# R2.02 : développement d'application avec IHM
# Feuille TD n° 2

## Création manuelle d'une interface graphique simple
## Création d'un premier programme événementiel graphique

***Objectifs*** *:*

1.- *Qt – Découverte des premières classes graphiques simples*
2.- *Associer un comportement aux widgets de l'interface graphique*

## Activité : Création de l'application convertisseur de températures v0

### Sujet :

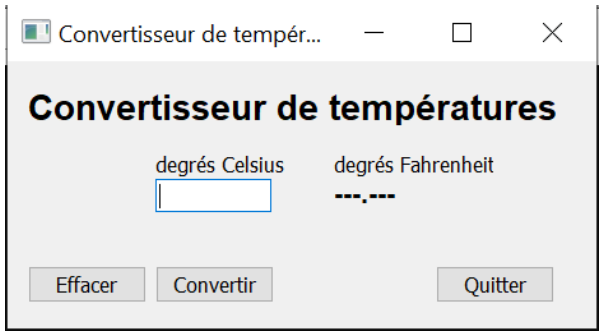
Créer un convertisseur de températures simple :

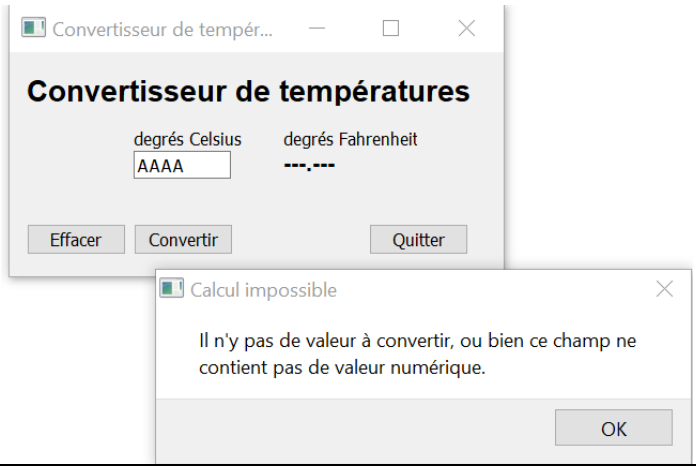
Etant donnée une température entrée en degrés Celsius, le programme l'affiche en degrés Fahrenheit.

Le calcul est lancé grâce au bouton « Convertir ».
Le bouton « Effacer » efface le contenu de la zone de texte et réinitialise le résultat avec "---.---".
On quitte l'application grâce au bouton « Quitter ».

La fenêtre a pour titre : « Convertisseur de températures – v0 »

Lorsque le calcul n'est pas possible, une boîte de Dialogue indique la possible raison.




### Objectif de l'Activité :

L'exercice consiste à créer l'application **sans utilisation d'un générateur d'interfaces** : vous devrez écrire toutes les lignes de code de ce programme, y compris celles créant les objets graphiques par instanciation des classes graphiques nécessaires.

### 1. Étape 1 : Type de fenêtre et identification des objets graphiques qui la composent

Les objets graphiques visibles dans la zone client de la fenêtre principale seront des **composants** de cette fenêtre principale.

a) Pour chaque objet graphique présent sur cette interface, donner les noms en français et en anglais.
Exemple : 1 barre de titre – Title bar

b) Identification du type de la fenêtre principale :
- La fenêtre a-t-elle besoin d'une barre de Titre ?
- La fenêtre a-t-elle besoin d'une barre de Menus ?
- La fenêtre a-t-elle besoin d'une barre de Statut ?

- La fenêtre a-t-elle besoin de barres de défilement ?
- Est-ce que la fenêtre de l'application est une fenêtre de dialogue ?

**Liens :**
eLearn / Ressources IHM /chapitre IHM-4_ElémentsGraphiques

## 2. Étape 2 : Création du projet

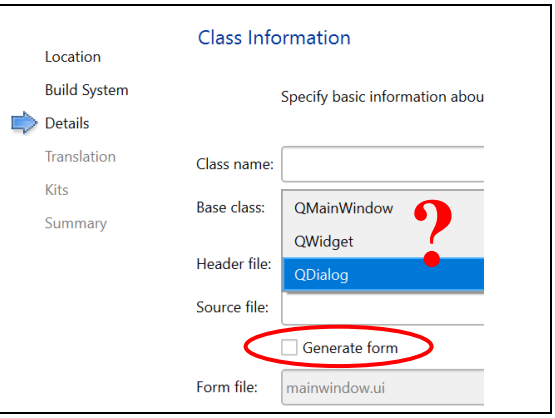
a) Dans votre répertoire td2, **créez** avec l'assistant de QtCreator, créer un projet **Qt graphique** (de type « **Application Qt – Qt Widgets Application** »)

b) Nom de la classe de la fenêtre principale (et unique) de l'application : **`ConvertisseurTemp`**

c) Type de la classe de base de la fenêtre principale de l'application :

La classe de la fenêtre à créer peut hériter de 3 classes graphiques différentes : QMainWindow / QWidget / QDialog

Choisir la classe de base en fonction des caractéristiques de la fenêtre identifiées à l'étape précédente.




d) Vous NE cocherez PAS l'option « Generate Form ».

