TP Qt : QtCreator / QtDesigner

Sommaire

Environnement de Développement Intégré (EDI) **2**

Travail pratique **2**

Objectifs 2
Mise en situation 2
Créer un projet Qt 3
Qt Designer 7
Utiliser le code généré 10
Créer ses propres signaux et slots 15
Modifier le code généré 17

Il est fortement conseillé d'utiliser la documentation Qt de référence en français (http ://qt.developpez.com/doc/) ou en anglais (http ://qt-project.org/doc/).

Environnement de Développement Intégré (EDI)

Qt Creator est l'environnement de développement intégré dédié à Qt et facilite la gestion d'un projet Qt. Son éditeur de texte offre les principales fonctions que sont la coloration syntaxique, le complètement, l'indentation, etc... **Qt Creator** intègre en son sein les outils **Qt Designer** et Qt Assistant. Il intègre aussi un mode débuggage.

Qt Designer est un logiciel qui permet de créer des interfaces graphiques Qt dans un environnement convivial. L'utilisateur, par glisser-déposer, place les composants d'interface graphique et y règle leurs propriétés facilement. Les fichiers d'interface graphique sont formatés en XML et portent l'extension `.ui`. Lors de la compilation, un fichier d'interface graphique est converti en classe C++ par l'utilitaire `uic`.

Travail pratique

Objectifs

Les objectifs de ce TP sont :

- mise en oeuvre de **Qt Creator** / **QtDesigner**
- utilisation de quelques *widgets*
- mise en oeuvre du mécanisme *signal/slot*

Mise en situation

Dans ce TP, on vous propose de développer votre première application graphique avec **QtCreator** / **QtDesigner**.

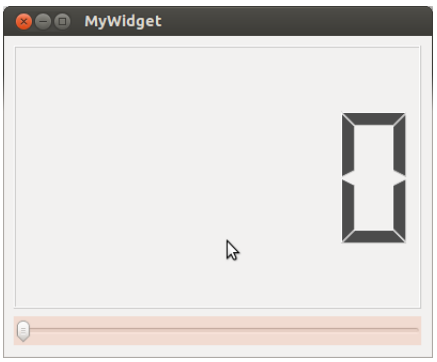
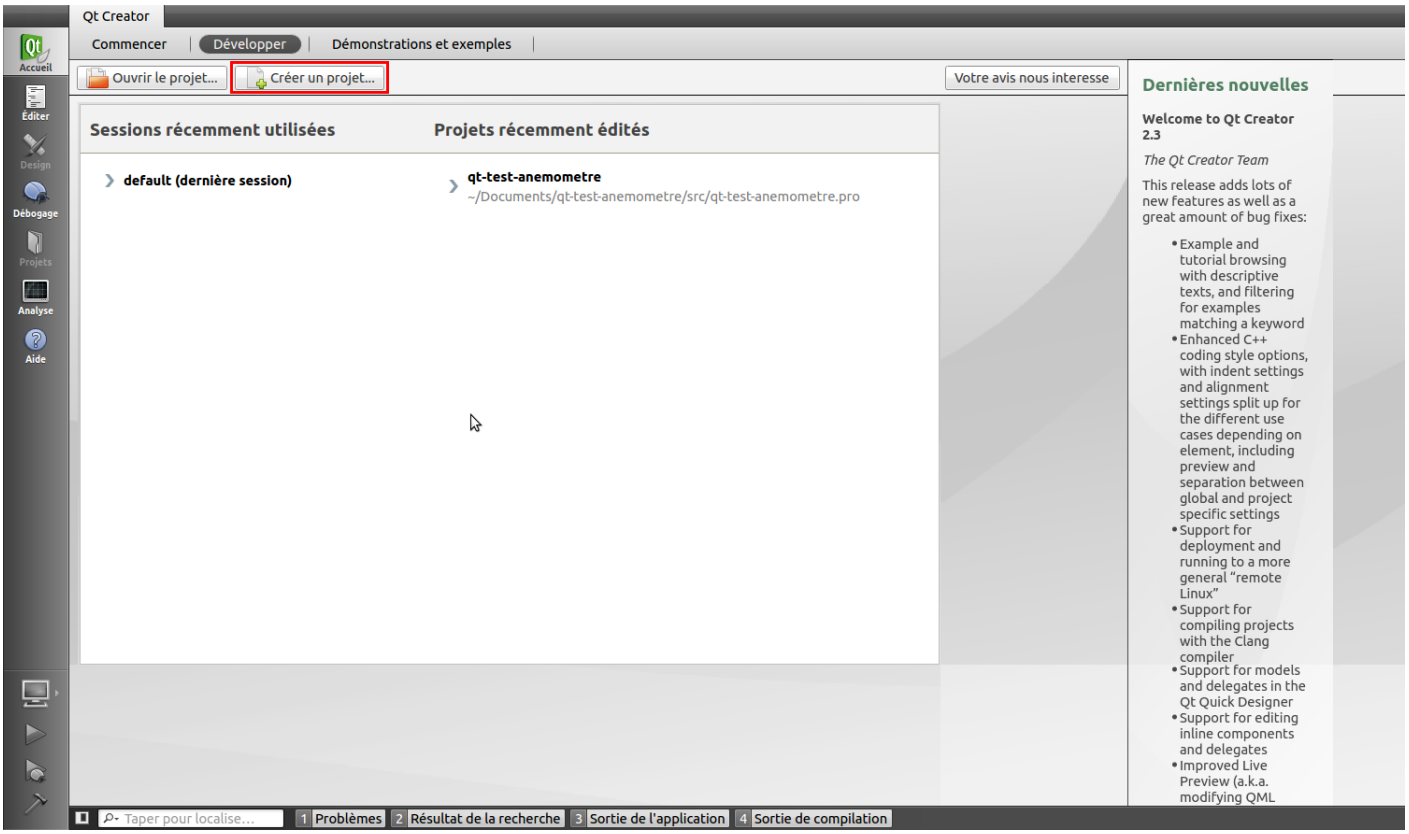


