Intégrer Inventor et Qt

Jean-Yves Didier

Université d'Evry

9 février 2010

Qt : présentation

- Bibliothèque C++ destinée aux IHM, oeuvre de la société Trolltech (rachetée par Nokia en 2008) ;
- Licence QPL/GPL :
  - développement avec la version GPL, travail dérivé sous GPL ;
  - développement propriétaire, utilisation de la licence propriétaire.
- Multi plate-forme : X11 / Windows :
  $\rightarrow$ fournit une couche d'abstraction par rapport à l'OS.
- *Bindings* Inventor/Qt : bibliothèque *SoQt* ou *Quarter*.
- Plusieurs versions cohabitent actuellement :
  - Qt 3.3.8b (KDE3) – La version vue dans ces transparents ;
  - Qt 4.x (KDE4).

Gestion des évènements

Qt repose sur le conception *signal/slot* :

- Il s'agit d'un système de *callbacks* déguisé ;
- **Signal** : appel à une fonction de *callback* potentielle ;
- **Slot** : fonction de *callback* ;
- Signaux et slots peuvent être connectés/déconnectés dynamiquement ;
- Toute classe dérivant de `QObject` peut implémenter ce mécanisme.

Créer ses propres signaux/slots

Créer une classe dérivant de QObject

```
#include <qobject.h>
class Toto:public QObject  // ou classe dérivant de
    QObject
{
  Q_OBJECT // nécessaire pour le mécanisme signal/slot
  public:
    Toto( ...);
  public slots: // peuvent être private ou protected
    void slotA(); // void : type de retour imposé
    void slotB(int a);
  signals:
    void signalA();  // void : type de retour imposé
};
```

Créer ses propres signaux/slots

- Un slot a une implémentation ;
- Un signal n'a pas d'implémentation !
- Émettre un signal :

  → `emit monSignal(param1,param2,...);`
- Connecter un signal et un slot :

  → `connect(objectA, SIGNAL(monSignal(int,int)), objectB, SLOT(monSlot(int,int)));`
  - Utilisation des macros `SLOT` et `SIGNAL` ;
  - `objectA` et `objectB` sont de type `QObject*` ;
  - les signatures des signaux et des slots doivent correspondre.

Les widgets Qt

- Pas de liste exhaustive ici, juste les principaux ;
- Dérivent d'une classe `QWidget` qui dérive de `QObject`
  $\rightarrow$ chacun possède des signaux et des slots !
- Widgets organisés en hiérarchie : les fenêtres, menus et *layout* sont les parents d'autres widgets ;
- Pour chaque widget, nous verrons :
  - le constructeur ;
  - l'aspect ;
  - les principaux signaux et slots.

Les widgets Qt

QWidget : la classe mère

→ `QWidget(QWidget* parent=0);`

Slots :

```
void show();    void move(int, int);
void hide();    void resize(int, int);
void enabled(bool);
```

QLabel : affichage de texte

→ `QLabel(QString text, QWidget* parent=0);`

Slots :

```
void setText(Qstring);
void clear();
```

Les widgets Qt

Les boutons

QPushButton : le bouton classique

→ `QPushButton(QString text, QWidget* parent=0);`

Signaux :

```
void clicked();
```

QCheckBox : la case à cocher

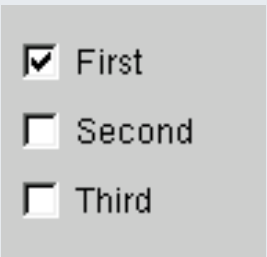
→ `QCheckBox(QString text, QWidget* parent=0);`

Slots :

```
void setChecked(bool);
```

Signaux :

```
void clicked();
void toggled(bool);
```



Les widgets Qt

Les boutons

QButtonGroup : grouper les boutons radio

→ `QButtonGroup(QString text, QWidget* parent=0);`

Signaux :

```
void clicked(int);
```

QButtonRadio : le bouton radio

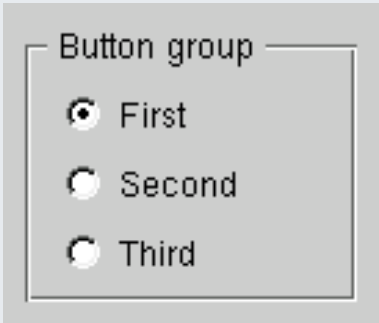
→ `QRadioButton(QString text, QWidget* parent=0);`

