

# Développement d'applications graphiques en C++ à l'aide de la bibliothèque graphique Qt

Génie Mathématique et Modélisation  
Marinette Bouet & Christophe de Vault

# Références

---

- Site officiel : <http://www.qt.io>
- Téléchargement de Qt Open Source :  
<https://www.qt.io/download-open-source/>
- Un cours sur Qt très détaillé : <http://tvaira.free.fr/>

# Plan du cours

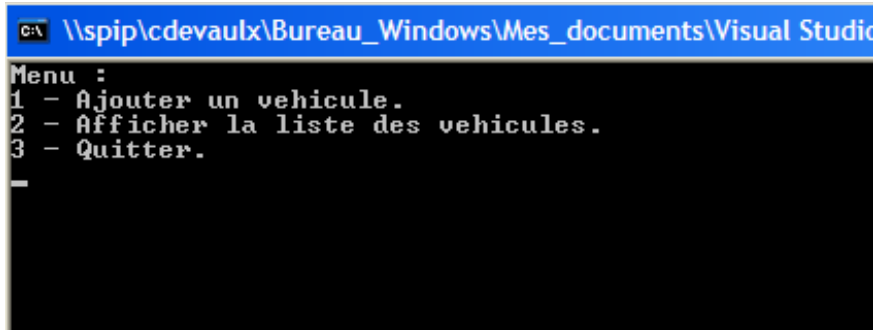
---

- Introduction à la programmation événementielle
- La bibliothèque graphique Qt
- Développement d'une application avec Qt Creator et Qt Designer
- Tracés graphiques 2D avec QPainter

# Introduction à la programmation événementielle

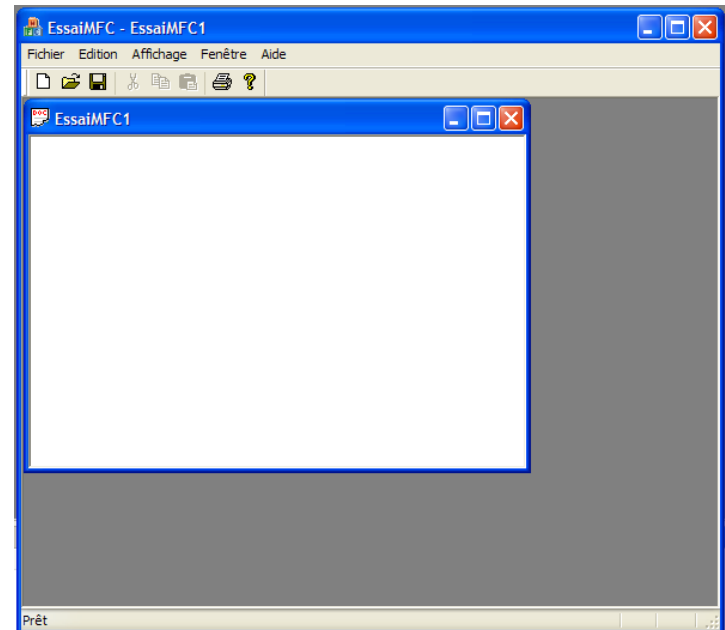
# Introduction

- Actuellement il existe deux grands types d'applications :



```
C:\\\spip\cdevaulx\Bureau_Windows\Mes_documents\Visual Studio
Menu :
1 - Ajouter un vehicule.
2 - Afficher la liste des vehicules.
3 - Quitter.
```

Les applications  
de type console



Les applications  
graphiques

# Introduction

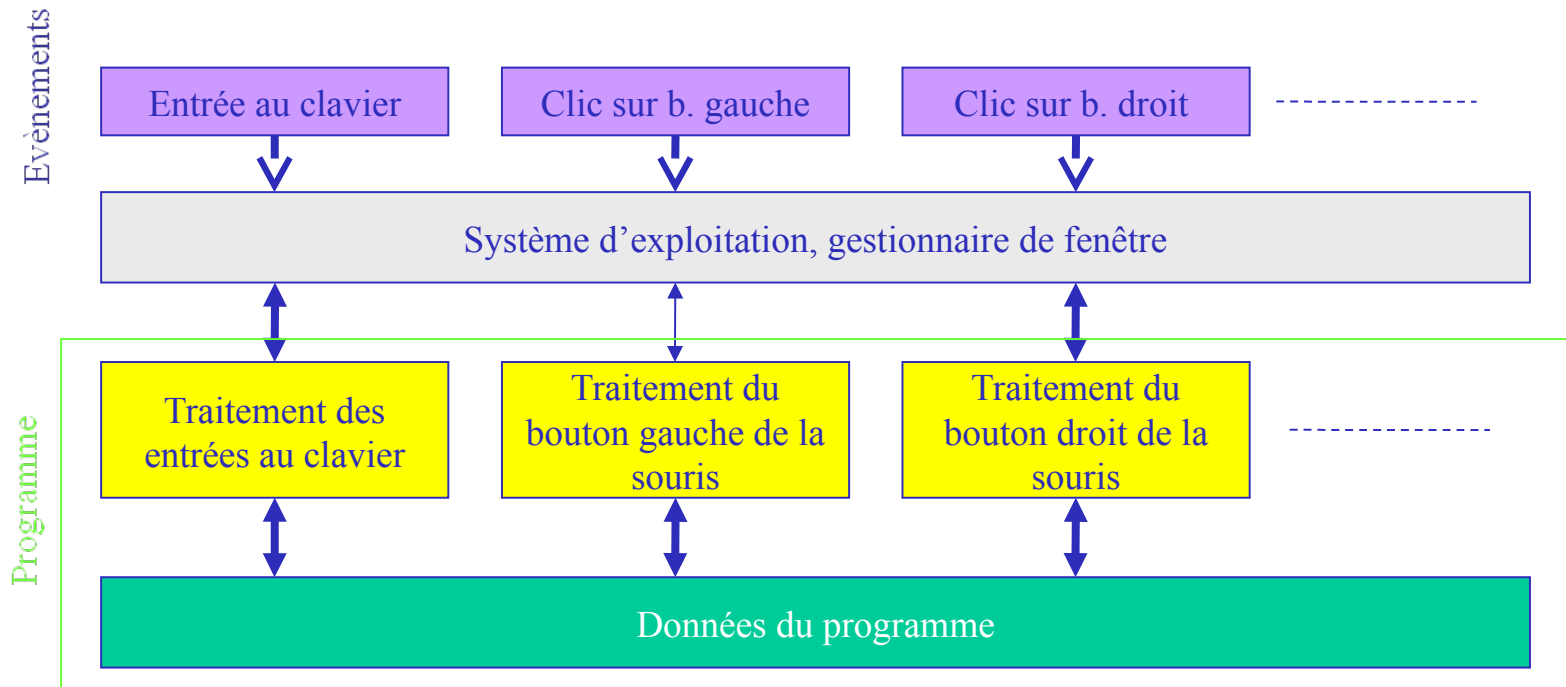
- Ces deux types de programmes sont très différents l'un de l'autre :
  - une application de type console est généralement assez simple. Elle est exécutée de manière séquentielle à l'intérieur d'un terminal dont elle prends le contrôle. Son interface est textuelle ;
  - une application graphique est beaucoup plus complexe :
    - tout d'abord elle doit posséder une interface graphique (GUI : Graphical User Interface) ;
    - deuxièmement, son code n'est pas exécuté de manière séquentielle mais en réponse aux actions de l'utilisateur.

# Programmation événementielle

- Modèle de programmation utilisé pour développer des applications graphique pour Windows, Mac OS X ou encore pour Linux.
- Le nom de ce modèle provient du fait que les programmes ne sont pas exécutés de manière séquentielle mais de manière fragmenté en réponse à des événements. Ces événements sont des signaux qui sont déclenchés par les différentes actions de l'utilisateur (clic souris, appui sur une touche du clavier...) ou par les composants matériel de l'ordinateur (tic timer...).

# Programmation événementielle

- La structure des programmes développés à l'aide de ce modèle de programmation est donc très différente de celle d'un programme classique :





# La bibliothèque graphique Qt

# Généralités

- Qt est une bibliothèque graphique orientée objet (codée en C++) développée par Nokia.
- Cette bibliothèque est utilisée par de nombreuses entreprises (Nokia, Adobe, etc). Elle est par exemple à la base de l'environnement graphique KDE.
- Elle est disponible sous Windows, Mac OS X, Linux...

# Généralités

---

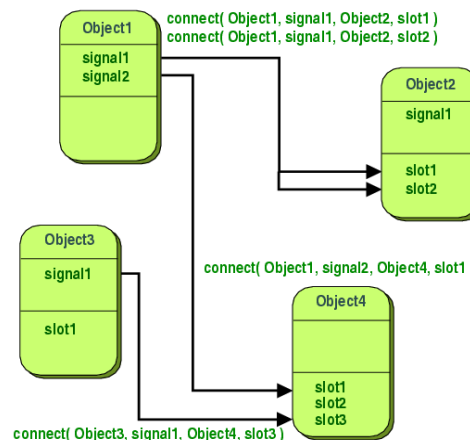
- Qt est disponible sous deux types de licences :
  - Commerciale
  - Open source (GNU LGPL v. 3)

# La classe QObject

- La classe QObject est le coeur du modèle objet de Qt.
- La plupart des objets que l'on peut manipuler grâce à cette librairie héritent de la classe QObject.

# Signaux / slots

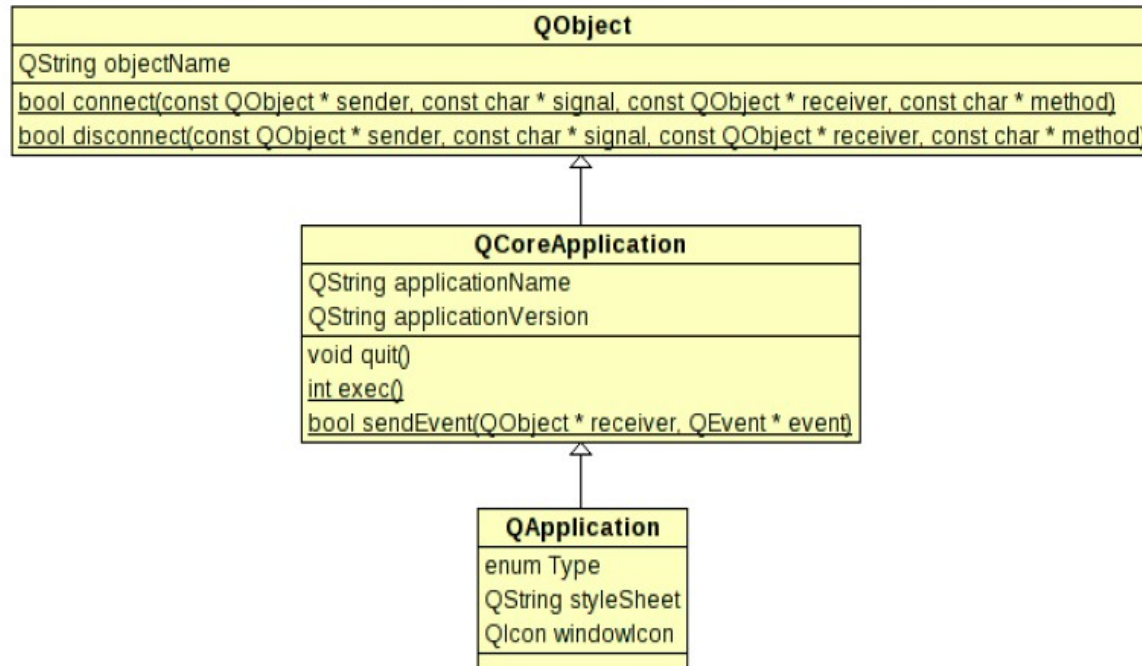
- L'autre élément central du modèle objet de la bibliothèque Qt est son mécanisme de communication entre les objets basé sur les notions de signaux et de slots.
- Lorsqu'ils changent d'état, les composants de Qt émettent des signaux (événements). Ces signaux peuvent être connectés à des slots (méthodes, gestionnaire d'évènement) pour être traités. :



# Signaux / slots

- Modularités des signaux :
  - Un signal peut être connecté à plusieurs slots ;
  - Plusieurs signaux peuvent être connectés à un même slot ;
- Remarques :
  - Le composant qui émet un signal n'a aucune idée de l'identité du composant qui va le recevoir ;
  - De même, le composant qui reçoit un signal ne connaît pas l'identité du composant qui l'a émis.

# La classe QApplication



- Une instance de la classe **QApplication** doit être impérativement créée avant tout autre objet graphique.

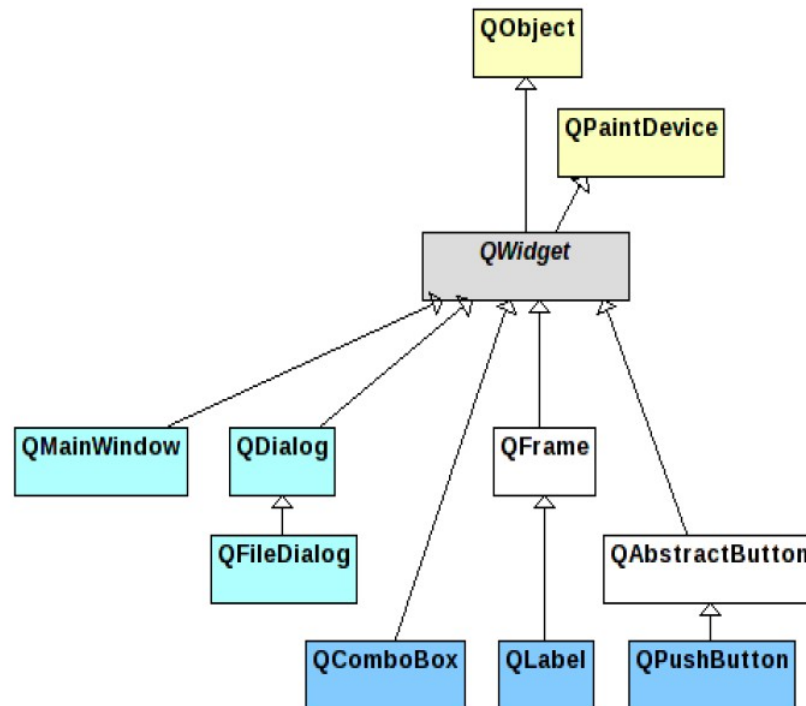
# La classe QApplication

- L'instance de la classe QApplication, initialise l'application et reçoit tous les arguments transmis à la fonction main (argc, argv).
- Elle s'occupe de la gestion des événements (réception + transmission aux widgets concernés).
- Elle est toujours accessible grâce au pointeur global nommé qApp.
- Les applications doivent se terminer proprement en appelant QApplication::quit(). Cette méthode est appelée automatiquement lors de la fermeture du dernier widget.