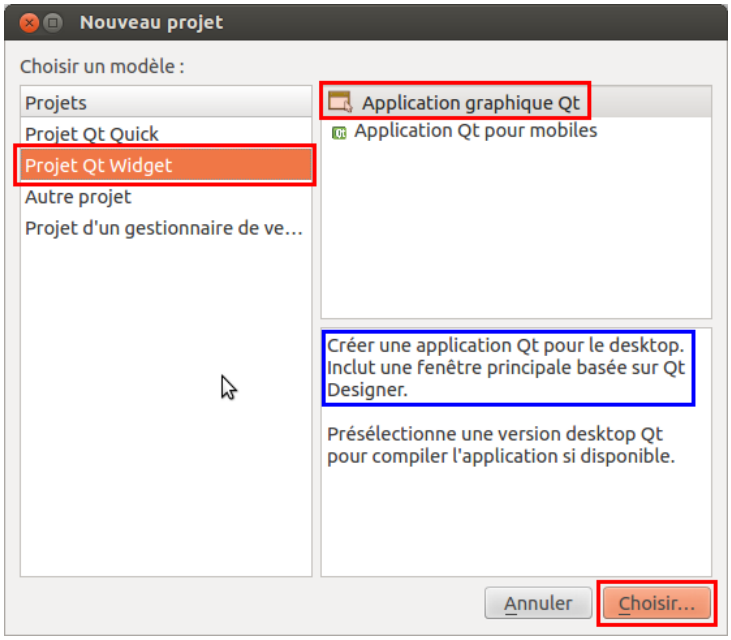
FIGURE *1 – Un sélecteur de nombre*

Créer un projet Qt

Étape 1. Lancer **QtCreator** et "Créer un projet ..."

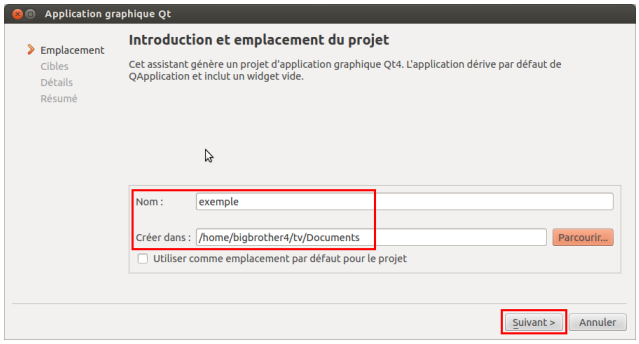


Étape 2. On va ici choisir de développer une application Qt basée sur des *widgets* pour un PC (*desktop*).

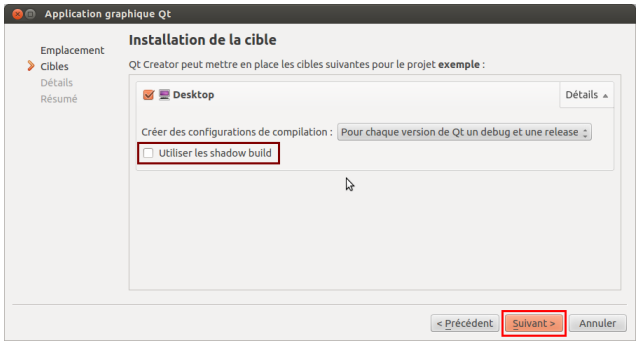


L'assistant va vous guider en **4 étapes** pour créer un **squelette** d'application Qt.

Étape 3. Donner un nom à votre projet et choisissez un emplacement ...



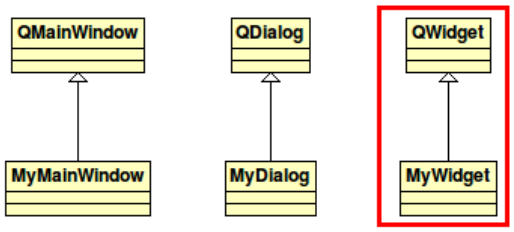
Étape 4. Choix de l'installation de la cible ...



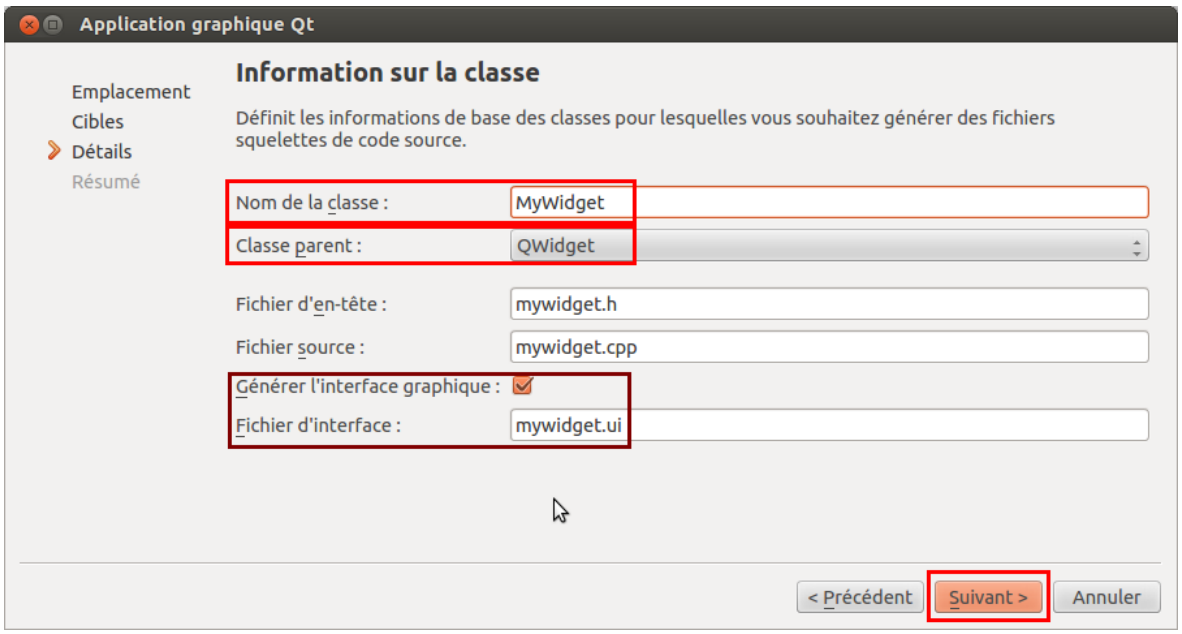
L'option "shadow build" : Qt propose maintenant de séparer le dossier contenant vos sources des dossiers de fabrication (*build*). Le (ou les) dossier(s) de fabrication contiennent des fichiers qui sont reconstruits à chaque fabrication (fichiers objets, fichiers issus du `moc` et de `uic`, exécutable, ...). Ce n'est pas utile ici pour un TP.