Slots :

```
void setChecked(bool);
```

Signaux :

```
void clicked();
void toggled(bool);
```



Les widgets Qt

Les listes

QListBox : liste en lecture seule

→ `QListBox(QWidget* parent=0);`

Slots :

```
void clear();
```

Signaux :

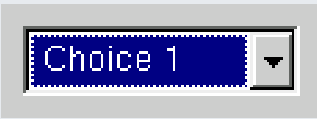
```
void selected(int);
void selected(QString);
```

QComboBox : liste déroulante (éditable)

→ `QComboBox(QWidget* parent=0);`

Signaux/slots :

cf `QListBox`



Les widgets Qt

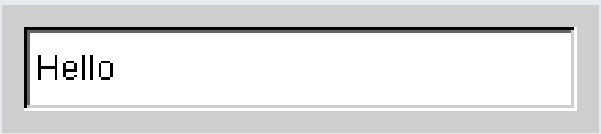
Saisir du texte

QLineEdit : pour du texte et des valeurs

→ `QLineEdit(QString contents,QWidget* parent);`

Slots :

```
void setText(QString);
void clear();
```



Signaux :

```
void textChanged(QString);
void returnPressed();
```

Pour des valeurs entières ou flottantes :

```
void setValidator(QValidator* );
```

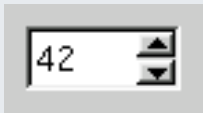
`QIntValidator`, `QDoubleValidator`

Les widgets Qt

Spinner et slider

QSpinBox : spinner

→ `QSpinBox(int min, int max, int step, QWidget* parent=0);`



Slots :

`void setValue(int);`

Signaux :

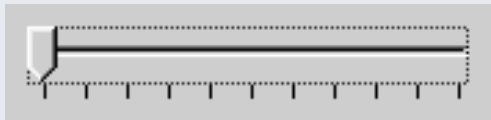
`void valueChanged(int);`

QSlider : ascenseurs

→ `QSlider(int min, int max, int step, int v, Orientation ori={Qt::Vertical|Qt::Horizontal}, QWidget* parent=0 );`

Signaux/slots :

cf `QSpinBox`



Les widgets Qt

Fenêtres et mise en page

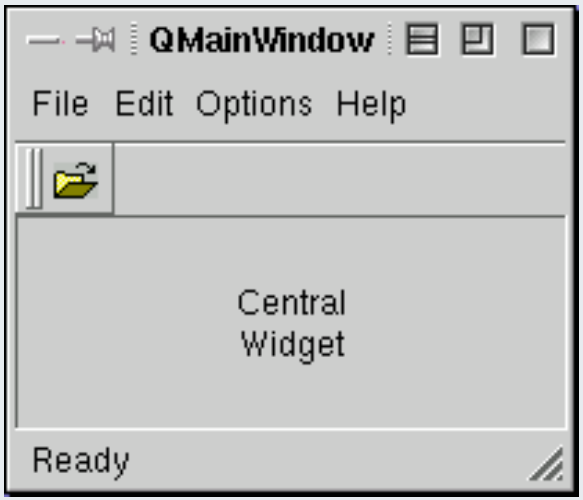
`QMainWindow` : la fenêtre principale

→ QMainWindow(QWidget* parent) ; Contient :

- Une barre de menu : `QMenuBar` ;
- Une barre d'état : `QStatusBar` ;
- Un widget central : `QWidget`.

Le widget central peut-être un *layout* :

- en grille : `QGrid` ;
- vertical : `QVBox` ;
- horizontal : `QHBox`.



Utilitaires Qt

qmake : le gestionnaire de projets Qt

Capable de générer un `Makefile` ou un projet *Visual Studio* suivant la plate-forme cible à partir d'un projet (extension `.pro`). Nécessaire pour appeler les autres utilitaires Qt.