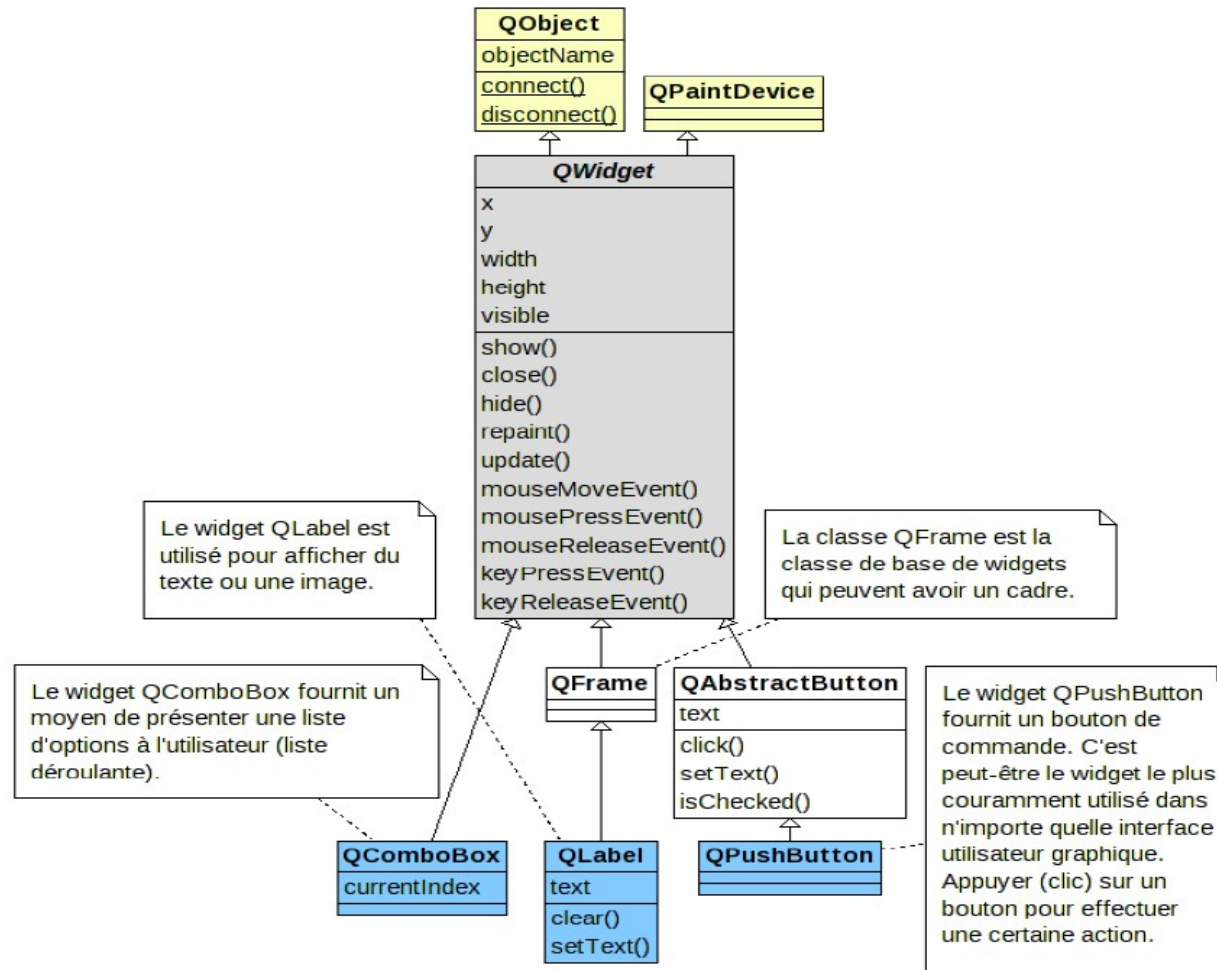


# La classe QWidget

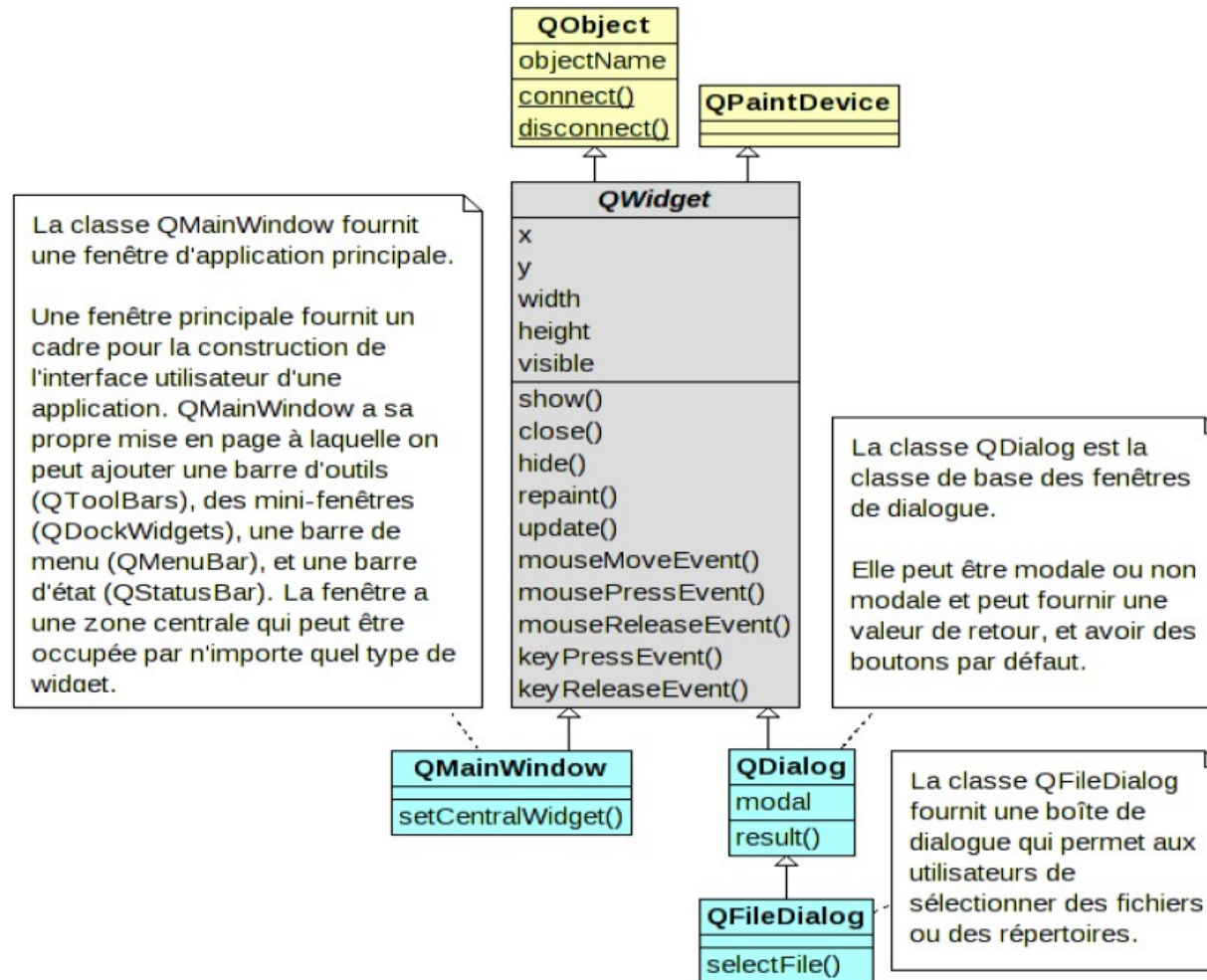
- La classe QWidget est la classe de base de tous les composants graphiques de Qt :



# Les composants élémentaires

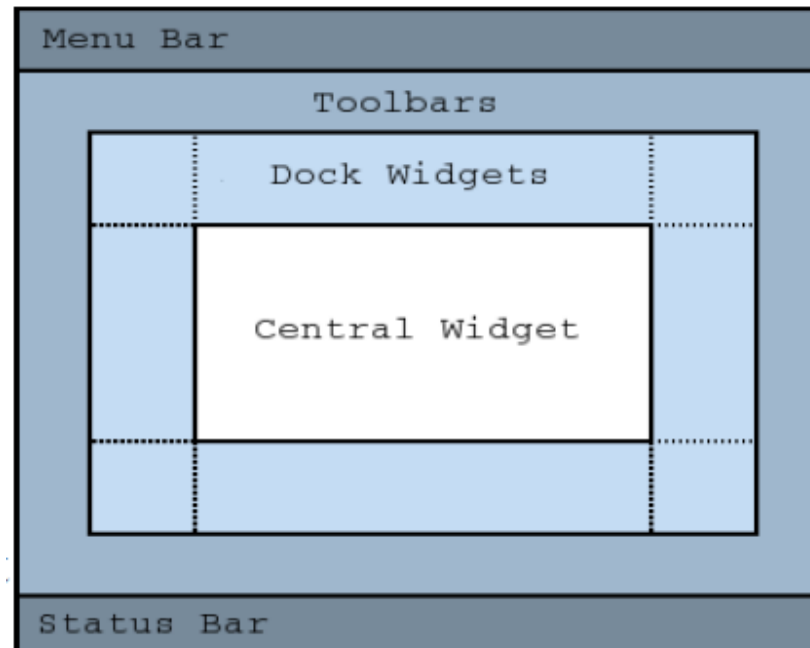


# Les fenêtres



# La classe QMainWindow

- La classe QMainWindow sert à réaliser la fenêtre principale d'une application.
- Cette dernière possède une structure complexe :



# La classe QDockWidget

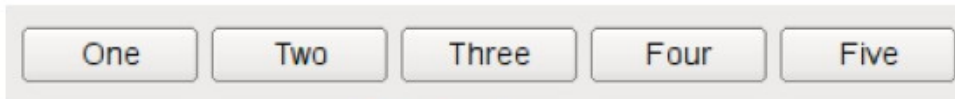
- La classe QDockWidget permet de créer des palettes d'outils qui peuvent être fixées dans la fenêtre principale ou "flotter" de manière indépendante sur le bureau.
- Beaucoup d'applications connues utilisent ces palettes d'outils : Photoshop, Qt Designer, OpenOffice, Code::Blocks , ...

# Les Layout

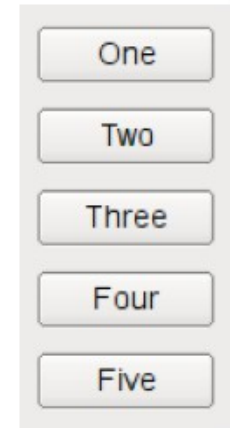
- Tout comme Java, Qt propose des gestionnaires de positionnement (layout) pour faciliter l'organisation et le positionnement des composants d'une fenêtre ou d'un conteneur.
- Pour appliquer un gestionnaire de positionnement à un widget il faut utiliser la fonction : `QWidget::setLayout()`.

# Les Layout

- Exemples :



**QHBoxLayout**

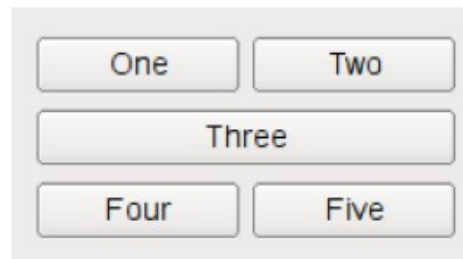


**QVBoxLayout**

**QFormLayout**



**QGridLayout**



# La classe QString

- Classe utilisée par Qt pour manipuler les chaînes de caractères.
- Conversion QString / std::string

```
QString qtChaine("Coucou");  
string stdChaine = qtChaine.toStdString();  
cout << stdChaine << endl;
```

QString → std::string

std::string → QString

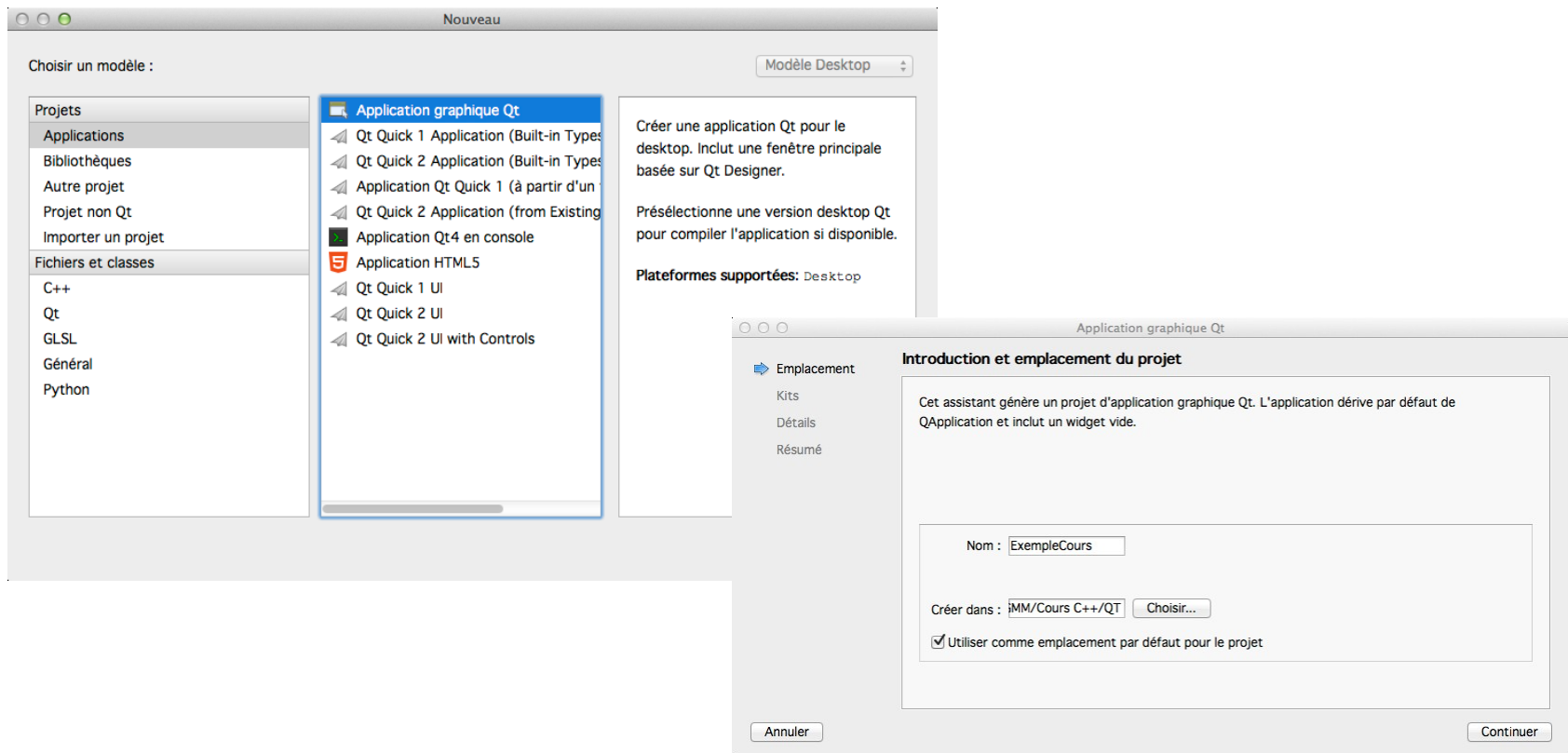
```
string stdChaine = "Bonjour";  
QString qtChaine;  
qtChaine.fromStdString(stdChaine);
```