Étape 5. Donner un nom à votre classe principale et choisissez la classe de base (`QMainWindow` ou `QDialog` ou `QWidget`). Ces informations vont être utilisées par Qt pour générer les squelettes de votre application.

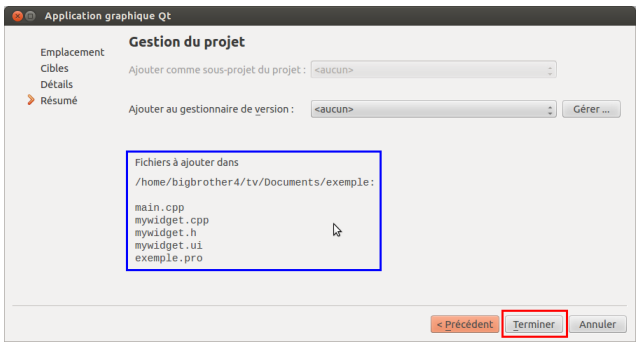
Rappels : il y a trois façons possibles pour créer sa propre application GUI avec Qt

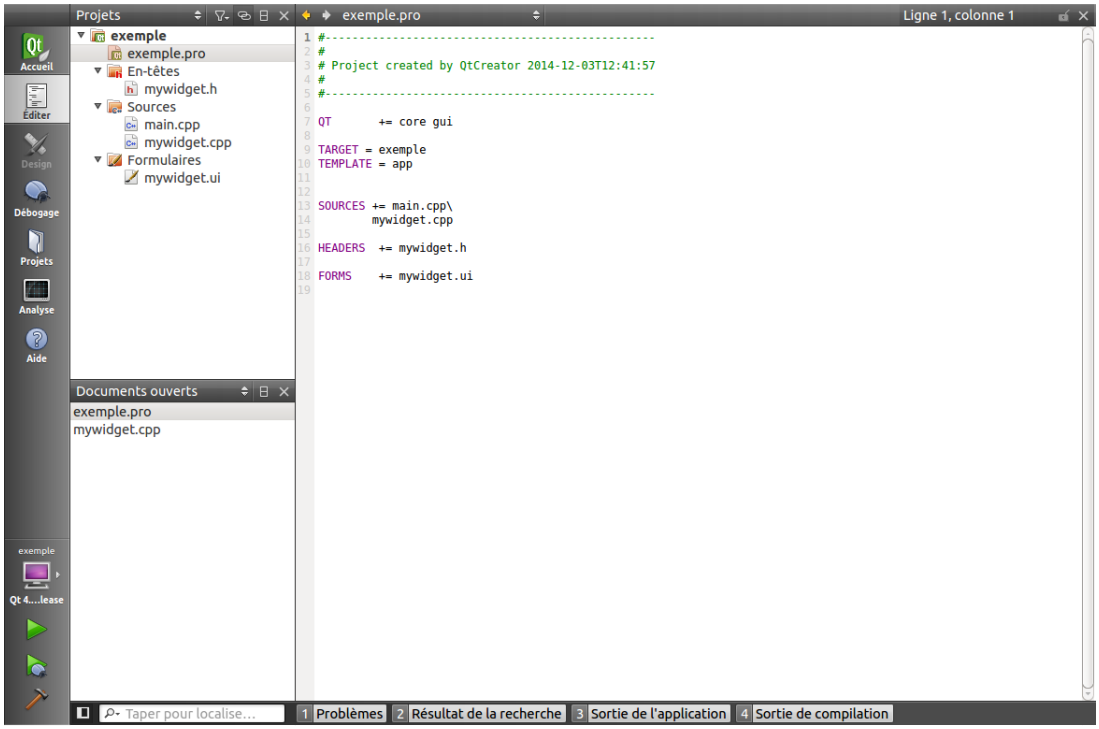


On hérite d'une classe Qt. Le choix de la classe mère se fera en fonction des besoins. Ici, on va choisir `QWidget`.

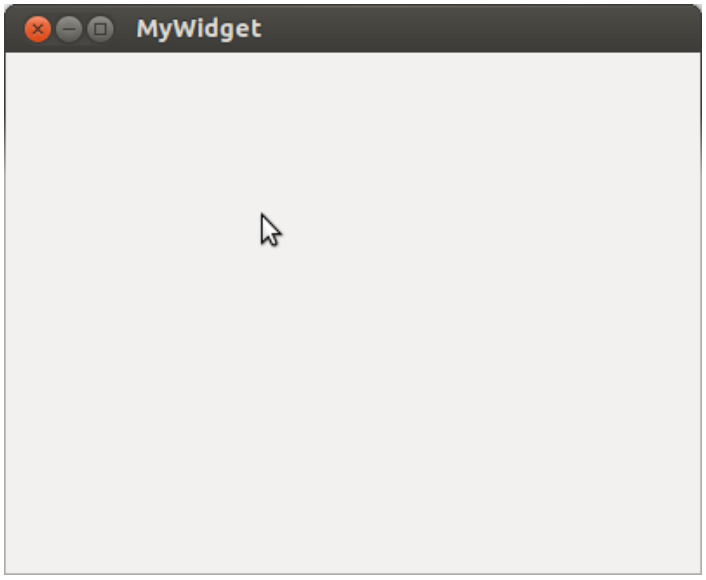


Étape 6. Terminer l'assistant de création de projet.

