

## R2.02 : développement d'application avec IHM Feuille TD n° 1

### Découverte de QtCreator Application événementielle non graphique simple

#### Objectifs :

- 1.- Activation licence Qt
- 2.- Découverte QtCreator et Qt
- 3.- Création d'une application événementielle simple NON graphique : découverte de signal /slot

### Activité 1

Activation licence Qt : Suivre le tuto disponible sur eLearn

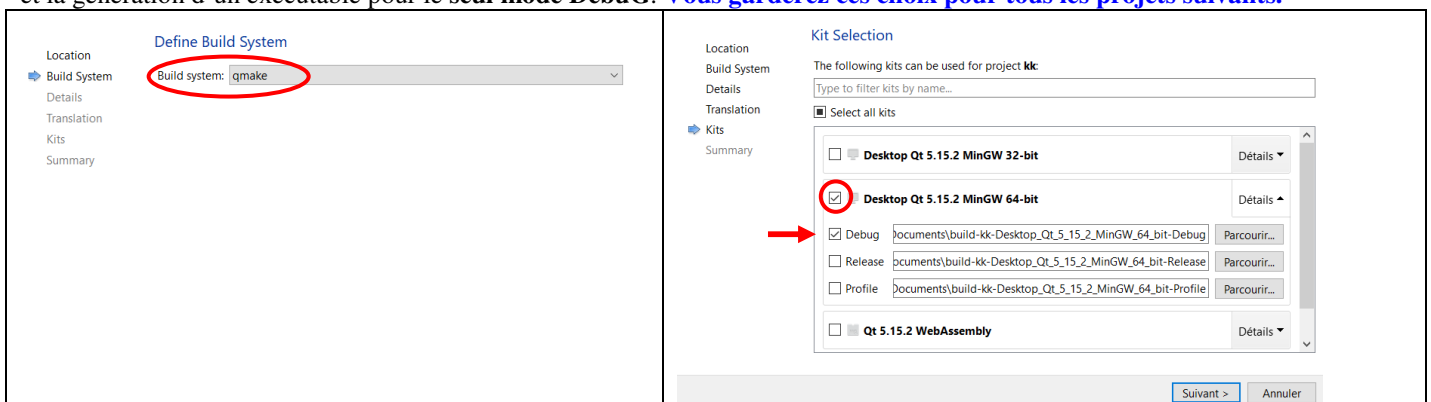
### Activité 2

Dans votre espace de travail, créer un dossier R202DevAplisAvecIHM (ou tout autre nom qui vous conviendra, en évitant espaces et lettres accentuées), dans lequel vous déposerez les travaux réalisés dans le cadre de cette ressource.

### Activité 3

A l'aide de l'Assistant de QtCreator, créer un projet nommé « nonQtProject », de type Application C++ classique NON Qt (cf. Figure 1), pour créer la fameuse application minimale « Hello world ».

Dans les étapes **Define Build System** et **Kit Selection**, vous choisirez respectivement **qmake** et le Compilateur MinGW 64-bit et la génération d'un exécutable pour le **seul mode Debug**. **Vous garderez ces choix pour tous les projets suivants.**



#### Compiler, tester.

Consulter votre dossier afin de voir où sont rangés les codes sources du projet, les fichiers exécutables :

- Sont-ils rangés dans le même dossier ?
- Intérêt de cette organisation ?

Prenez un peu de temps pour vous familiariser avec l'interface de QtCreator et les principales fonctionnalités : compilation, exécution, scinder les fenêtre – pour du code ou des infos sur le projet –, les différents onglets sur le bas de la fenêtre permettant de visualiser les résultats de la compilation, ou de l'exécution du code, l'accès à la documentation via l'écran d'accueil, ...

#### Liens vers présentation QtCreator :

<https://doc.qt.io/qtcreator/creator-overview.html>

<https://doc.qt.io/qtcreator/creator-getting-started.html> se familiariser avec l'interface de l'IDE QtCreator.

## Activité 4

A l'aide de QtCreator, créer un projet nommé « QtNonGUIProject », de type Application Qt non graphique (cf. Figure 2) pour la mise en œuvre d'une application événementielle non graphique.

- 1) Compiler le projet créé et lancer une exécution.
- 2) Expliquez ce que vous constatez : ouverture d'une fenêtre console, vide, et où rien ne se passe. Est-ce 'normal' ? Pourquoi ?

### Création de MaClasse

On va ajouter à ce projet une classe C++, intitulée **MaClasse**.