# Développement d'une application avec Qt Creator et Qt Designer

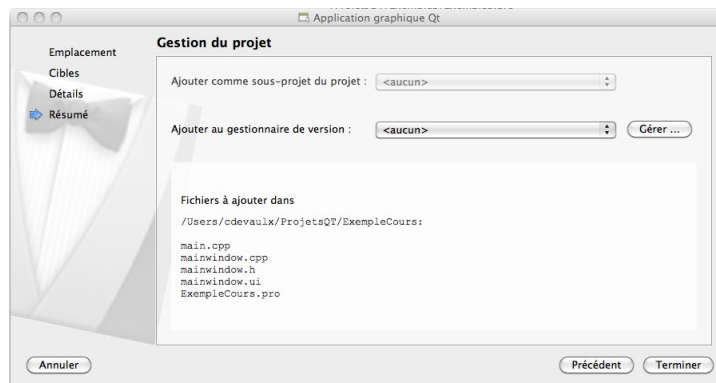
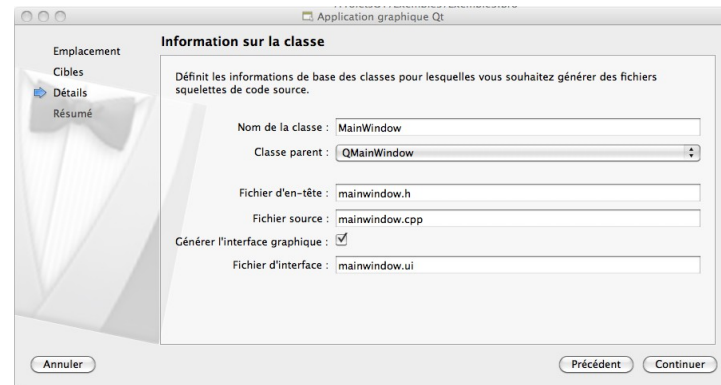
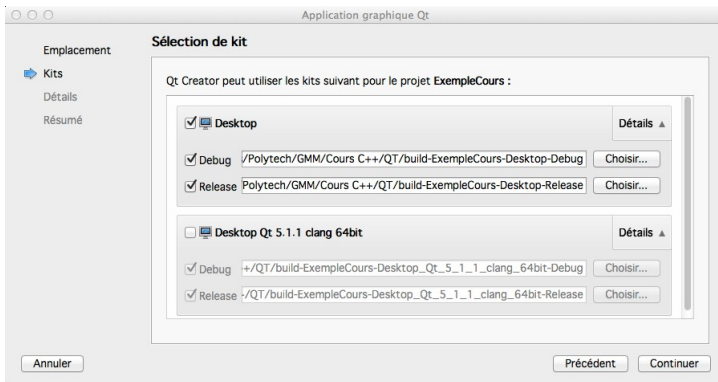
# Création du projet

- Fichier → Nouveau fichier ou projet

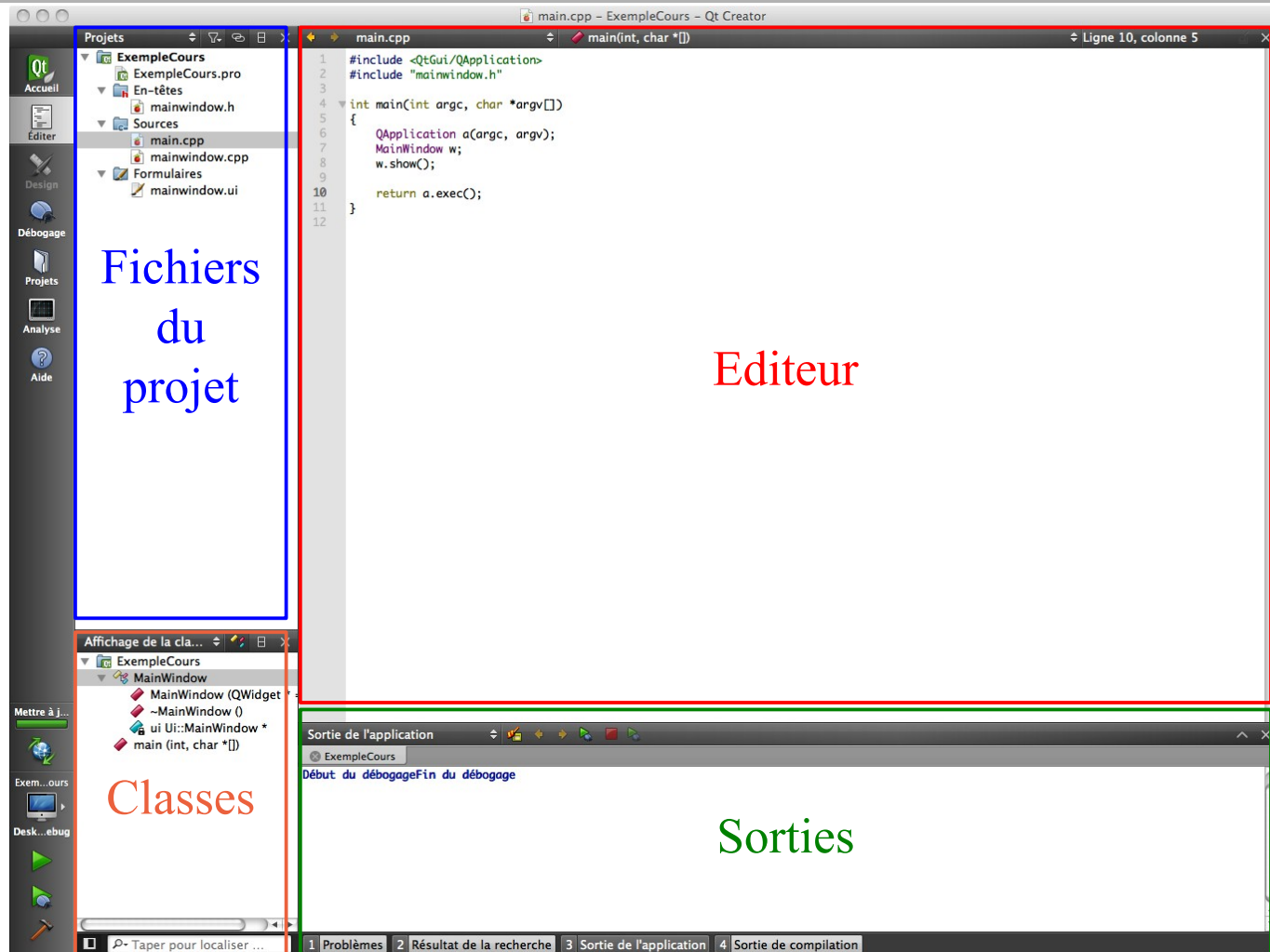


# Création du projet

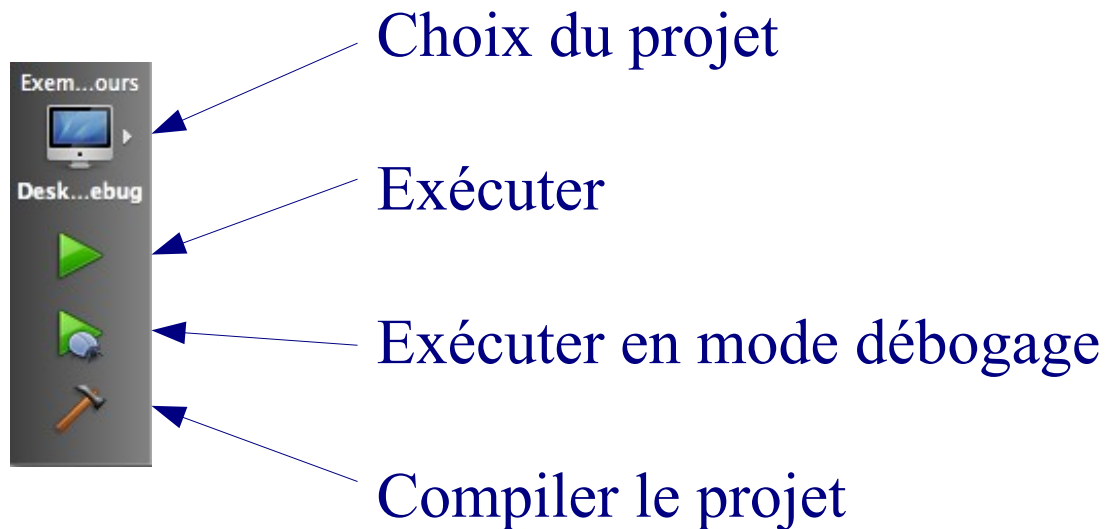
- Suite :



# Interface de Q Creator



# Commandes de compilation, débogage et d'exécution



# Fenêtre de débogage

The screenshot shows the Qt Creator IDE with the following components:

- Project Explorer:** Shows the project structure for 'ExempleCours', including 'main.cpp' and 'mainwindow.cpp'.
- Editor:** Displays the code in 'main.cpp'. The code is as follows:

```
1 #include <QtGui/QApplication>
2 #include "mainwindow.h"
3
4 int main(int argc, char *argv[])
5 {
6     QApplication a(argc, argv);
7     MainWindow w;
8     w.show();
9
10    return a.exec();
11 }
12
```
- Debugger:** The 'Threads' window shows the application is stopped at a breakpoint in the 'main' function at line 8. The 'Variables' window shows the values of 'argc' (1) and 'argv' (0x7ffffb7f7...).
- Stack:** The 'Stack' window shows the call stack, with the top frame being 'main (int, char \*)' at line 8.
- Bottom Panel:** The 'Pile' (Stack) and 'Inspecteur QML' (QML Inspector) are visible.

# Squelette de l'application

- mainwindow.ui : fichier xml décrivant la fenêtre principale générée par Qt Designer (taille, position et nom de la fenêtre ; liste et caractéristiques des widgets qui ont été ajoutés à la fenêtre principale).

<pre>&lt;?xml version="1.0" encoding="UTF-8"?&gt; &lt;ui version="4.0"&gt;   &lt;class&gt;MainWindow&lt;/class&gt;   &lt;widget class="QMainWindow" name="MainWindow"&gt;     &lt;property name="geometry"&gt;       &lt;rect&gt;         &lt;x&gt;0&lt;/x&gt;         &lt;y&gt;0&lt;/y&gt;         &lt;width&gt;400&lt;/width&gt;         &lt;height&gt;300&lt;/height&gt;       &lt;/rect&gt;     &lt;/property&gt;     &lt;property name="windowTitle"&gt;       &lt;string&gt;MainWindow&lt;/string&gt;     &lt;/property&gt;     &lt;widget class="QWidget" name="centralWidget"/&gt;     &lt;widget class="QMenuBar" name="menuBar"&gt;       &lt;property name="geometry"&gt;         &lt;rect&gt;           &lt;x&gt;0&lt;/x&gt;           &lt;y&gt;0&lt;/y&gt;           &lt;width&gt;400&lt;/width&gt;           &lt;height&gt;22&lt;/height&gt;         &lt;/rect&gt;       &lt;/property&gt;     &lt;/widget&gt;</pre>	<pre>&lt;widget class="QToolBar" name="mainToolBar"&gt;   &lt;attribute name="toolBarArea"&gt;     &lt;enum&gt;TopToolBarArea&lt;/enum&gt;   &lt;/attribute&gt;   &lt;attribute name="toolBarBreak"&gt;     &lt;bool&gt;false&lt;/bool&gt;   &lt;/attribute&gt; &lt;/widget&gt; &lt;widget class="QStatusBar" name="statusBar"/&gt; &lt;/widget&gt; &lt;layoutdefault spacing="6" margin="11"/&gt; &lt;resources/&gt; &lt;connections/&gt; &lt;/ui&gt;</pre>
--	--

# Squelette de l'application

- `ui_mainwindow.h` : fichier généré automatiquement à la compilation par l'utilitaire `uic` à partir du fichier xml précédent. Ce fichier contient le code de la classe `Ui::MainWindow` qui gère l'interface utilisateur de la fenêtre.

```
/* Form generated from reading UI file 'mainwindow.ui'
**
** Created by: Qt User Interface Compiler version
** 5.7.0
**
** WARNING! All changes made in this file will be lost
** when recompiling UI file!
*****/

#ifndef UI_MAINWINDOW_H
#define UI_MAINWINDOW_H

#include <QtCore/QVariant>
#include <QtWidgets/QAction>
#include <QtWidgets/QApplication>
#include <QtWidgets/QButtonGroup>
#include <QtWidgets/QHeaderView>
#include <QtWidgets/QMainWindow>
#include <QtWidgets/QMenuBar>
#include <QtWidgets/QStatusBar>
#include <QtWidgets/QToolBar>
#include <QtWidgets/QWidget>
```

```
QT_BEGIN_NAMESPACE
```

```
class Ui_MainWindow
```

```
{
public:
```

```
QMenuBar *menuBar;
QToolBar *mainToolBar;
QWidget *centralWidget;
QStatusBar *statusBar;
```

```
void setupUi(QMainWindow *MainWindow)
{
```

```
    if (MainWindow->objectName().isEmpty())
        MainWindow->setObjectName(QStringLiteral("MainWindow"));
    MainWindow->resize(400, 300);
    menuBar = new QMenuBar(MainWindow);
    menuBar->setObjectName(QStringLiteral("menuBar"));
    MainWindow->setMenuBar(menuBar);
    mainToolBar = new QToolBar(MainWindow);
    mainToolBar->setObjectName(QStringLiteral("mainToolBar"));

    MainWindow->addToolBar(mainToolBar);
    centralWidget = new QWidget(MainWindow);
    centralWidget->setObjectName(QStringLiteral("centralWidget"));
    MainWindow->setCentralWidget(centralWidget);
    statusBar = new QStatusBar(MainWindow);
```



# Squelette de l'application

```
        statusBar->setObjectName(QStringLiteral("statusBar"));
        MainWindow->setStatusBar(statusBar);

        retranslateUi(MainWindow);

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

    void retranslateUi(QMainWindow *MainWindow)
    {
        MainWindow->setWindowTitle(QApplication::translate("MainWindow", "MainWindow", 0));
    } // retranslateUi

};

namespace Ui {
    class MainWindow: public Ui_MainWindow {};
} // namespace Ui

QT_END_NAMESPACE

#endif // UI_MAINWINDOW_H
```

# Squelette de l'application

- mainwindow.h

```
#ifndef MAINWINDOW_H  
#define MAINWINDOW_H
```

```
#include <QMainWindow>
```

```
namespace Ui {  
class MainWindow;  
}
```

```
class MainWindow : public QMainWindow  
{
```

```
    Q_OBJECT
```

```
public:
```

```
    explicit MainWindow(QWidget *parent = 0);
```

```
    ~MainWindow();
```

```
private:
```

```
    Ui::MainWindow *ui;
```

```
};
```

```
#endif // MAINWINDOW_H
```

La classe MainWindow hérite de la classe QMainWindow

Macro obligatoire pour tout objet implémentant des signaux ou des slots

Constructeur

Destructeur

Pointeur sur un objet de type Ui::MainWindow. Ce pointeur permet d'accéder aux différents composants graphiques de la fenêtre.