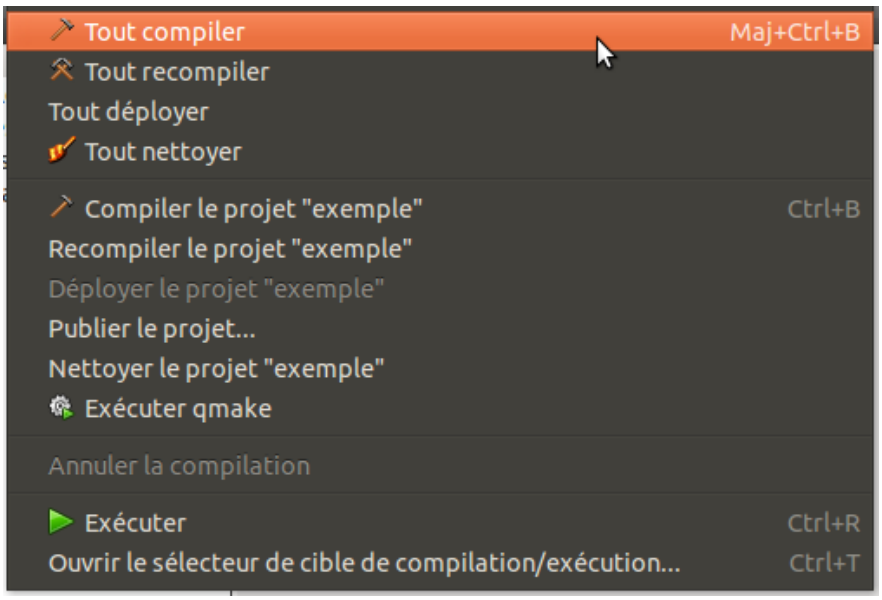




Étape 7. Fabriquer et tester le squelette.

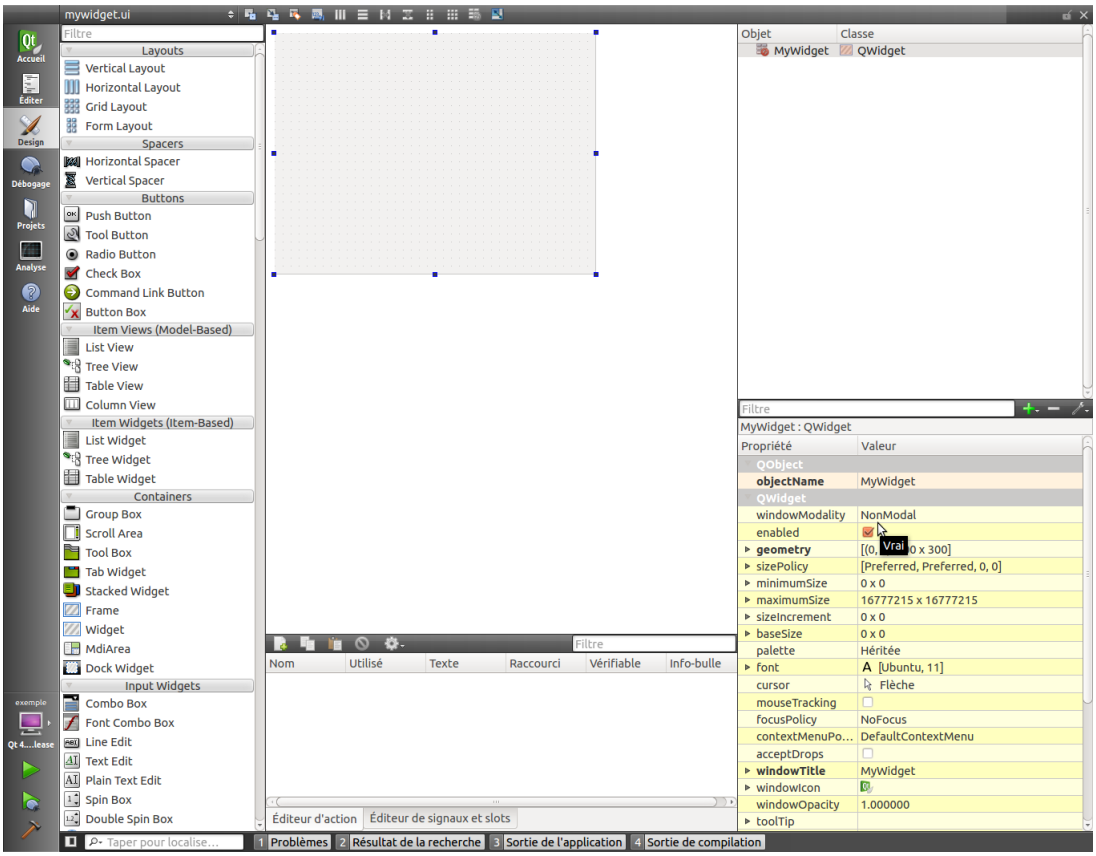


Vous pouvez compiler et lancer l'application directement avec le raccourci clavier `Ctrl-R` (la petite flèche verte). Les autres possibilités sont accessibles dans le menu **Compiler**.