Exemple de fichier projet pour intégrer Inventor

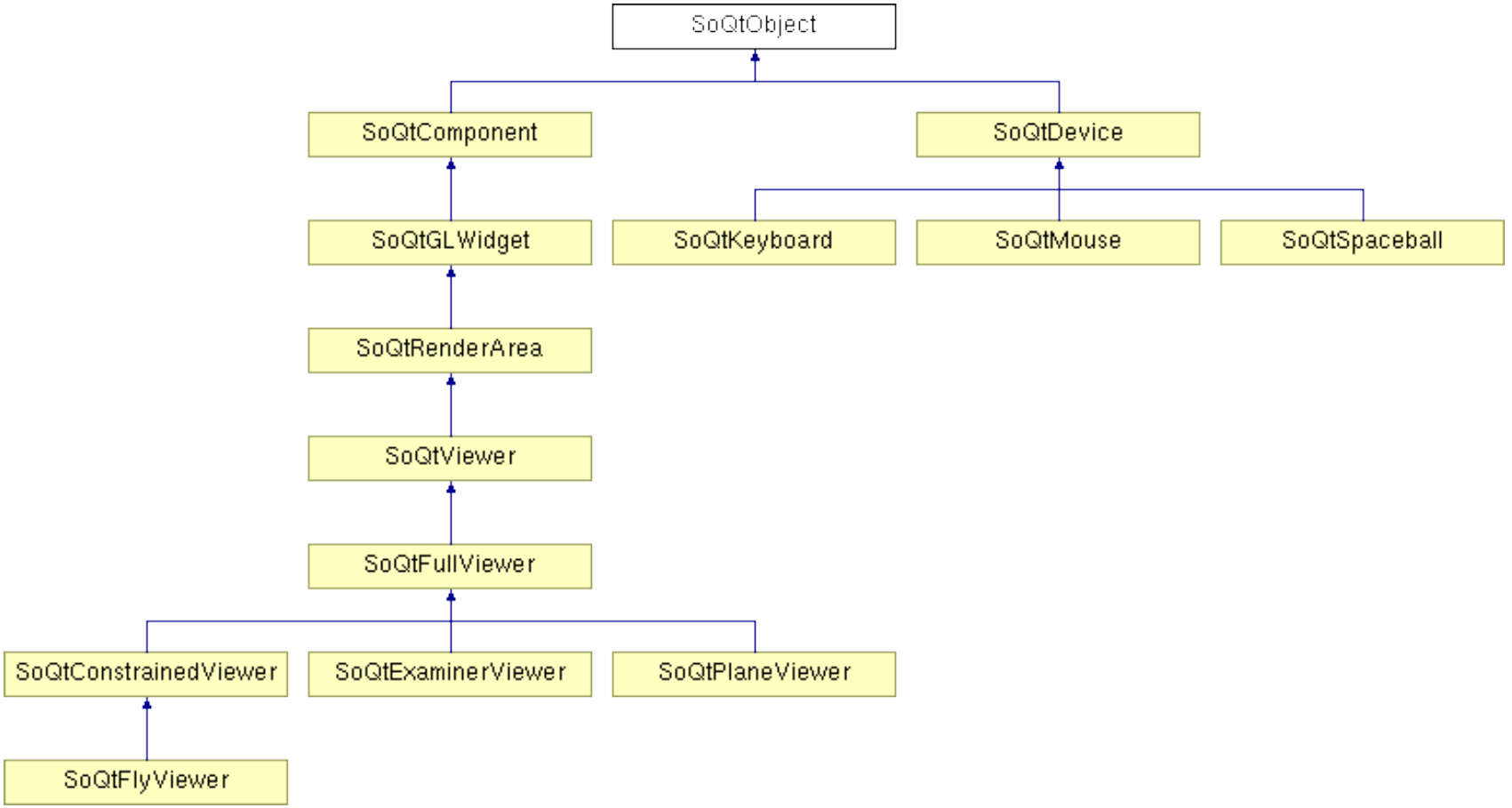
```
TEMPLATE = app
SOURCES = cone.cpp
INCLUDEPATH+= $$system(soqt-config --includedir)
LIBS += $$system(soqt-config --ldflags --libs)
```

Les autres utilitaires

moc – (Meta Object Compiler) : gestion des signaux et des slots ;

uic – (UI Compiler) : interfaces générées par le **designer**.

La hiérarchie SoQt



A vous de jouer !