# Squelette de l'application

- mainwindow.cpp

```
#include "mainwindow.h"  
#include "ui_mainwindow.h"
```

```
MainWindow::MainWindow(QWidget *parent) :  
    QMainWindow(parent),  
    ui(new Ui::MainWindow)  
{  
    ui->setupUi(this);  
}
```

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

Appel du constructeur de la classe MainWindow

Allocation d'un objet de type Ui::MainWindow dont l'adresse est stockée dans le pointeur ui

Initialisation de l'objet pointé par ui (allocation et initialisation des différents widgets qui composent la fenêtre principale)

# Squelette de l'application

- main.cpp

```
#include <QtWidgets/QApplication>
#include "mainwindow.h"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

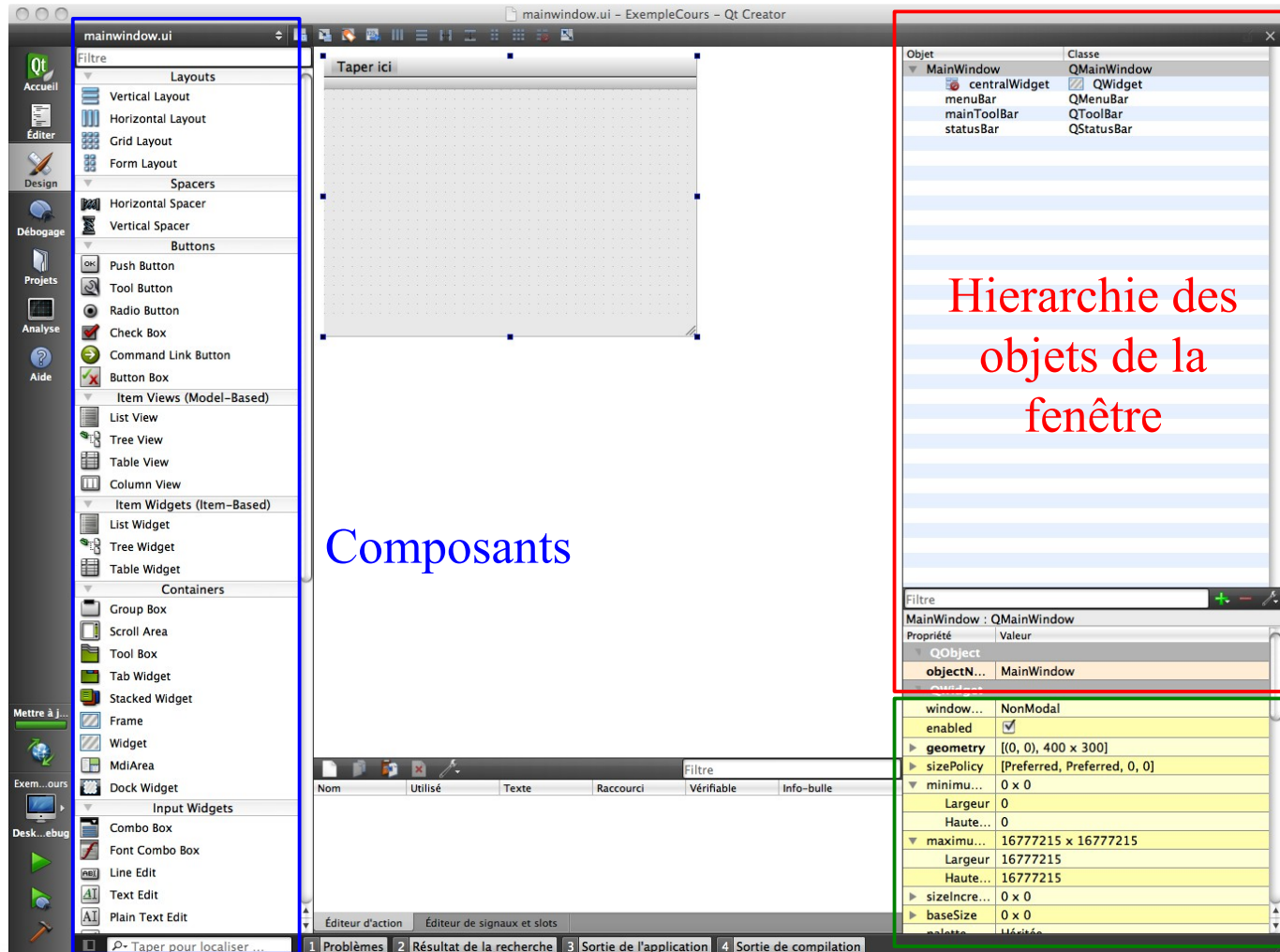
Création de l'objet de type QApplication

Création de la fenêtre principale

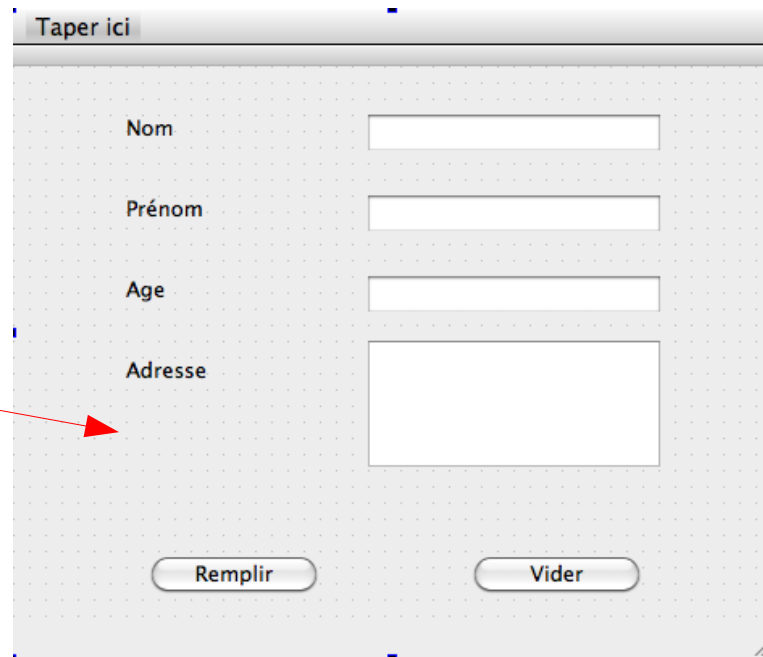
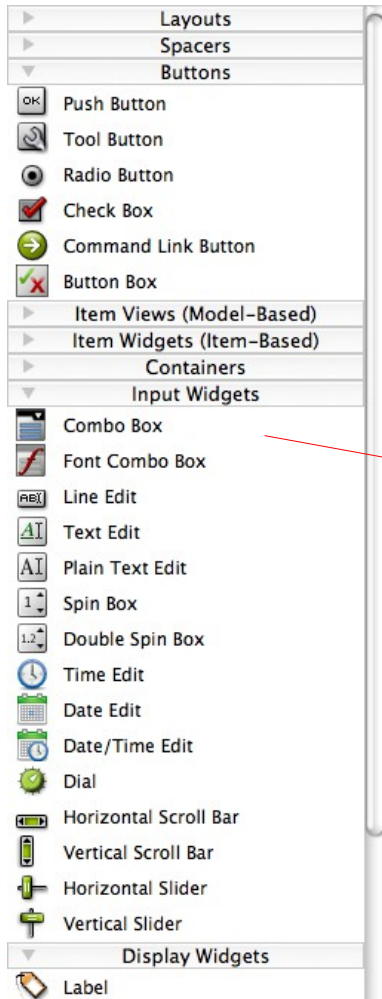
Affichage de la fenêtre principale

Démarrage d'une boucle infinie dont le rôle est de capter et de distribuer les événements aux différents objets

# Personnalisation de la fenêtre principale avec Qt Designer



# Ajouter les composants, les renommer et changer leurs textes



Objet	Classe
MainWindow	QMainWindow
centralWidget	QWidget
bRemplir	QPushButton
bVider	QPushButton
label	QLabel
label_2	QLabel
label_3	QLabel
label_4	QLabel
leAge	QLineEdit
leNom	QLineEdit
lePrenom	QLineEdit
teAdresse	QTextEdit
menuBar	QMenuBar
mainToolBar	QToolBar
statusBar	QStatusBar

Pour changer le nom d'un objet : clic droit objet → Modifier objectName  
Pour changer le texte d'un objet : clic droit objet → Modifier le texte...

# Modifications apportées au fichier mainwindow.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
  <class>MainWindow</class>
  <widget class="QMainWindow" name="MainWindow">
    <property name="geometry">
      <rect>
        <x>0</x>
        <y>0</y>
        <width>669</width>
        <height>606</height>
      </rect>
    </property>
    <property name="windowTitle">
      <string>MainWindow</string>
    </property>
    <widget class="QWidget" name="centralWidget">
      <widget class="QLineEdit" name="leNom">
        <property name="geometry">
          <rect>
            <x>210</x>
            <y>60</y>
            <width>331</width>
            <height>31</height>
          </rect>
        </property>
      </widget>
      <widget class="QLineEdit" name="lePrenom">
        <property name="geometry">
          <rect>
            <x>210</x>
            <y>120</y>
            <width>331</width>
            <height>31</height>
          </rect>
        </property>
      </widget>
      <widget class="QLineEdit" name="leAge">
        <property name="geometry">
          <rect>
```

```

            <x>210</x>
            <y>180</y>
            <width>331</width>
            <height>31</height>
          </rect>
        </property>
      </widget>
      <widget class="QTextEdit" name="teAdresse">
        <property name="geometry">
          <rect>
            <x>210</x>
            <y>250</y>
            <width>331</width>
            <height>191</height>
          </rect>
        </property>
      </widget>
      <widget class="QLabel" name="label">
        <property name="geometry">
          <rect>
            <x>110</x>
            <y>60</y>
            <width>161</width>
            <height>31</height>
          </rect>
        </property>
        <property name="text">
          <string>Nom</string>
        </property>
      </widget>
      <widget class="QLabel" name="label_2">
        <property name="geometry">
          <rect>
            <x>110</x>
            <y>120</y>
            <width>161</width>
            <height>31</height>
          </rect>
        </property>
```

# Modifications apportées au fichier mainwindow.ui

```
<property name="text">
  <string>Prénom</string>
</property>
</widget>
<widget class="QLabel" name="label_3">
  <property name="geometry">
    <rect>
      <x>110</x>
      <y>180</y>
      <width>161</width>
      <height>31</height>
    </rect>
  </property>
  <property name="text">
    <string>Age</string>
  </property>
</widget>
<widget class="QLabel" name="label_4">
  <property name="geometry">
    <rect>
      <x>110</x>
      <y>250</y>
      <width>161</width>
      <height>31</height>
    </rect>
  </property>
  <property name="text">
    <string>Adresse</string>
  </property>
</widget>
<widget class="QPushButton" name="bRemplir">
  <property name="geometry">
    <rect>
      <x>150</x>
      <y>480</y>
      <width>113</width>
      <height>32</height>
    </rect>
  </property>
  <property name="text">
```

```
<string>Remplir</string>
</property>
</widget>
<widget class="QPushButton" name="bVider">
  <property name="geometry">
    <rect>
      <x>390</x>
      <y>480</y>
      <width>113</width>
      <height>32</height>
    </rect>
  </property>
  <property name="text">
    <string>Vider</string>
  </property>
</widget>
</widget>
<widget class="QMenuBar" name="menuBar">
  <property name="geometry">
    <rect>
      <x>0</x>
      <y>0</y>
      <width>669</width>
      <height>22</height>
    </rect>
  </property>
</widget>
<widget class="QToolBar" name="mainToolBar">
  <attribute name="toolBarArea">
    <enum>TopToolBarArea</enum>
  </attribute>
  <attribute name="toolBarBreak">
    <bool>false</bool>
  </attribute>
</widget>
<widget class="QStatusBar" name="statusBar"/>
</widget>
<layoutdefault spacing="6" margin="11"/>
<resources/>
<connections/>
</ui>
```



# Modifications apportées au fichier ui\_mainwindow.h

```
/*
*****
** Form generated from reading UI file 'mainwindow.ui'
**
** Created by: Qt User Interface Compiler version 5.7.0
**
** WARNING! All changes made in this file will be lost
** when recompiling UI file!
*****
*/

#ifndef UI_MAINWINDOW_H
#define UI_MAINWINDOW_H

#include <QtCore/QVariant>
#include <QtWidgets/QAction>
#include <QtWidgets/QApplication>
#include <QtWidgets/QButtonGroup>
#include <QtWidgets/QHeaderView>
#include <QtWidgets/QLabel>
#include <QtWidgets/QLineEdit>
#include <QtWidgets/QMainWindow>
#include <QtWidgets/QMenuBar>
#include <QtWidgets/QPushButton>
#include <QtWidgets/QStatusBar>
#include <QtWidgets/QTextEdit>
#include <QtWidgets/QToolBar>
#include <QtWidgets/QWidget>

QT_BEGIN_NAMESPACE

class Ui_MainWindow
{
public:
    QWidget *centralWidget;
    QLineEdit *leNom;
    QLineEdit *lePrenom;