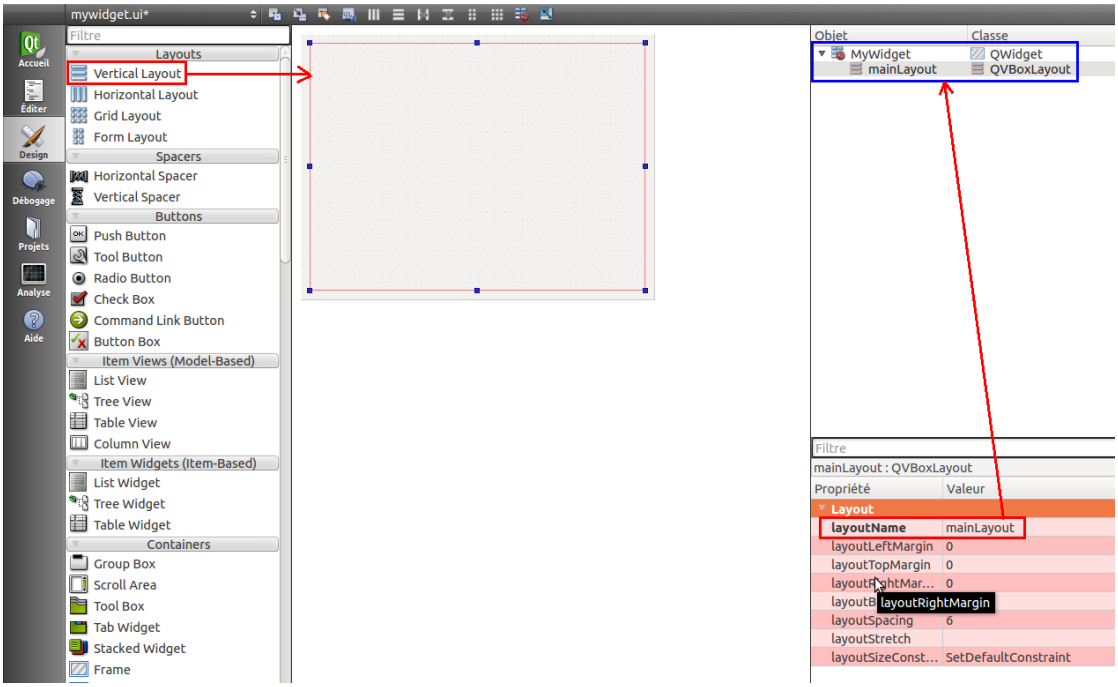


Qt Designer

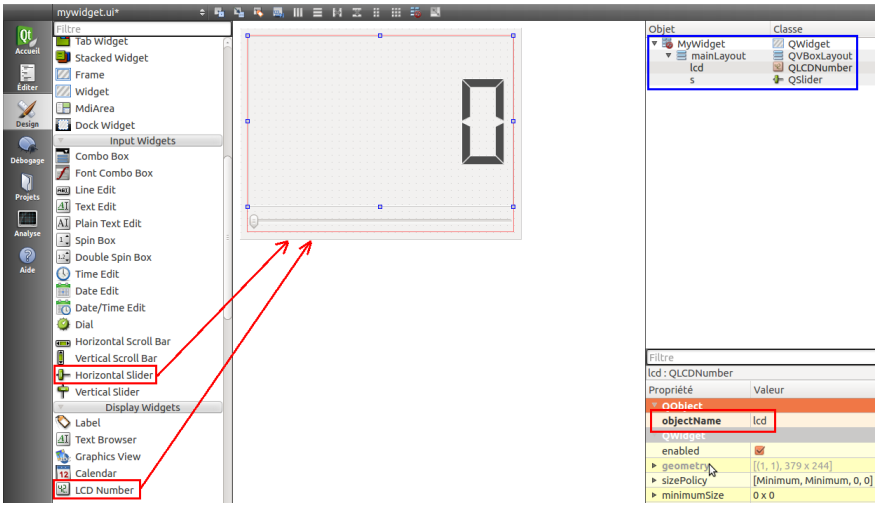
Étape 8. Lancer **Qt Designer**. Il vous suffit de double-cliquer sur votre *form* `mywidget.ui`.



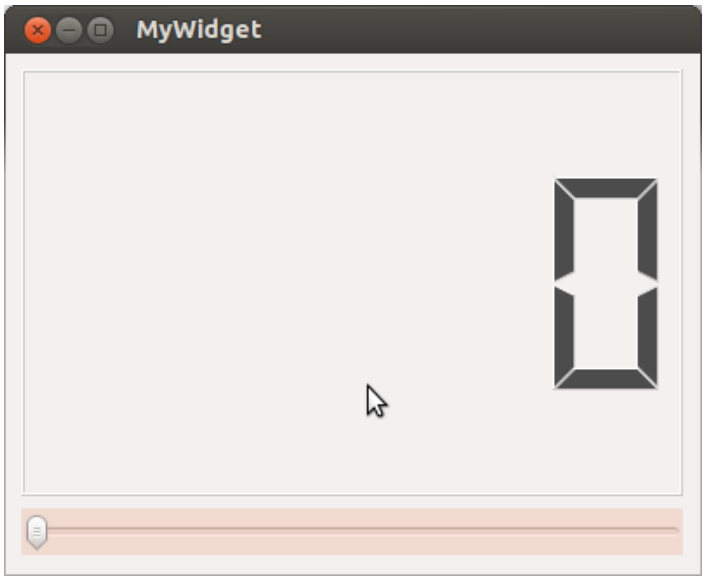
Étape 9. Ajouter un *Vertical Layout*. Renommez-le en `mainLayout`.



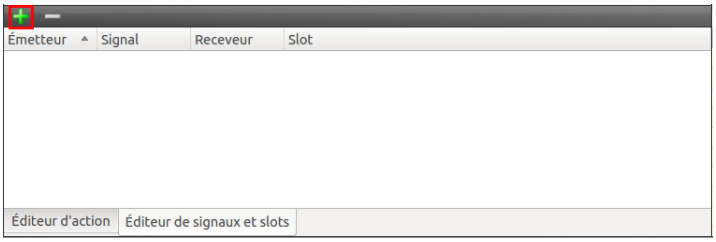
Étape 10. Ajouter les deux *widgets* : un *Vertical Slider* et un *LCD Number*. Renommez-les `s` et `lcd`.

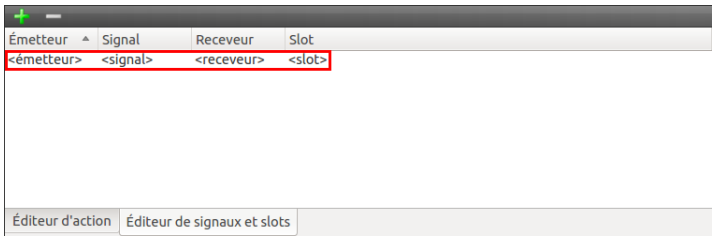


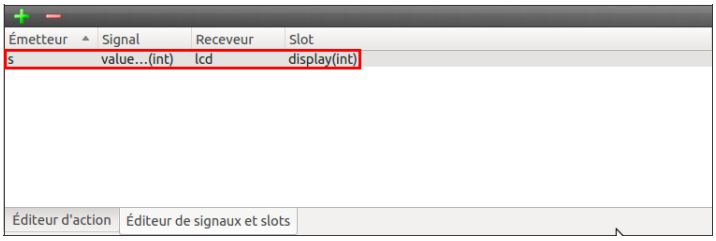
Étape 11. Fabriquer et tester l'application.