**Avant de procéder**, répondre à la question 3) après avoir lu avec attention la description de la classe.

La classe **MaClasse** compte :

- une propriété privée : `int _numero ;` // rang de création de l'objet dans le `main()`
- un constructeur (cf. Figure 6)
- un signal `estEmis(int)`, où l'objet émet (en paramètre) son `_numero`.
- une méthode `emettre()`, qui a pour objectif d'émettre un signal contenant son `_numero` **après** affichage d'un message où l'objet indique qui il est (quel est son `_numero`) et qu'il s'apprête à envoyer un signal (cf. Figure 6)
- une méthode `recevoir(int)`, déclenchée par un signal `estEmis(int)` d'un autre objet, où l'objet affiche qui il est (quel est son `_numero`) et quel a été le signal intercepté (cf. Figure 6)

- 3) D'après le comportement attendu des objets, et en vous basant sur le cours (disponible sur eLearn), indiquer quelles sont les options à fournir à l'Assistant Qt (Figure 3) pour qu'il génère le squelette de la classe **MaClasse** (Figure 4).
- 4) Une fois le squelette de la classe créé par l'Assistant, compléter sa déclaration et son corps.

**Astuce** : une fois la déclaration d'une méthode écrite dans le `.h`, il est possible de générer automatiquement son corps à l'aide du menu contextuel : positionner le curseur sur le nom de la méthode, clique-droit-ReFactor-ajouter la définition dans `maclasse.cpp`

**Compiler uniquement ! **

N'oubliez pas les bonnes pratiques de programmation...

### Méthodologie de développement pour les étapes qui suivent

On va compléter le `main()` en plusieurs étapes.

Afin de garder trace de l'évolution du code, la réponse à chaque étape sera codée dans un sous-programme portant le nom de l'étape.

Le `main()` se contentera d'appeler chaque sous-programme créé.

#### 5) Compléter le programme – étape 1

Le comportement attendu du programme est le suivant (cf. Figure 6) :

- Création de 2 objets de la classe **MaClasse**, le premier aura pour numéro 1, le second pour numéro 2.
  - L'objet n°1 émet un signal (appel de `emettre()`).
  - L'objet n°2 l'intercepte.
- a) Le code du sous-programme `etape1()` (cf. Figure 5) est-il complet ?
  - b) Ajuster afin que le programme fasse ce qui est demandé. Compiler et exécuter.
  - c) Dessiner tous les objets de l'application et les connexions existant entre eux.

**Liens** (autres que le cours) :

<https://doc.qt.io/qt-5/signalsandslots.html>

<https://doc.qt.io/>

#### 6) Compléter le programme – étape 2 .

Le comportement attendu du programme est le suivant (cf. Figure 7) :

- Création de 4 objets de la classe **MaClasse**, Chacun a un numéro correspondant à son rang de création.
- L'objet n°1 émet un signal (appel de `emettre()`).
- Seul L'objet n°2 l'intercepte.
- L'objet n°1 émet un signal (appel de `emettre()`).
- Seuls les objets n°3 et n°4 l'interceptent.

- a) Compléter le corps de `etape2()` afin que le programme fasse ce qui est demandé. Compiler et exécuter.

## Enrichir la classe MaClasse

La classe **MaClasse** est munie d'une nouvelle fonctionnalité :

- une méthode publique **getNumero()**, qui retourne la valeur de la propriété **\_numero**

7) Compléter la déclaration et le corps de la classe. Compiler seulement.

8) Compléter le programme – **étape 3**

Le comportement attendu du programme est le suivant (cf. Figure 8) :

- Création de 2 objets de la classe **MaClasse**, le premier aura pour numéro 1, le second pour numéro 2.
- L'objet n°1 émet un signal, sans utiliser la méthode **emettre()**.
- L'objet n°2 l'intercepte.
- L'objet n°1 arrête le programme

- Sur quel objet du programme faudra-t-il agir pour obtenir le résultat souhaité ?
- Consulter la documentation (cf. Liens)
- Sans modifier la classe **MaClasse**, compléter le corps de **etape3()** afin que le programme fasse ce qui est demandé. Compiler et exécuter.

**Liens** (autres que le cours) :

<https://doc.qt.io/qt-5/qcoreapplication.html>  
<https://doc.qt.io/qt-5/signalsandslots.html>  
<https://doc.qt.io/>

**QCoreApplication** \*QCoreApplication::instance()

Returns a pointer to the application's **QCoreApplication** (or **QGuiApplication/QApplication**) instance.

If no instance has been allocated, **nullptr** is returned.

**void QCoreApplication::quit()**

[static slot]

Tells the application to exit with return code 0 (success). Equivalent to calling **QCoreApplication::exit(0)**.

It's common to connect the **QGuiApplication::lastWindowClosed()** signal to **quit()**, and you also often connect e.g. **QAbstractButton::clicked()** or signals in **QAction**, **QMenu**, or **QMenuBar** to it.

It's good practice to always connect signals to this slot using a **QueuedConnection**. If a signal connected (non-queued) to this slot is emitted before control enters the main event loop (such as before "int main" calls **exec()**), the slot has no effect and the application never exits. Using a queued connection ensures that the slot will not be invoked until after control enters the main event loop.