```

```
QLineEdit *leAge;
QTextEdit *teAdresse;
QLabel *label;
QLabel *label_2;
QLabel *label_3;
QLabel *label_4;
QPushButton *bRemplir;
QPushButton *bVider;
QMenuBar *menuBar;
QToolBar *mainToolBar;
QStatusBar *statusBar;

void setupUi(QMainWindow *MainWindow)
{
    if (MainWindow->objectName().isEmpty())
        MainWindow->setObjectName(QStringLiteral("MainWindow"));
    MainWindow->resize(669, 606);
    centralWidget = new QWidget(MainWindow);
    centralWidget->setObjectName(QStringLiteral("centralWidget"));
    leNom = new QLineEdit(centralWidget);
    leNom->setObjectName(QStringLiteral("leNom"));
    leNom->setGeometry(QRect(210, 60, 331, 31));
    lePrenom = new QLineEdit(centralWidget);
    lePrenom->setObjectName(QStringLiteral("lePrenom"));
    lePrenom->setGeometry(QRect(210, 120, 331, 31));
    leAge = new QLineEdit(centralWidget);
    leAge->setObjectName(QStringLiteral("leAge"));
    leAge->setGeometry(QRect(210, 180, 331, 31));
    teAdresse = new QTextEdit(centralWidget);
    teAdresse->setObjectName(QStringLiteral("teAdresse"));
    teAdresse->setGeometry(QRect(210, 250, 331, 191));
    label = new QLabel(centralWidget);
    label->setObjectName(QStringLiteral("label"));
    label->setGeometry(QRect(110, 60, 161, 31));
    label_2 = new QLabel(centralWidget);
    label_2->setObjectName(QStringLiteral("label_2"));
    label_2->setGeometry(QRect(110, 120, 161, 31));
    label_3 = new QLabel(centralWidget);
    label_3->setObjectName(QStringLiteral("label_3"));

```

# Modifications apportées au fichier ui\_mainwindow.h

```
label_3->setGeometry(QRect(110, 180, 161, 31));
label_4 = new QLabel(centralWidget);
label_4->setObjectName(QStringLiteral("label_4"));
label_4->setGeometry(QRect(110, 250, 161, 31));
bRemplir = new QPushButton(centralWidget);
bRemplir->setObjectName(QStringLiteral("bRemplir"));
bRemplir->setGeometry(QRect(150, 480, 113, 32));
bVider = new QPushButton(centralWidget);
bVider->setObjectName(QStringLiteral("bVider"));
bVider->setGeometry(QRect(390, 480, 113, 32));
MainWindow->setCentralWidget(centralWidget);
menuBar = new QMenuBar(MainWindow);
menuBar->setObjectName(QStringLiteral("menuBar"));
menuBar->setGeometry(QRect(0, 0, 669, 22));
MainWindow->setMenuBar(menuBar);
mainToolBar = new QToolBar(MainWindow);
mainToolBar->setObjectName(QStringLiteral("mainToolBar"));
MainWindow->addToolBar(Qt::TopToolBarArea, mainToolBar);
statusBar = new QStatusBar(MainWindow);
statusBar->setObjectName(QStringLiteral("statusBar"));
MainWindow->setStatusBar(statusBar);

retranslateUi(MainWindow);

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

void retranslateUi(QMainWindow *MainWindow)
{
    MainWindow->setWindowTitle(QApplication::translate("MainWindow", "MainWindow", 0));
    label->setText(QApplication::translate("MainWindow", "Nom", 0));
    label_2->setText(QApplication::translate("MainWindow", "Pr\303\251nom", 0));
    label_3->setText(QApplication::translate("MainWindow", "Age", 0));
    label_4->setText(QApplication::translate("MainWindow", "Adresse", 0));
    bRemplir->setText(QApplication::translate("MainWindow", "Remplir", 0));
    bVider->setText(QApplication::translate("MainWindow", "Vider", 0));
} // retranslateUi

};
```

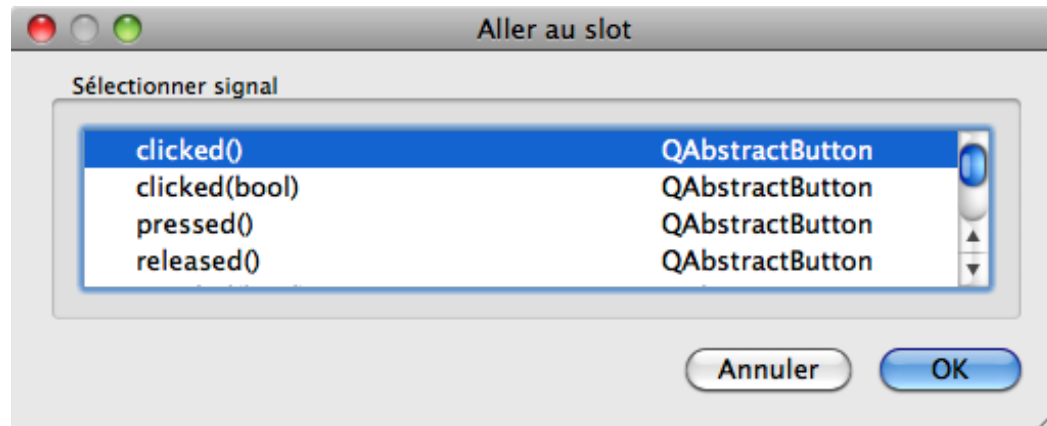
```
namespace Ui {
    class MainWindow: public Ui_MainWindow {};
} // namespace Ui

QT_END_NAMESPACE

#endif // UI_MAINWINDOW_H
```

# Création des slots pour traiter les signaux des boutons

- Clic droit sur le bouton → Aller au slot



# Création des slots pour traiter les signaux des boutons

- mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

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

private slots:
    void on_bRemplir_clicked();

    void on_bVider_clicked();

private:
    Ui::MainWindow *ui;
};

#endif // MAINWINDOW_H
```

# Création des slots pour traiter les signaux des boutons

- mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

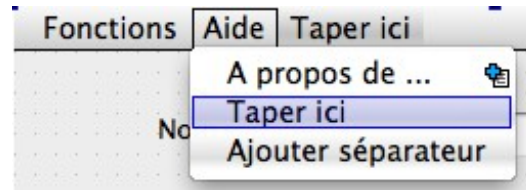
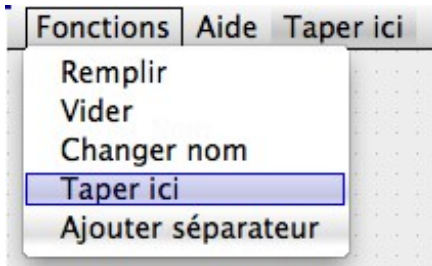
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::on_bRemplir_clicked()
{
    this->ui->leNom->setText("Dupont");
    this->ui->lePrenom->setText("Stephane");
    this->ui->leAge->setText("15");
    this->ui->teAdresse->setText("Polytech Clermont-
Ferrand\nCampus des Cezeaux");
}

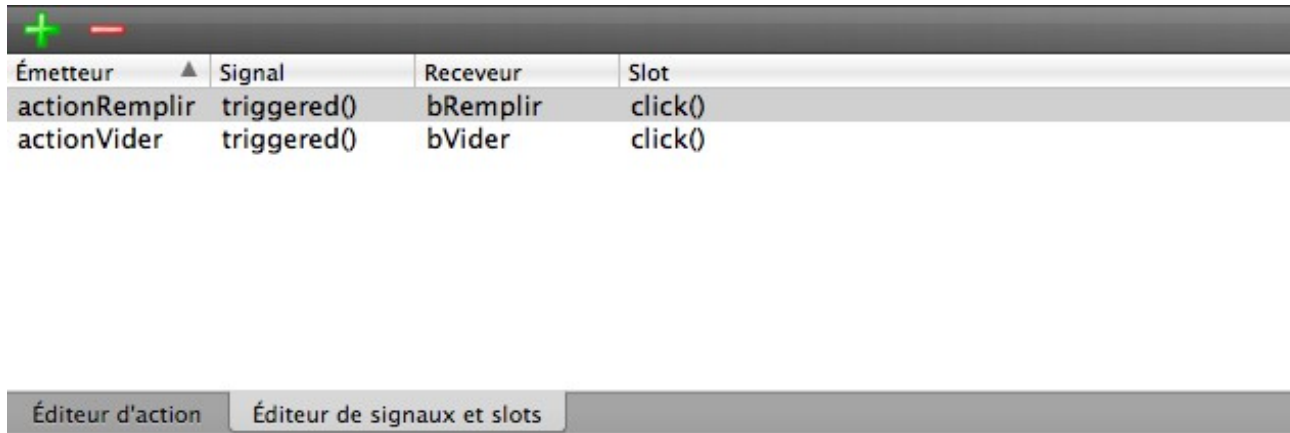
void MainWindow::on_bVider_clicked()
{
    this->ui->leNom->clear();
    this->ui->lePrenom->clear();
    this->ui->leAge->clear();
    this->ui->teAdresse->clear();
}
```

# Ajouter des menus et des actions à la barre de Menus



Objet	Classe
▼ MainWindow	QMainWindow
▼ centralWidget	QWidget
bRemplir	QPushButton
bVider	QPushButton
label	QLabel
label_2	QLabel
label_3	QLabel
label_4	QLabel
leAge	QLineEdit
leNom	QLineEdit
lePrenom	QLineEdit
teAdresse	QTextEdit
▼ menuBar	QMenuBar
▼ menuFonctions	QMenu
actionRemplir	QAction
actionVider	QAction
action_Changer_nom	QAction
▼ menuAide	QMenu
actionA_propos_de	QAction
▶ mainToolBar	QToolBar
statusBar	QStatusBar

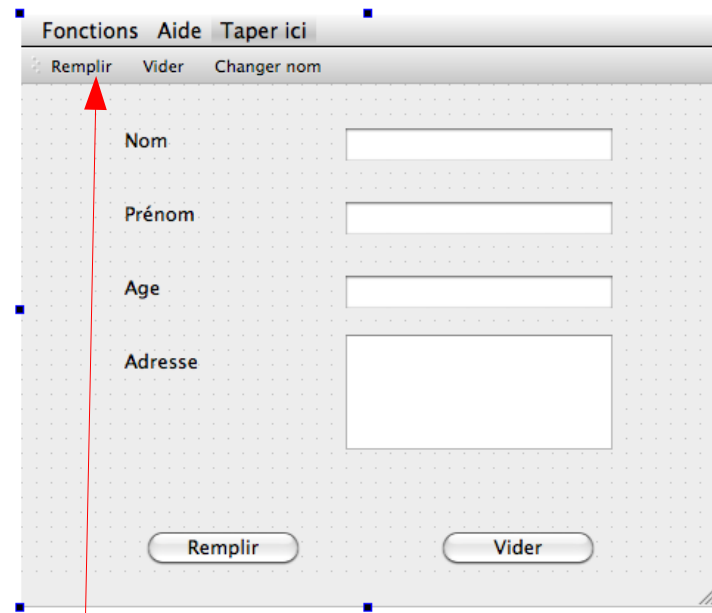
# Connecter les actions des menus aux slots des boutons



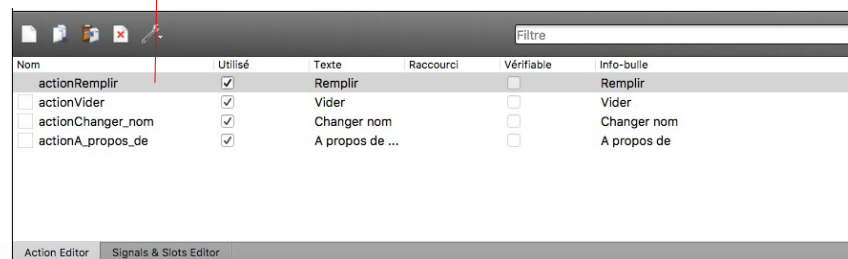
Émetteur ▲	Signal	Receveur	Slot
actionRemplir	triggered()	bRemplir	click()
actionVider	triggered()	bVider	click()

Éditeur d'action   Éditeur de signaux et slots

# Ajouter des actions à la barre d'outils



Glisser / déposer





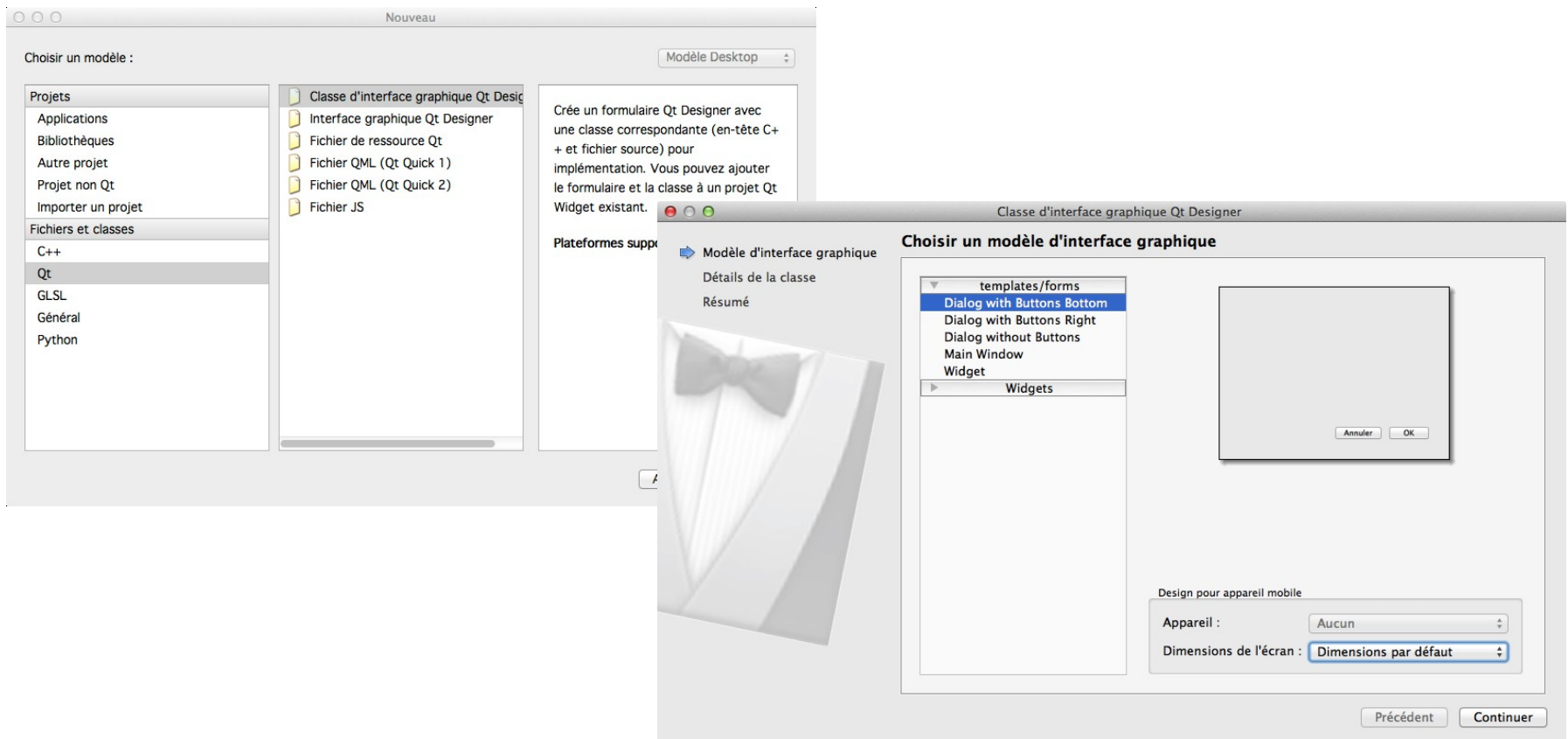
# Modifications apportées aux différents fichiers

---

A vous de regarder !!!!