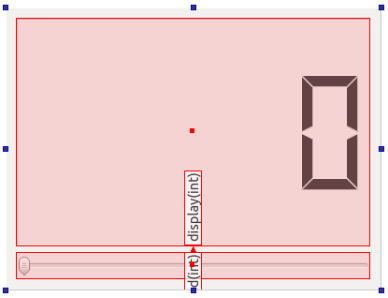
Étape 12. Cliquer sur l'onglet "Éditeur de signaux et slots". On va ajouter une connexion *signal/slot* en cliquant sur "+" puis cliquez et choisissez successivement : l'émetteur, le signal, le receveur et le slot



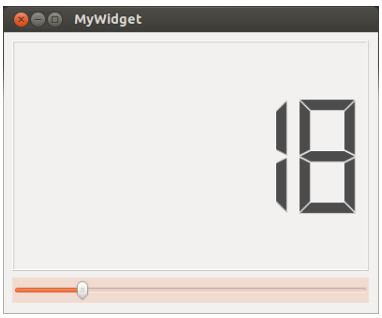




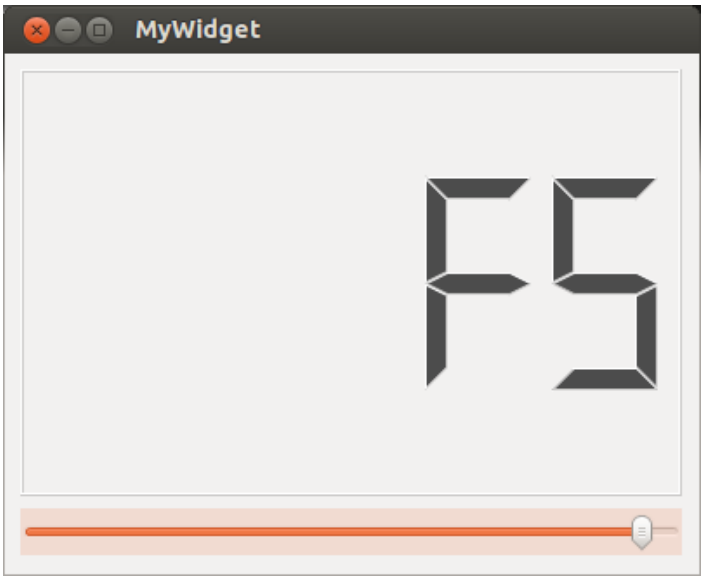
Vous pouvez visualiser vos connexions *signal/slot* avec **F4**.



Étape 13. Fabriquer et tester l'application.



Question 1. En utilisant seulement **Qt Designer**, on vous demande de modifier l'application pour réaliser un **sélecteur de nombre héxadécimal (de 0 à 255)**.



Vous devez modifier une propriété de l'objet `lcd` (`QLCDNumber`) et une propriété de l'objet `s` (`QSlider`).

Utiliser le code généré

Étape 14. Accéder au code généré par **Qt Designer** et `uic`.

Le fichier d'interface graphique est formaté en XML et porte l'extension `.ui`. En voici un extrait :

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MyWidget</class>
 <widget class="QWidget" name="MyWidget">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>400</width>
    <height>300</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>MyWidget</string>
  </property>
  <widget class="QWidget" name="verticalLayoutWidget">
   <property name="geometry">
    <rect>
     <x>9</x>
     <y>9</y>
     <width>381</width>
     <height>281</height>
    </rect>
   </property>
```

```
   <layout class="QVBoxLayout" name="mainLayout">
    <item>
     <widget class="QLCDNumber" name="lcd">
      <property name="mode">
       <enum>QLCDNumber::Hex</enum>
      </property>
      <property name="segmentStyle">
       <enum>QLCDNumber::Filled</enum>
      </property>
     </widget>
    </item>
    <item>
     <widget class="QSlider" name="s">
      <property name="maximum">
       <number>255</number>
      </property>
      <property name="orientation">
       <enum>Qt::Horizontal</enum>
      </property>
     </widget>
    </item>
   </layout>
  </widget>
 </widget>
...
```

Code 1 – mywidget.ui

Lors de la compilation, le fichier d'interface graphique `.ui` est converti en classe C++ par l'utilitaire `uic` pour produire un fichier `.h` contenant le code C++.

```
/********************************************************************************
** Form generated from reading UI file 'mywidget.ui'
**
** Created: Wed Dec 3 15:27:31 2014
**      by: Qt User Interface Compiler version 4.8.1
**
** WARNING! All changes made in this file will be lost when recompiling UI file!
********************************************************************************/

#ifndef UI_MYWIDGET_H
#define UI_MYWIDGET_H

#include <QtCore/QVariant>
#include <QtGui/QAction>
#include <QtGui/QApplication>
#include <QtGui/QButtonGroup>
#include <QtGui/QHeaderView>
#include <QtGui/QLCDNumber>
#include <QtGui/QSlider>
#include <QtGui/QVBoxLayout>
#include <QtGui/QWidget>

QT_BEGIN_NAMESPACE
```



```
class Ui_MyWidget {
public:
    QWidget *verticalLayoutWidget;
    QVBoxLayout *mainLayout;
    QLCDNumber *lcd;
    QSlider *s;

    void setupUi(QWidget *MyWidget) {
        if (MyWidget->objectName().isEmpty())
            MyWidget->setObjectName(QString::fromUtf8("MyWidget"));
        MyWidget->resize(400, 300);
        verticalLayoutWidget = new QWidget(MyWidget);
        verticalLayoutWidget->setObjectName(QString::fromUtf8("verticalLayoutWidget"));
        verticalLayoutWidget->setGeometry(QRect(9, 9, 381, 281));
        mainLayout = new QVBoxLayout(verticalLayoutWidget);
        mainLayout->setSpacing(6);
        mainLayout->setContentsMargins(11, 11, 11, 11);
        mainLayout->setObjectName(QString::fromUtf8("mainLayout"));
        mainLayout->setContentsMargins(0, 0, 0, 0);
        lcd = new QLCDNumber(verticalLayoutWidget);
        lcd->setObjectName(QString::fromUtf8("lcd"));
        lcd->setMode(QLCDNumber::Hex);
        lcd->setSegmentStyle(QLCDNumber::Filled);

        mainLayout->addWidget(lcd);

        s = new QSlider(verticalLayoutWidget);
        s->setObjectName(QString::fromUtf8("s"));
        s->setMaximum(255);
        s->setOrientation(Qt::Horizontal);

        mainLayout->addWidget(s);

        retranslateUi(MyWidget);
        QObject::connect(s, SIGNAL(valueChanged(int)), lcd, SLOT(display(int)));