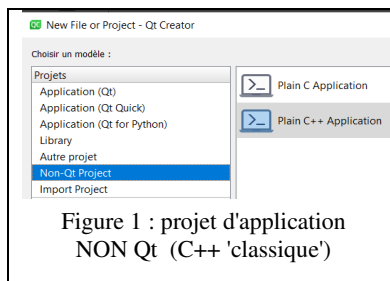


Figure 1 : projet d'application NON Qt (C++ 'classique')

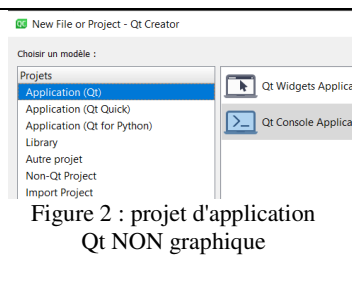


Figure 2 : projet d'application Qt NON graphique

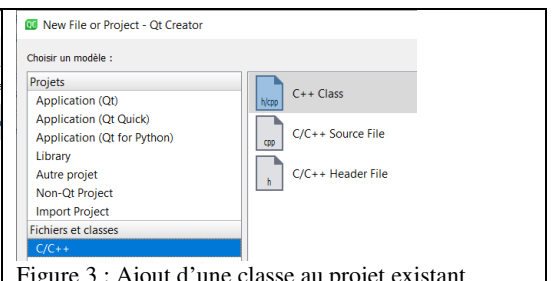


Figure 3 : Ajout d'une classe au projet existant

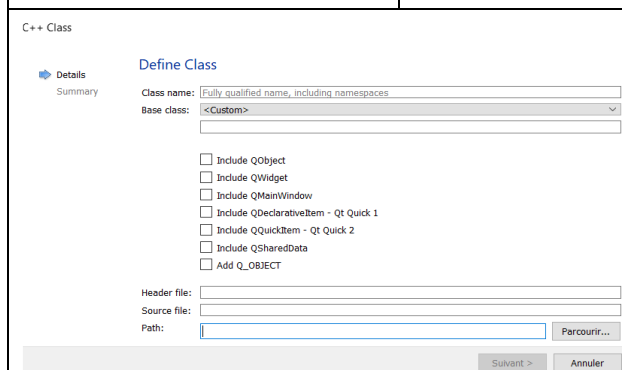


Figure 4 : Caractéristiques de **MaClasse**

```
1  #include <QCoreApplication>
2  #include "maclasse.h"
3
4  // ----- Un sous-programme par étape d'évolution du programme
5  // ----- cf. td n°1
6  void etape1();
7  void etape2();
8  void etape3();
9
10 // -----
11
12 int main(int argc, char *argv[])
13 {
14     QCoreApplication a(argc, argv);
15
16     etape1();
17
18     return a.exec();
19 }
20
21 void etape1()
22 {
23     MaClasse objet1(1);
24     MaClasse objet2(2);
25
26     objet1.emettre();
27 }
```

Figure 5 : **main()** et squelette de **etape1()**

```
creation de 1
creation de 2
Moi, 1, j'emets un signal
Moi, 2, j'ai reçu le signal 1
```

Figure 6 : Comportement étape 1

```
creation de 1
creation de 2
creation de 3
creation de 4
Moi, 1, j'emets un signal
Moi, 2, j'ai reçu le signal 1
Moi, 1, j'emets un signal
Moi, 3, j'ai reçu le signal 1
Moi, 4, j'ai reçu le signal 1
```

Figure 7 : Comportement étape 2

```
creation de 1
creation de 2
Moi, 2, j'ai reçu le signal 1
Appuyez sur <ENTRÉE> pour fermer cette fenêtre...
```

Figure 8 : Comportement étape 3

## Activité 5

- 1) A l'aide de QtCreator, créer un projet nommé « zero », de type Application Qt graphique qui crée l'application graphique minimale.

La classe de la fenêtre principale se nommera Principale, et vous cochez l'option « Generate Form ».

- 2) Compiler. Exécuter

Constaté qu'elle comporte les caractéristiques décrites dans les diapositives ci-dessous, et **mémoriser** les noms de ces composants afin d'être capables de les nommer correctement plus tard.

- 3) Repérer où se trouvent les fichiers sources, et quel est le contenu de chacun d'entre eux.

3.- -  
WIMP - Window, Icon, Menu, Pointer (5/15)

★ □ Window - Fenêtre

- Fenêtre **primaire / principale**

Dite **standard** si elle contient les 10 composants **principaux** suivants :

- 1 poignée de redimensionnement
- 1 barre de titre (généralement, le titre d'une fenêtre primaire est le titre de l'application elle-même)
- 1 menu système
- 3 boutons de minimisation / maximisation / fermeture
- 1 barre de menu / d'actions
- 2 barres de défilement (horizontal / vertical)
- 1 zone client

Elle peut aussi contenir les composants **complémentaires** suivants :

- des barres d'outils
- 1 barre d'état

DUT Informatique - M2105 Introduction aux IHM - 4 - Interactions graphiques 7

3.- -  
WIMP - Window, Icon, Menu, Pointer (6/15)

□ Window - Fenêtre

- Application **SDI** : l'UNIQUE fenêtre est **primaire / principale**

DUT Informatique - M2105 Introduction aux IHM - 4 - Interactions graphiques 17