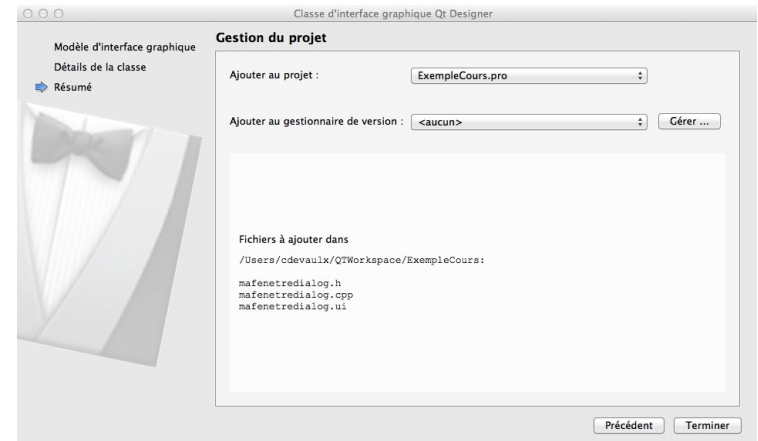
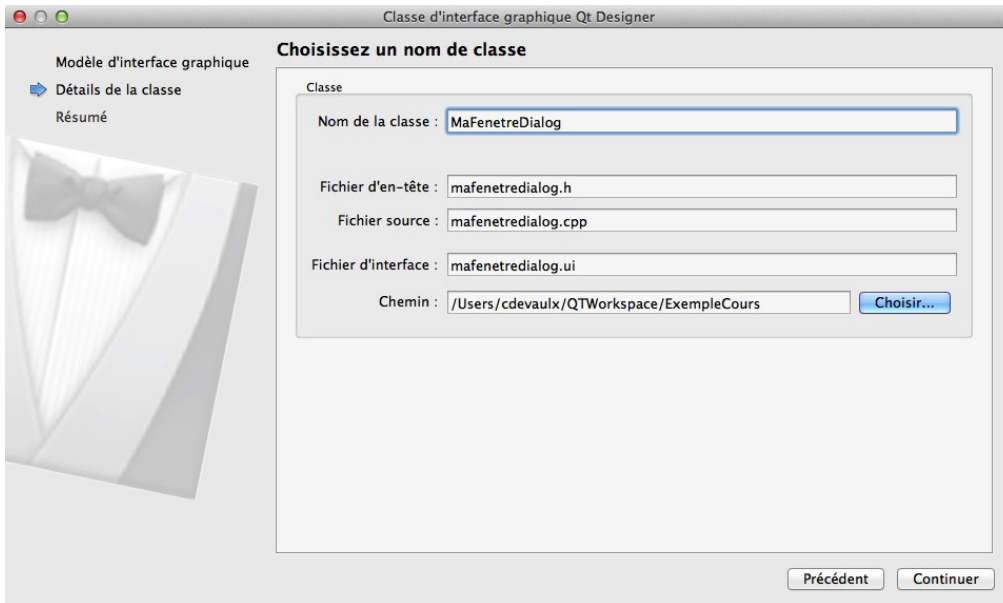
# Ajouter une boîte de dialogue à l'application

- Fichier → Nouveau fichier ou projet...

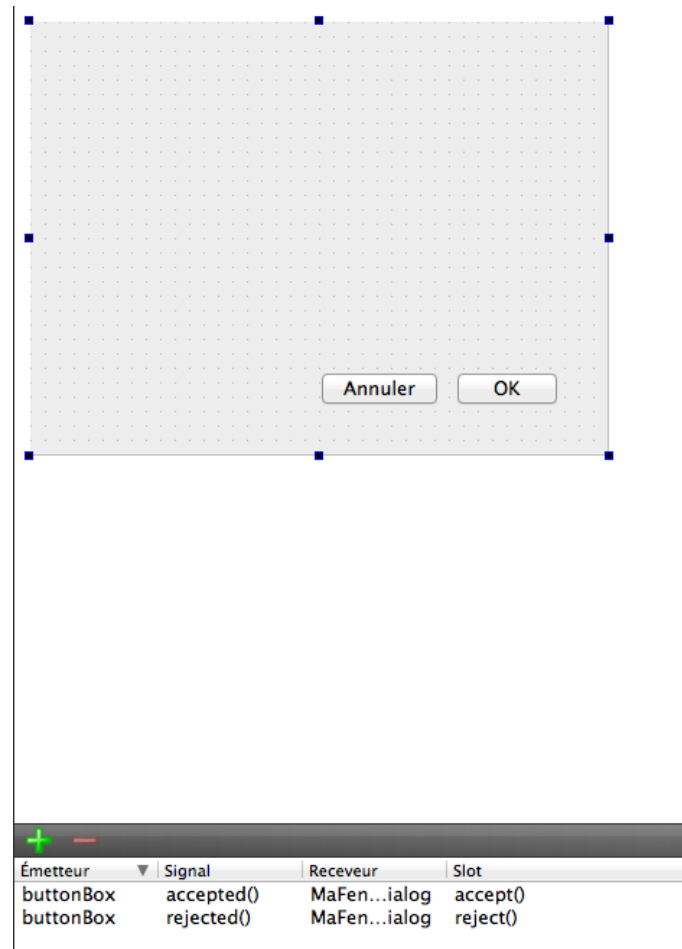


# Ajouter une boîte de dialogue à l'application

- Suite :



# Ajouter une boîte de dialogue à l'application



# Ajouter une boite de dialogue à l'application

- mafenetredialog.h

```
#ifndef MAFENETREDIALOG_H
#define MAFENETREDIALOG_H

#include <QDialog>

namespace Ui {
class MaFenetreDialog;
}

class MaFenetreDialog : public QDialog
{
    Q_OBJECT

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

private:
    Ui::MaFenetreDialog *ui;
};

#endif // MAFENETREDIALOG_H
```

# Ajouter une boite de dialogue à l'application

- mafenetredialog.cpp

```
#include "mafenetredialog.h"
#include "ui_mafenetredialog.h"

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

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

# Ajouter une boite de dialogue à l'application

- mafenetredialog.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
  <class>MaFenetreDialog</class>
  <widget class="QDialog" name="MaFenetreDialog">
    <property name="geometry">
      <rect>
        <x>0</x>
        <y>0</y>
        <width>400</width>
        <height>300</height>
      </rect>
    </property>
    <property name="windowTitle">
      <string>Dialog</string>
    </property>
    <widget class="QDialogButtonBox" name="buttonBox">
      <property name="geometry">
        <rect>
          <x>30</x>
          <y>240</y>
          <width>341</width>
          <height>32</height>
        </rect>
      </property>
      <property name="orientation">
        <enum>Qt::Horizontal</enum>
      </property>
      <property name="standardButtons">
        <set>QDialogButtonBox::Cancel|QDialogButtonBox::Ok</set>
      </property>
    </widget>
  </widget>
```

```
<resources/>
<connections>
  <connection>
    <sender>buttonBox</sender>
    <signal>accepted</signal>
    <receiver>MaFenetreDialog</receiver>
    <slot>accept</slot>
    <hints>
      <hint type="sourcelabel">
        <x>248</x>
        <y>254</y>
      </hint>
      <hint type="destinationlabel">
        <x>157</x>
        <y>274</y>
      </hint>
    </hints>
  </connection>
  <connection>
    <sender>buttonBox</sender>
    <signal>rejected</signal>
    <receiver>MaFenetreDialog</receiver>
    <slot>reject</slot>
    <hints>
      <hint type="sourcelabel">
        <x>316</x>
        <y>260</y>
      </hint>
      <hint type="destinationlabel">
        <x>286</x>
        <y>274</y>
      </hint>
    </hints>
  </connection>
</connections>
</ui>
```

# Ajouter une boite de dialogue à l'application

- `ui_mafenetredialog.h`

```
/*
** Form generated from reading UI file 'mafenetredialog.ui'
**
** Created by: Qt User Interface Compiler version 5.7.0
**
** WARNING! All changes made in this file will be lost when recompiling UI file!
**
*/

#ifndef UI_MAFENETREDIALOG_H
#define UI_MAFENETREDIALOG_H

#include <QtCore/QVariant>
#include <QtWidgets/QAction>
#include <QtWidgets/QApplication>
#include <QtWidgets/QButtonGroup>
#include <QtWidgets/QDialog>
#include <QtWidgets/QDialogButtonBox>
#include <QtWidgets/QHeaderView>

QT_BEGIN_NAMESPACE

class Ui_MaFenetreDialog
{
public:
    QDialogButtonBox *buttonBox;

    void setupUi(QDialog *MaFenetreDialog)
    {
        if (MaFenetreDialog->objectName().isEmpty())
            MaFenetreDialog->setObjectName(QStringLiteral("MaFenetreDialog"));
        MaFenetreDialog->resize(400, 300);
        buttonBox = new QDialogButtonBox(MaFenetreDialog);
        buttonBox->setObjectName(QStringLiteral("buttonBox"));
        buttonBox->setGeometry(QRect(30, 240, 341, 32));
        buttonBox->setOrientation(Qt::Horizontal);
        buttonBox->setStandardButtons(QDialogButtonBox::Cancel|QDialogButtonBox::Ok);
    }
};

#endif
```



# Ajouter une boite de dialogue à l'application

- ui\_mafenetredialog.h suite

```
retranslateUi(MaFenetreDialog);
QObject::connect(buttonBox, SIGNAL(accepted()), MaFenetreDialog, SLOT(accept()));
QObject::connect(buttonBox, SIGNAL(rejected()), MaFenetreDialog, SLOT(reject()));

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

void retranslateUi(QDialog *MaFenetreDialog)
{
    MaFenetreDialog->setWindowTitle(QApplication::translate("MaFenetreDialog", "Dialog", 0));
} // retranslateUi

};

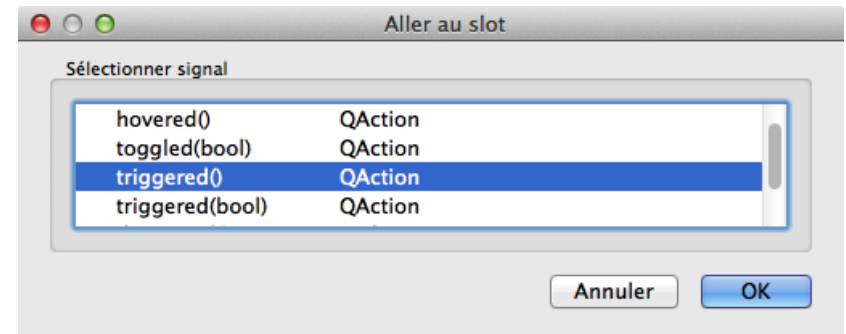
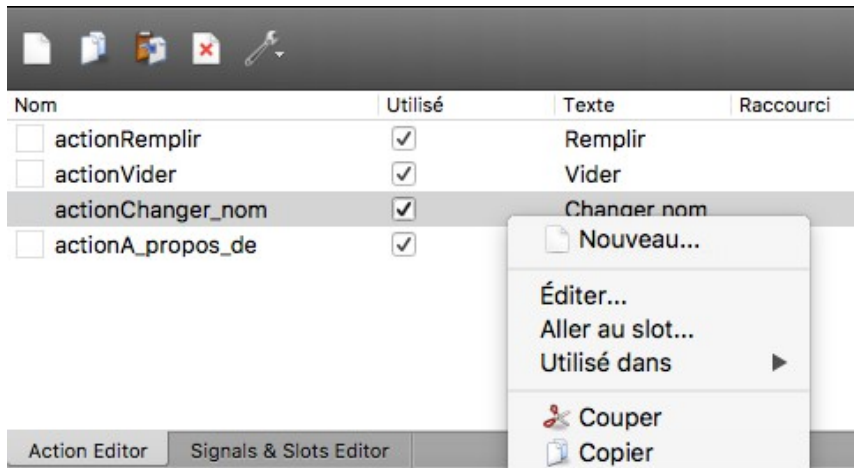
namespace Ui {
    class MaFenetreDialog: public Ui_MaFenetreDialog {};
} // namespace Ui

QT_END_NAMESPACE

#endif // UI_MAFENETREDIALOG_H
```

# Ouvrir la boîte de dialogue depuis la fenêtre principale

- Clic droit sur l'action `action_Changer_Nom` → Aller au slot...



# Ouvrir la boîte de dialogue depuis la fenêtre principale

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

namespace Ui {class MainWindow;}

class MainWindow : public QMainWindow
{
    Q_OBJECT

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

private slots:
    void on_bRemplir_clicked();
    void on_bVider_clicked();
    void on_action_Changer_nom_triggered();

private:
    Ui::MainWindow *ui;
};

#endif // MAINWINDOW_H
```

mainwindow.h

# Ouvrir la boîte de dialogue depuis la fenêtre principale

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "mafenetredialog.h"





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

//...

void MainWindow::on_action_Changer_nom_triggered()
{
    MaFenetreDialog f(this);
    f.exec();
}
```

mainwindow.cpp

# Personnaliser la boîte de dialogue

Objet	Classe
▼  MaFenetreDialog	QDialog
buttonBox	 QDialogButtonBox
label	 QLabel
leNom	 QLineEdit

# Communication entre la boîte de dialogue et la fenêtre principale

```
#ifndef MAFENETREDIALOG_H
#define MAFENETREDIALOG_H

#include <QDialog>

namespace Ui { class MaFenetreDialog; }

class MaFenetreDialog : public QDialog {
    Q_OBJECT

public:
    explicit MaFenetreDialog(QWidget *parent = 0);
    ~MaFenetreDialog();
    void setNom(QString nom);
    QString getNom();

private:
    Ui::MaFenetreDialog *ui;
};

#endif // MAFENETREDIALOG_H
```

mafenetredialog.h

# Communication entre la boîte de dialogue et la fenêtre principale

```
#include "mafenetredialog.h"
#include "ui_mafenetredialog.h"

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

MaFenetreDialog::~MaFenetreDialog() {
    delete ui;
}

void MaFenetreDialog::setNom(QString nom) {
    ui->leNom->setText(nom);
}

QString MaFenetreDialog::getNom() {
    return ui->leNom->text();
}
```

mafenetredialog.cpp

# Communication entre la boîte de dialogue et la fenêtre principale

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "mafenetredialog.h"

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

//...