        QMetaObject::connectSlotsByName(MyWidget);
    } // setupUi

    void retranslateUi(QWidget *MyWidget) {
        MyWidget->setWindowTitle(QApplication::translate("MyWidget", "MyWidget", 0,
            QApplication::UnicodeUTF8));
    } // retranslateUi

};

namespace Ui {
    class MyWidget: public Ui_MyWidget {};
} // namespace Ui

QT_END_NAMESPACE

#endif // UI_MYWIDGET_H
```

Étape 15. Accéder aux objets *widgets* à partir du code source.

Le squelette de la classe `MyWidget` a été généré automatiquement par l'assistant de Qt. Celui-ci a intégré la classe `UI` générée par **Qt Designer**.

```
#ifndef MYWIDGET_H
#define MYWIDGET_H

#include <QWidget>

namespace Ui {
class MyWidget;
}

class MyWidget : public QWidget
{
    Q_OBJECT

public:
    explicit MyWidget(QWidget *parent = 0);
    ~MyWidget();

private:
    Ui::MyWidget *ui; // un pointeur sur la classe générée par uic et créée avec Qt Designer
};

#endif // MYWIDGET_H
```

Code 3 – mywidget.h

```
#include "mywidget.h"
#include "ui_mywidget.h"

MyWidget::MyWidget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::MyWidget) // instancie ma fenêtre ui
{
    ui->setupUi(this); // initialise ma fenêtre ui
}

MyWidget::~MyWidget()
{
    delete ui; // libère ma fenêtre ui
}
```

Code 4 – mywidget.cpp

Pour accéder aux objets *widgets*, il suffit donc de "passer" par l'objet `ui` (qui représente l'objet interface graphique) soit :

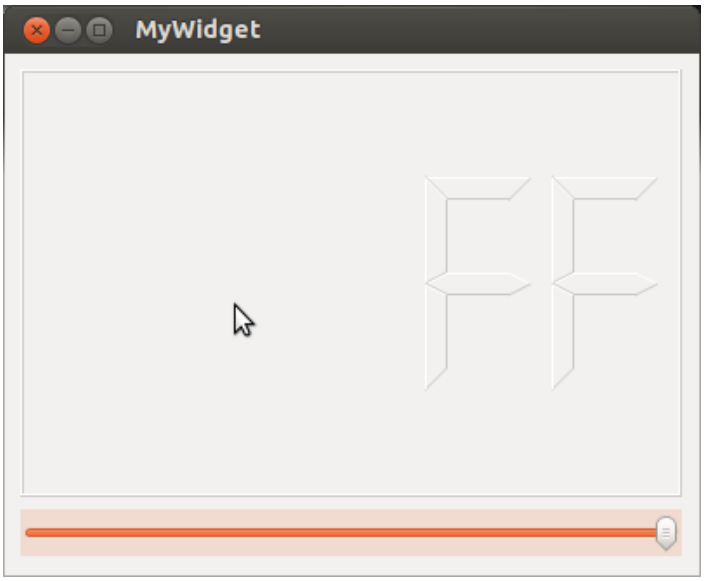
`ui->lcd` pour accéder au widget `lcd` de type `QLCDNumber`
`ui->s` pour accéder au widget `s` de type `QSlider`

Par exemple, si l'on veut modifier le **style de segment** de l'objet `lcd`, on fera :

```
#include "mywidget.h"
#include "ui_mywidget.h"

MyWidget::MyWidget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::MyWidget)
{
    ui->setupUi(this);

    // modifie le style de segment de l'objet lcd
    ui->lcd->setSegmentStyle(QLCDNumber::Outline);
}

MyWidget::~MyWidget()
{
    delete ui;
}
```

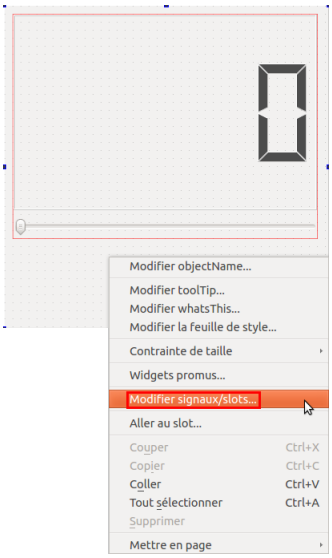
Code 5 – mywidget.cpp

Question 2. Dans le constructeur de la classe `MyWidget`, initialiser l'objet `s` avec la valeur `255` (`0xFF`).

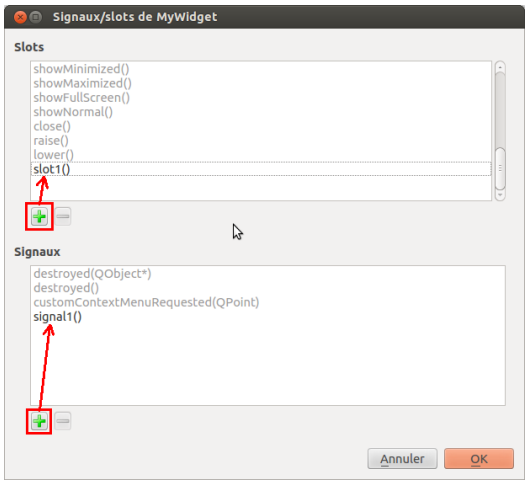


Créer ses propres signaux et slots

Étape 16. Dans **Qt Designer**, cliquez avec le bouton droit sur votre fenêtre :

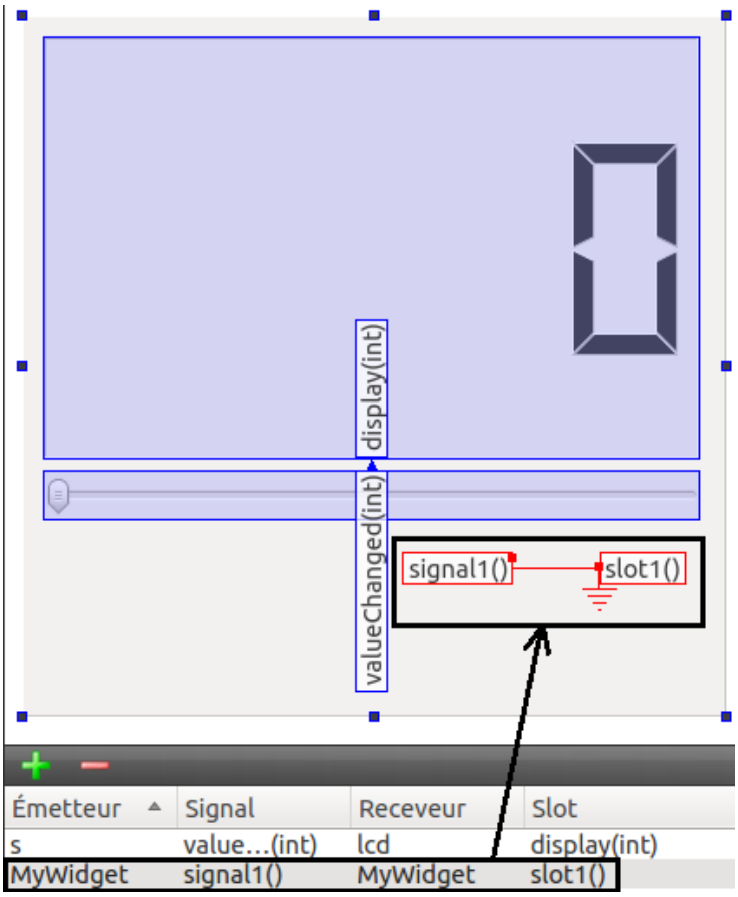


Cliquez sur "Modifier signaux/slots" ...



et ajoutez un slot et un signal

Vous pouvez visualiser la nouvelle connexion *signal/slot* avec **F4**.



Il vous faudra déclarer vos signaux et slots puis définir vos slots dans la classe `MyWidget`.

Déclaration du *signal* `signal1()` et du *slot* `slot1()` :

