech.com/4.4/porting4.html

Importation

Le module qt devient PyQt4 et divisé en sous-modules, dont QtGui pour les objets graphiques et QtCore pour les classes non graphique du coeur de Qt.

Name

Le constructeur de QObject ne prend plus de paramètre name, de même que les constructeurs des objets dérivés de QObject comme QWidget par exemple. La propriété name devient objectName et peut être modifiée par la méthode setObjectName.

Flags

Le type des Flags est maintenant QtCore.Qt.WindowFlags. On ne peut plus passer en paramètres des fonctions wflags=0 car cela fait une erreur de type. A la place on peut omettre le paramètre ou mettre wflags=QtCore.Qt.WindowFlags().

Modality

Les widget ont une méthode setWindowModality permettant de changer le flag WindowModality pour NonModal, WindowModal, ApplicationModal. Les dialog sont automatiquement en mode modal si on utilise la méthode exec (exec_ en python). Si on utilise le valeur WindowModal pour WindowModality, seules les fenêtres parentes seront bloquées.

QWidget

setCaption -> setWindowTitle
setIcon -> setWindowIcon

QIcon et QPixmap

Pour la plupart des widgets : setPixmap -> setIcon
QIcon(filename) ou QIcon(QPixmap)

Layouts

Il faut affecter le layout à la widget avec QWidget.setLayout(QLayout).
Puis ajouter les composants au layout : QLayout.addWidget(QWidget).
Attention, ne pas mettre le widget en parent dans le constructeur du layout, c'est redondant avec le setLayout, et ne pas mettre le widget en parent dans le constructeur des composants, c'est redondant avec addWidget.

QBoxLayout

Ne prend plus les attributs margin et spacing en paramètres du constructeur.
Les classes QHBox et QVBox n'existent plus, il faut les remplacer par des QWidget auxquels on met un QHBoxLayout ou un QVBoxLayout.

QGridLayout

Le constructeur ne peut plus prendre en paramètre le nombre de lignes et de colonnes, le spacing et le margin. Pas de méthodes pour fixer le nombre de lignes ni le nombre de colonnes. Au moment d'ajouter un élément, on donne la position dans le grid : addWidget(QWidget, row, col).
row et col commencent à 0.
Pour spécifier spacing et margin : setSpacing et setContentsMargin.

QFileDialog

L'ordre des paramètres des fonctions statiques getExistingDirectory et autres a changé.

Signal

les fonctions SIGNAL, SLOT sont maintenant dans le module QtCore. La fonction PYSIGNAL n'existe plus, on peut mettre le nom d'une fonction python en paramètre de SIGNAL.

QComboBox

setCurrentText -> setEditText
insertItem -> addItem (la methode insertItem existe aussi mais demande un index)
setCurrentItem -> setCurrentIndex
currentItem -> currentIndex

Button

On ne peut plus associer un QPixmap à un bouton mais un QIcon, qui peut se contruire à partir d'un pixmap.
Dans le constructeur avec le label du bouton et un widget parent, le parent est le deuxième paramètre.

Qt Designer et .ui

Le format des fichiers .ui générés par qt designer a changé dans la version 4. Les fichiers .ui ne sont pas compatibles d'une version à l'autre. Il est possible de convertir un ancien .ui dans le nouveau format avec la command uic3 -convert ancien.ui >nouveau.ui mais ça ne marche pas très bien quand le fichier contient des icones.
Pour charger un .ui : uic.loadUi(uiFile, baseInstance). Cela ajoute automatiquement les sous widgets en attributs dans le widget principal.

QProcess

La classe n'a plus de méthode addArgument, on passe les arguments au démarrage du process via la méthode start. La façon de récupérer le exitStatus et exitCode a changé aussi.