void MainWindow::on_action_Changer_nom_triggered()
{
    MaFenetreDialog f(this) ;
    f.setNom(this->ui->leNom->text());
    if (f.exec() == QDialog::Accepted)
    {
        this->ui->leNom->setText(f.getNom());
    }
}
```

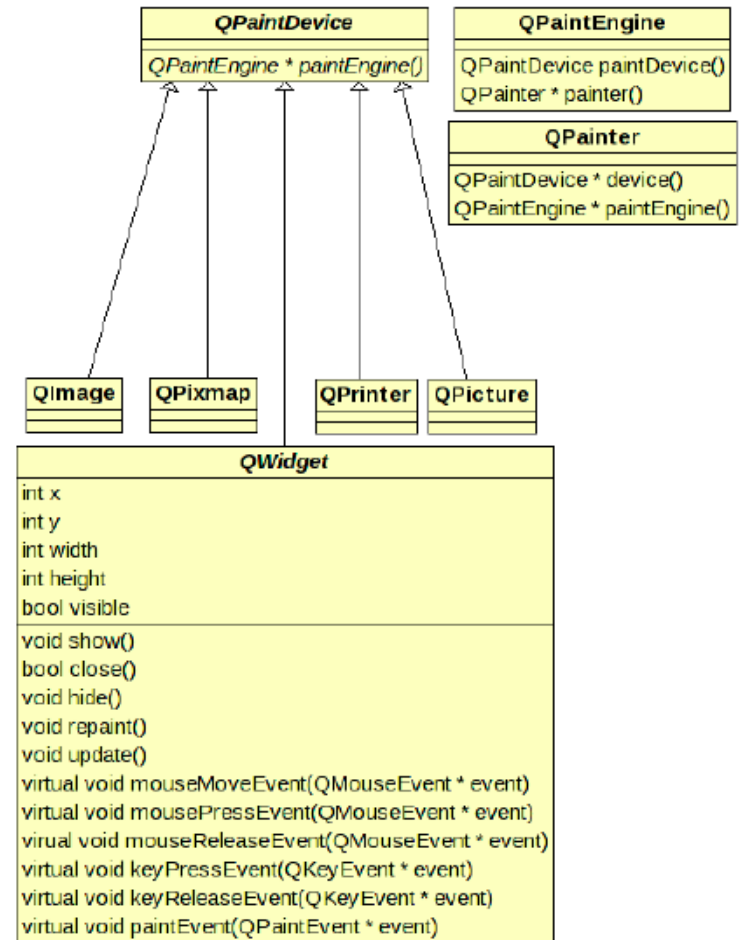
mainwindow.cpp



# Tracés graphiques 2D avec QPainter

# La classe QPainter

- La classe **QPainter** est la classe utilisée pour effectuer les opérations de dessin 2D sous Qt.
- Cette classe peut fonctionner sur n'importe quel objet qui dérive de la classe **QPaintDevice**.
- La classe **QPaintEngine** fourni le moteur de rendu et l'interface abstraite qui permettent à QPainter de s'adapter à la plateforme utilisée.



# La classe QPainter

- Les objets instanciés à partir de la classe QPainter peuvent-être vus comme des "feuilles de dessin blanches" sur lesquelles il est possible de tracer des formes, d'afficher des images...
- Pour effectuer ces opérations il faut utiliser les méthodes fournies avec cette classe. En voici quelques unes :
  - drawLine, drawRect, fillRect, drawEllipse, fillEllipse ;
  - drawPixmap ;
  - DrawText.

# La classe QPainter

- Il est possible de modifier
  - le crayon utilisé pour tracer le contour des objets à l'aide de la méthode setPen ;
  - le pinceau utilisé pour colorer l'intérieur des objets à l'aide de la méthode setBrush ;
  - La police de caractères utilisée pour tracer du texte à l'aide de la méthode setFont.

# La méthode paintEvent

- C'est cette méthode qui est appelée lorsque la fenêtre doit être redessinée.
- Pour assurer la persistance des dessins réalisées dans une fenêtre, cette dernière doit donc redéfinir la méthode **void paintEvent(QPaintEvent\* e)**.
- Il est possible de forcer le réaffichage d'une fenêtre à l'aide de la commande : **repaint()**.

# Exemple

```
#ifndef CRECTANGLE_H
#define CRECTANGLE_H

#include <QPainter>

class CRectangle
{
private:
    int x;
    int y;
    int largeur;
    int hauteur;

public:
    CRectangle();
    CRectangle(int x, int y, int l, int h);
    void deplacer(int dx, int dy);
    void deplacerDe(int dx, int dy);
    void dessiner(QPainter * p);
};

#endif // CRECTANGLE_H
```

crectangle.h

# Exemple

```
#include "crectangle.h"

CRectangle::CRectangle() {
    x=0;
    y=0;
    largeur=0;
    hauteur=0;
}

CRectangle::CRectangle(int x, int y, int l, int h) {
    this->x=x;
    this->y=y;
    this->largeur=l;
    this->hauteur=h;
}

void CRectangle::deplacer(int dx, int dy) {
    x=dx;
    y=dy;
}
```

crectangle.cpp

# Exemple

```
void CRectangle::deplacerDe(int dx, int dy) {
    x+=dx;
    y+=dy;
}

void CRectangle::dessiner(QPainter * p) {
    p->setPen( QPen(Qt::black, 1) );
    p->drawRect( x, y, largeur, hauteur );
    p->fillRect( x+1, y+1, largeur-1, hauteur-1, QBrush(Qt::green) );
}
```

crectangle.cpp



# Exemple

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include "crectangle.h"

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

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

private:
    Ui::MainWindow *ui;
    CRectangle *r;
    void paintEvent(QPaintEvent* e);
};

#endif // MAINWINDOW_H
```

mainwindow.h

# Exemple

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "crectangle.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow) {
    ui->setupUi(this);
    r = new CRectangle(10,50,70,30);
}

MainWindow::~MainWindow() {
    delete ui;
    delete r;
}
```

mainwindow.cpp

# Exemple

```
void MainWindow::paintEvent(QPaintEvent* e) {  
    QWidget::paintEvent(e);  
    QPainter painter(this);  
  
    painter.setPen( QPen(Qt::red, 1) );  
    painter.setFont(QFont("Arial",16));  
    painter.drawText(10,30,QString("Exemple de tracés 2D"));  
  
    r->dessiner(&painter);  
}
```

mainwindow.cpp

# Exemple



# Capter les événements de la souris

- Pour capter les événements de la souris, il suffit de surcharger les gestionnaires d'évènement associés à ce périphérique dans la classe QWidget :
  - void mousePressEvent(QMouseEvent\* e) ;
  - void mouseReleaseEvent(QMouseEvent\* e) ;
  - void mouseMoveEvent(QMouseEvent\* e) ;
  - void mouseDoubleClickEvent(QMouseEvent\* e).

# La classe QMouseEvent

---

- Cette classe contient toutes les informations relatives à un évènement de la souris :
  - L'identité du bouton
  - La position de la souris
  - Etc...

# Exemple

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QMouseEvent>
#include "crectangle.h"

namespace Ui { class MainWindow; }

class MainWindow : public QMainWindow {
    Q_OBJECT

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

private:
    Ui::MainWindow *ui;
    CRectangle *r;
    void paintEvent(QPaintEvent* e);
    void mousePressEvent(QMouseEvent *e);
};

#endif // MAINWINDOW_H
```

mainwindow.h

# Exemple

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "crectangle.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow) {
    ui->setupUi(this);
    r = new CRectangle(10,50,70,30);
}

MainWindow::~MainWindow() {
    delete ui;
    delete r ;
}
```

mainwindow.cpp



# Exemple

```
void MainWindow::paintEvent(QPaintEvent* e) {
    QWidget::paintEvent(e);
    QPainter painter(this);

    painter.setPen( QPen(Qt::red, 1) );
    painter.setFont(QFont("Arial",16));
    painter.drawText(10,30,QString("Exemple de tracés 2D"));

    r->dessiner(&painter);
}

void MainWindow::mousePressEvent(QMouseEvent *e) {
    if(e->button() == Qt::LeftButton)
    {
        r->deplacer(e->x(),e->y());
        this->repaint();
    }
}
```

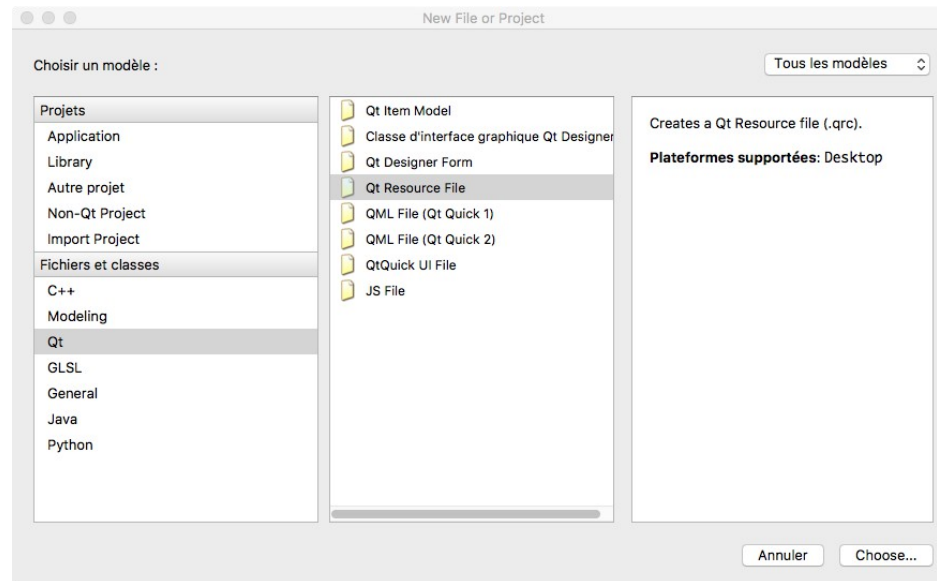
mainwindow.cpp

# Exemple

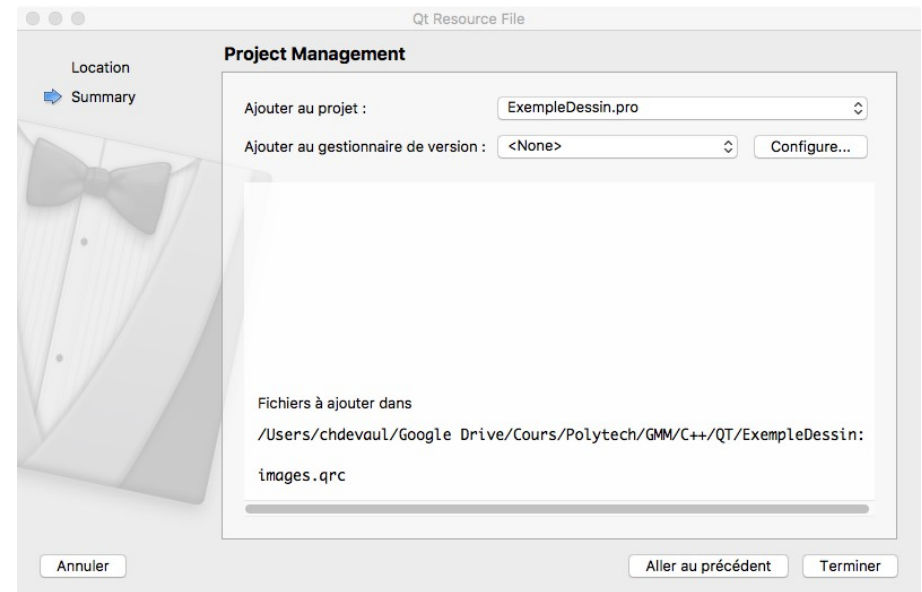
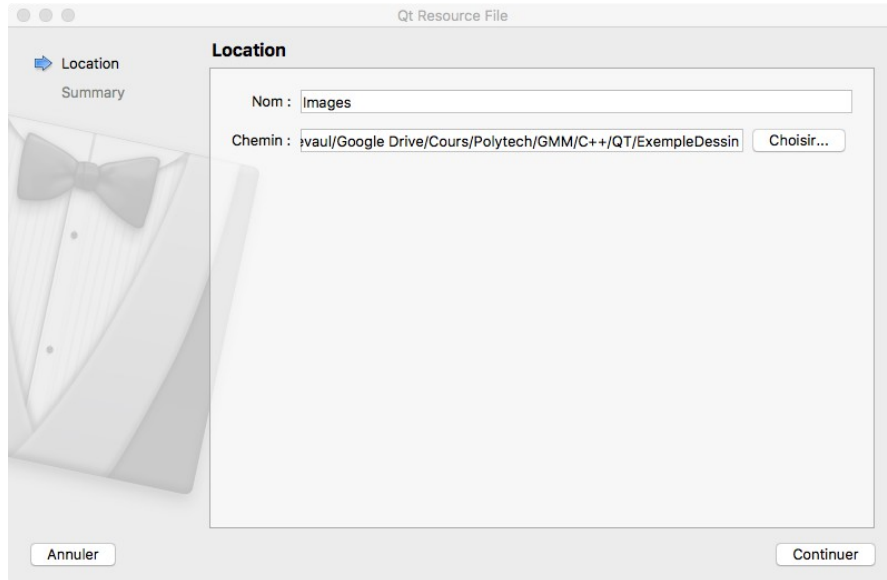


# Afficher une image dans une fenêtre

- Pour afficher une image dans une fenêtre il faut d'abord l'ajouter aux ressources du projet :
  - Fichier → Nouveau fichier ou projet...

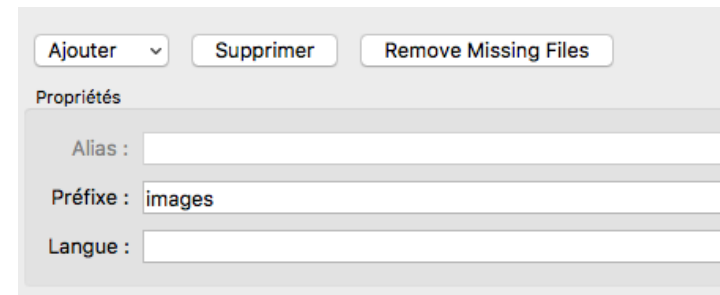
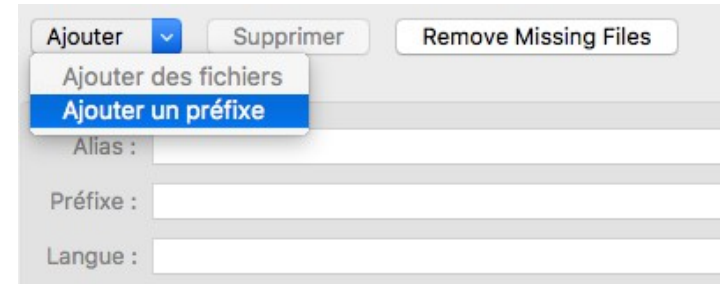
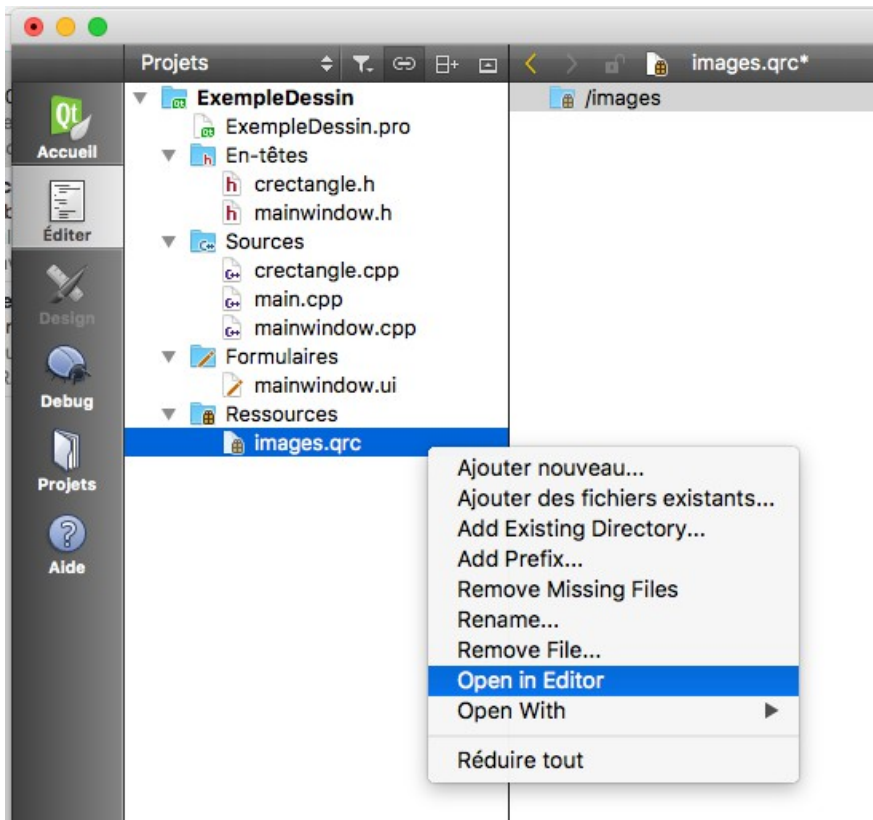


# Afficher une image dans une fenêtre



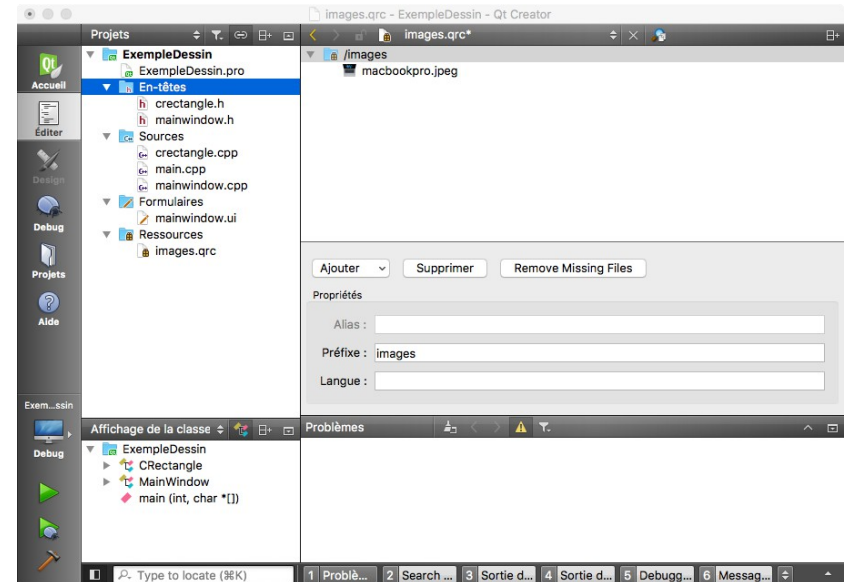
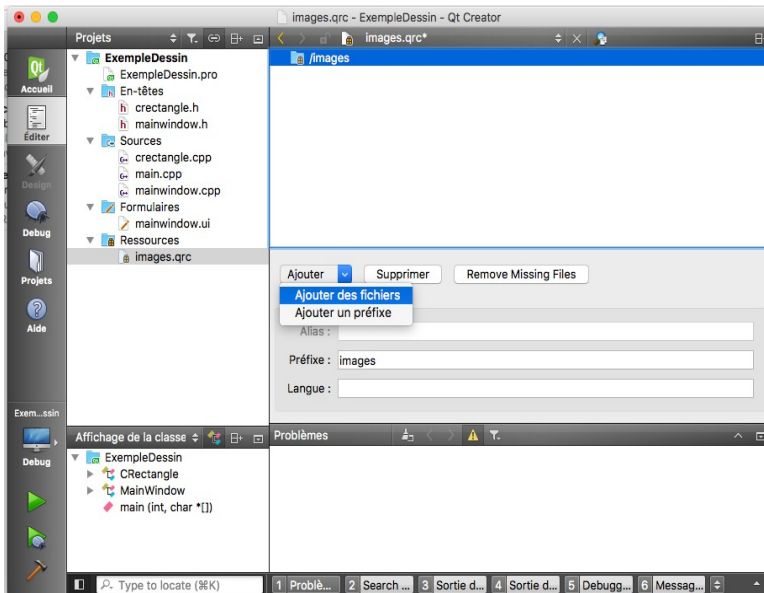
# Afficher une image dans une fenêtre

- On ajoute ensuite un préfixe :



# Afficher une image dans une fenêtre

– On ajoute enfin l'image :



# Afficher une image dans une fenêtre

- On utilise ensuite la méthode `drawPixmap`.  
Exemple :

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "crectangle.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow) {
    ui->setupUi(this);
    r = new CRectangle(10,50,70,30);
}

MainWindow::~MainWindow() {
    delete ui;
    delete r ;
}
```