```
#ifndef MYWIDGET_H
#define MYWIDGET_H

#include <QWidget>
#include "ui_mywidget.h"

class MyWidget : public QWidget, public Ui::MyWidget
{
    Q_OBJECT

public:
    explicit MyWidget(QWidget *parent = 0);
    ~MyWidget();

private:

signals:
    void signal1();

public slots:
    void slot1();
};

#endif // MYWIDGET_H
```

Et déclaration du *slot* `slot1()` :

```
#include "mywidget.h"
#include <QDebug>

...
void MyWidget::slot1()
{
    qDebug() << "slot1";
}
...
```


Modifier le code généré

La relation entre la classe “Application” `MyWidget` et la classe (G)UI est implémentée par Qt par une **agrégation** : l'application est composée d'une GUI (c'est le membre `ui` dans la classe `MyWidget`).

```
#ifndef MYWIDGET_H
#define MYWIDGET_H

#include <QWidget>

namespace Ui {
class MyWidget;
}

class MyWidget : public QWidget
{
    Q_OBJECT

public:
    explicit MyWidget(QWidget *parent = 0);
    ~MyWidget();

private:
    Ui::MyWidget *ui; // Agrégation (ou composition par pointeur) générée par Qt
};

#endif // MYWIDGET_H
```

Code 8 – mywidget.h

On va modifier cette relation pour mettre en oeuvre un **héritage** entre la classe “Application” `MyWidget` et la classe (G)UI. Pour cela, on va supprimer le membre `ui` et réaliser un héritage multiple :

```
#ifndef MYWIDGET_H
#define MYWIDGET_H

#include <QWidget>
#include "ui_mywidget.h" // <- accès à la déclaration de Ui::MyWidget

class MyWidget : public QWidget, public Ui::MyWidget // <- héritage multiple
{
    Q_OBJECT

public:
    explicit MyWidget(QWidget *parent = 0);
    ~MyWidget();
};

#endif // MYWIDGET_H
```

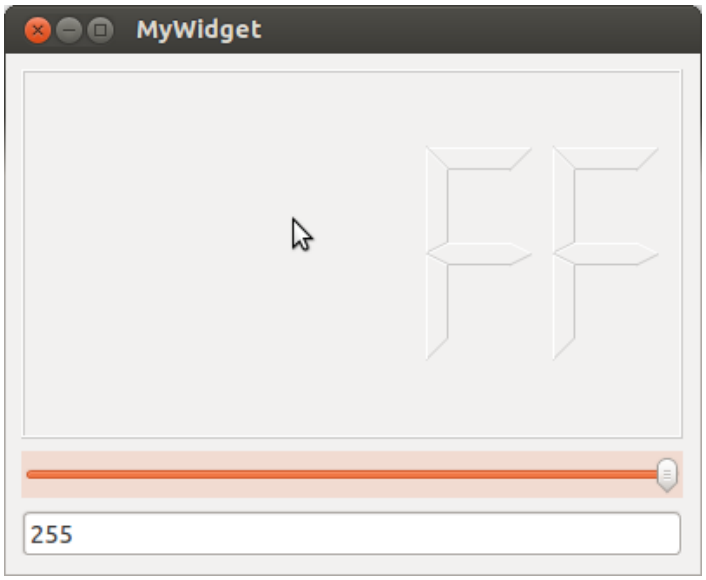
Code 9 – mywidget.h

On peut maintenant supprimer toutes les références à `ui` :

```
#include "mywidget.h"

MyWidget::MyWidget(QWidget *parent) :
    QWidget(parent)
{
    setupUi(this);

    lcd->setSegmentStyle(QLCDNumber::Outline);
    s->setValue(255);
}

MyWidget::~MyWidget()
{
}
```

Code 10 – mywidget.cpp

Question 3. Intégrer maintenant dans votre application un `QLineEdit` qui permet de saisir un nombre manuellement et de le "visualiser" dans le `QLCDNumber` (et le `QSlider` se met aussi à jour). Prévoir aussi le fonctionnement inverse : le nombre sélectionné avec le `QSlider` doit s'afficher dans le `QLineEdit`. La saisie et l'affichage dans le `QLineEdit` se fera en décimal.



Il vous faudra certainement créer vos propres "signaux et slots".