Threads et evennements

Chaque QThread peut avoir une boucle d'évennement qui se lance avec la fonction QThread.exec(). Un objet créé pendant l'exécution d'un thread est considéré comme appartenant à ce thread. On peut savoir quel est le thread associé à un object avec la fonction QObject.thread(). Les evennements concernant cet object seront alors gérés par la boucle d'évennement du thread. Attention, si un thread n'a pas de boucle d'évennement, les évennements de ses objets seront ignorés.
Plus d'infos ici : http://doc.trolltech.com/4.3/threads.html#per-thread-event-loop

QListView et QListViewItem, QListBox, QListBoxItem

Utiliser à la place QListWidget et QListWidgetItem, ou QTreeWidget si on a besoin d'une organisation hiérarchique.
QListWidget : une liste sans hierarchie. Pour mettre un item dedans : item=QListWidgetItem(liste) ou liste.addItem(item) ou liste.insertItem(row, item).
Supression d'un item : liste.takeItem(row).
Item sélectionné : liste.currentItem(), liste.currentRow()
Parcours : liste.itemAt(row), liste.count().
Pour avoir des items avec case à cocher : item.setCheckState(Qt.Checked) ou Qt.Unchecked...
Si on veut avoir un modèle personnalisé, il faut utiliser QListView et QTreeView.
QTreeWidget : setColumnCount(int), setColumnWidth(index, width), setHeaderLabels([])
renommer un item : QTreeWidget.editItem(item, column)

La taille des icônes est la même dans toute la vue et se définit au niveau du QTreeWidget avec la fonction setIconSize(QSize)

Tooltips

QWidget.setToolTip(string)

QPopupMenu

remplacer par QMenu. QMenu.addAction(texte, receiver)
Pour ouvrir le menu, possibilité de redéfinir contextMenuEvent(event) ou d'associer une méthode au signal customContextMenuRequested(const QPoint &). Attention pour que ce signal soit émis, il faut définir au niveau du widget setContextMenuPolicy(Qt.CustomContextMenu).

Fenêtre avec barre de menu

Utiliser QMainWindow (fenetre avec un widget principal, une barre de menu et une barre d'état) et les méthodes
- setCentralWidget(QWidget) pour définir le widget principal dans la fenetre,
- menuBar().addMenu(QString) : QMenu ... pour définir la barre de menu

Drag & drop

acceptDrops() -> bool
QDrag object contenant un QMimeData -> setText, setImageData... QDrag.setMimeData(QMimeData), QDrag.setPixmap(QPixmap)
QDrag.exec_() dans la fonction mousePressEvent pour démarrer le drag
Pour decoder un dragevent : QEvent.mimeData.hasText()

QMimeSourceFactory

Pour ajouter un chemin dans lequel rechercher les ressources (images, icones, docs...), utiliser QDir.addSearchPath(prefix, path) à la place de QMimeSourceFactory.defaultFactory().addFilePath(path). Le prefix permet de spécifier le type de ressource, par exemple "images". Ensuite pour chercher ce type de ressource, on écrit : "images:nomdufichier".

PaintEvent

Pour dessiner dans un widget, redéfinir paintEvent(QPaintEvent)
Y créer un QPainter(QWidget) et utiliser les primitives de dessin.
Méthode update pour mettre a jour l'affichage.
Pour les widget avec un scroll, il y a un viewport. C'est sur ce viewport qu'il faut appeler update et créer le QPainter.

QScrollView

Remplacé par QScrollArea. Créer un QWidget et l'ajouter au scroll area avec setWidget(). On peut accéder à ce widget avec la méthode widget(). Pour que le scroll area redimensionne automatiquement le widget interne, utiliser setWidgetResizable(True).

QSplitter

setResizeMode(widget, mode) -> setStretchFactor(index, stretch factor)
Utiliser la fonction addWidget(widget)

QWidgetStack -> QStackedWidget

QWT

Qwt 4 n'est pas compatible Qt4, il faut passer à Qwt5. L'api a un peu changé. Notamment pour ajouter des courbes :

```
plot=QwtPlot()
curve = QwtPlotCurve('name')
curve.setStyle(QwtPlotCurve.Lines)
curve.setPen(QPen(color))
curve.attach(plot)
```