mainwindow.cpp

# Afficher une image dans une fenêtre

```
void MainWindow::paintEvent(QPaintEvent* e) {
    QWidget::paintEvent(e);
    QPainter painter(this);

    painter.setPen( QPen(Qt::red, 1) );
    painter.setFont(QFont("Arial",16));
    painter.drawText(10,30,QString("Exemple de tracés 2D"));

    painter.drawPixmap(10,100,100,100,QPixmap(":/images/macbookpro.jpeg"));

    r->dessiner(&painter);
}

void MainWindow::mousePressEvent(QMouseEvent *e) {
    if(e->button() == Qt::LeftButton)
    {
        r->deplacer(e->x(),e->y());
        this->repaint();
    }
}
```

mainwindow.cpp



# Afficher une image dans une fenêtre



# Capter les événements du clavier

---

- Pour capter les événements du clavier, il suffit de surcharger le gestionnaire d'évènement associés à ce périphérique dans la classe QWidget :
  - `void keyPressEvent ( QKeyEvent * event )`

# La classe QKeyEvent

---

- Cette classe contient toutes les informations relatives à un évènement du clavier :
  - Le nombre de touches concernées par l'évènement ;
  - L'identité de la touche (ou des touches) ;
  - Etc...

# Exemple

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QMouseEvent>
#include <QKeyEvent>
#include "crectangle.h"

namespace Ui { class MainWindow; }

class MainWindow : public QMainWindow {
    Q_OBJECT

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

private:
    Ui::MainWindow *ui;
    CRectangle *r;
    void paintEvent(QPaintEvent* e);
    void mousePressEvent(QMouseEvent *e);
    void keyPressEvent ( QKeyEvent * event );
};

#endif // MAINWINDOW_H
```

mainwindow.h

# Exemple

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "crectangle.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow) {
    ui->setupUi(this);
    r = new CRectangle(10,50,70,30);
}

MainWindow::~MainWindow() {
    delete ui;
    delete r ;
}
```

mainwindow.cpp

# Exemple

```
void MainWindow::paintEvent(QPaintEvent* e) {
    QWidget::paintEvent(e);
    QPainter painter(this);

    painter.setPen( QPen(Qt::red, 1) );
    painter.setFont(QFont("Arial",16));
    painter.drawText(10,30,QString("Exemple de tracés 2D"));

    r->dessiner(&painter);
}

void MainWindow::mousePressEvent(QMouseEvent *e) {
    if(e->button() == Qt::LeftButton)
    {
        r->deplacer(e->x(),e->y());
        this->repaint();
    }
}
```

mainwindow.cpp

# Exemple

```
void MainWindow::keyPressEvent ( QKeyEvent * event ) {  
    switch(event->key())  
    {  
        case Qt::Key_K : r->deplacerDe(-100,0);  
                        break;  
        case Qt::Key_M : r->deplacerDe(100,0);  
                        break;  
        case Qt::Key_O : r->deplacerDe(0,-100);  
                        break;  
        case Qt::Key_L : r->deplacerDe(0,100);  
                        break;  
    }  
    this->repaint();  
}
```

mainwindow.cpp

# Utilisation d'un timer (classe QTimer)



# Création et destruction du timer

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QTimer>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

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

private:
    Ui::MainWindow *ui;
    QTimer *timer;
};

#endif // MAINWINDOW_H
```

mainwindow.h

# Création et destruction du timer

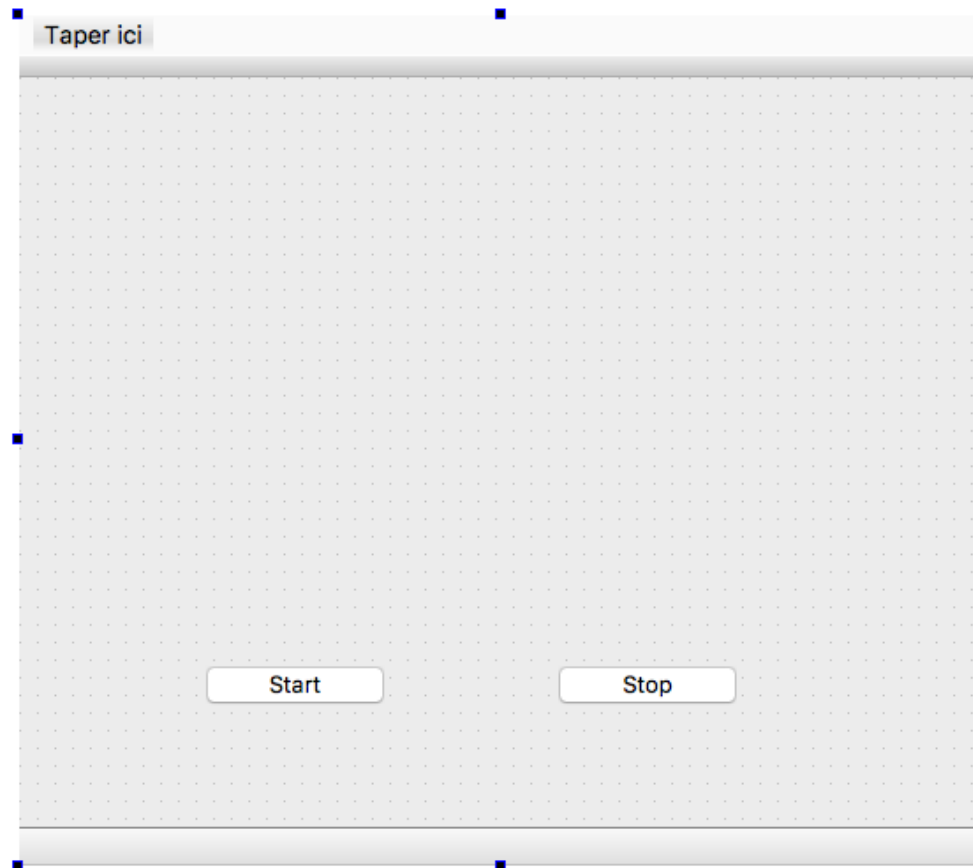
```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    timer = new QTimer(this);
}

MainWindow::~MainWindow()
{
    delete timer;
    delete ui;
}
```

mainwindow.cpp

# Démarrage et arrêt du timer



Objet	Classe
MainWindow	QMainWindow
centralWidget	QWidget
bStart	QPushButton
bStop	QPushButton
menuBar	QMenuBar
mainToolBar	QToolBar
statusBar	QStatusBar

# Démarrage et arrêt du timer

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QTimer>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

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

private slots:
    void on_bStart_clicked();

    void on_bStop_clicked();

private:
    Ui::MainWindow *ui;
    QTimer *timer;
};

#endif // MAINWINDOW_H
```

mainwindow.h

# Démarrage et arrêt du timer

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    timer = new QTimer(this);
}

MainWindow::~MainWindow()
{
    delete timer;
    delete ui;
}

void MainWindow::on_bStart_clicked()
{
    timer->start(1000);
}

void MainWindow::on_bStop_clicked()
{
    timer->stop();
}
```

MainWindow.cpp

# Création du gestionnaire d'évènement du timer

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QTimer>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

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

private slots:
    void on_bStart_clicked();
    void on_bStop_clicked();
    void update();

private:
    Ui::MainWindow *ui;
    QTimer *timer;
    int nbIntTimer;
};

#endif // MAINWINDOW_H
```

mainwindow.h

# Création du gestionnaire d'évènement du timer

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow) {
    ui->setupUi(this);
    timer = new QTimer(this);
    connect(timer, SIGNAL(timeout()), this, SLOT(update()));
    nbIntTimer = 0;
}

MainWindow::~MainWindow() {
    delete timer;
    delete ui;
}

void MainWindow::on_bStart_clicked() {
    timer->start(1000);
}

void MainWindow::on_bStop_clicked() {
    timer->stop();
}

void MainWindow::update() {
    nbIntTimer++;
}
```

mainwindow.cpp

# Affichage du nombre d'interruptions du timer

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QTimer>
#include <QPainter>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

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

private slots:
    void on_bStart_clicked();
    void on_bStop_clicked();
    void update();

private:
    Ui::MainWindow *ui;
    QTimer *timer;
    int nbIntTimer;
    void paintEvent(QPaintEvent* e);
};

#endif // MAINWINDOW_H
```

mainwindow.h



# Affichage du nombre d'interruptions du timer

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow) {
    ui->setupUi(this);
    timer = new QTimer(this);
    connect(timer, SIGNAL(timeout()), this, SLOT(update()));
    nbIntTimer = 0;
}

MainWindow::~MainWindow() {
    delete timer;
    delete ui;
}

void MainWindow::on_bStart_clicked() {
    timer->start(1000);
}

void MainWindow::on_bStop_clicked() {
    timer->stop();
}
```

Mainwindow.cpp

# Affichage du nombre d'interruptions du timer

```
void MainWindow::update() {
    nbIntTimer++;
    this->repaint();
}

void MainWindow::paintEvent(QPaintEvent* e) {
    QWidget::paintEvent(e);
    QPainter painter(this);

    painter.setPen( QPen(Qt::red, 1) );
    painter.setFont(QFont("Arial",200));
    painter.drawText(50,250,QString(QString::number(nbIntTimer)));
}
```

Mainwindow.cpp

# Affichage du nombre d'interruptions du timer