## 3. Étape 3 : Déclaration des variables correspondant aux objets graphiques

a) Quels noms donnerez-vous aux variables du programme correspondant aux différents objets graphiques ? Comparez votre stratégie de nommage à celle de vos voisins.

b) Dans quel fichier faut-il déclarer les objets graphiques composant la fenêtre principale ?

c) Chercher les classes Qt correspondant à ces éléments. **Liens :** https://doc.qt.io/qt-5/classes.html

d) Écrire les déclarations des variables **après avoir ajouté** la bibliothèque QWidgets en début de fichier.

```
#include <QtWidgets> // inclut tous les widgets de Qt5
```

e) Compiler seulement. (au fait, pourquoi se contenter de compiler uniquement ?)

## 4. Étape 4 : Création des composants graphiques et paramétrage (placement, taille, …)

Les objets sont créés par instanciation des classes graphiques.

a) Dans quel fichier / zone du fichier / sous-programme faudra-t-il écrire les instructions d'instanciation ?

Une fois les objets créés, il s'agit de les paramétrer, notamment selon la taille et la position :

b) Paramétrer les objets selon les indications ci-dessous :

- Etiquette Intitulé : police Arial 14 Gras, positionEtTaille (x, y, largeur, hauteur) : (16, 20, 430, 30). Le texte est cadré à gauche dans l'espace occupé par l'étiquette
- Etiquette « degrés Celsius » : (116,70,101,20)
- Etiquette « degrés Fahrenheit » : (256,70,125,20)
- Champ de texte mono-ligne : (116,92,91,26)
- Etiquette contenant le résultat en degrés Fahrenheit : Arial, 11, Normal, (256, 93, 63, 20)
- Bouton « Effacer » : (16, 160, 93, 29)
- Bouton « Convertir » : (116, 160, 93, 29)
- Bouton « Quitter » : (336, 160, 93, 29)

**Liens :**
https://doc.qt.io/qt-5/classes.html
https://doc.qt.io/qt-5/qwidget.html
https://doc.qt.io/qt-5/qwidget.html#setGeometry-1 : positionnement et taille des objets graphiques
https://doc.qt.io/qt-5/qfont.html

c) Compiler – exécuter. Vérifier la présence, taille et placement des objets graphiques.

## 5. Étape 5 : Décrire le comportement de l'application dans sa version actuelle

Pour ce faire, **compléter le tableau ci-dessous**, où les fonctionnalités sont séparées en deux catégories :

- Fonctionnalités '**métier'** : celles correspondant au but de l'application.
- Comportement associé aux composants graphiques utilisés.

| Fonctionnalités 'métier' | Comportement lié aux éléments d'interface |
|---|---|
| – Convertir : NON<br>– …<br>– … | – Fenêtre re-dimensionnable, déplaçable, iconisable' ? :<br>OUI, mais ….<br>– Déplacement entre éléments graphiques avec touche tabulation ?<br>– … |

## 6. Étape 6 : Description des fonctionnalités 'métier'

a) **Compléter le tableau ci-dessous** en indiquant, pour chaque fonctionnalité métier, quel est l'événement qui la déclenche.

Si l'événement est déclenché par un objet graphique de l'interface, préciser de quel objet graphique il s'agit.

| Fonctionnalités 'métier' | Evénement – objet graphique |
|---|---|
| – Convertir<br>– …<br>– … | – … |

b) - En déduire les **signals**, **slots** et méthodes 'classiques' **de la classe ConvertisseurTemp**
**-** Dans quel fichier les déclarez-vous ?

## 7. Étape 7 : Écrire les déclarations des signals, slots et méthodes

Ne pas oublier les buts….

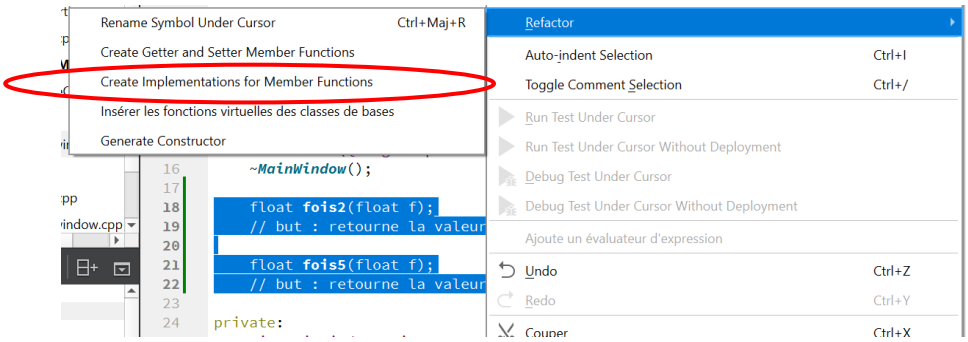
## 8. Étape 8 : Écrire et tester les définitions des slots et méthodes

**Développer dans l'ordre :**

- Fonctionnalité Quitter
- Fonctionnalité Effacer
- Fonctionnalité Convertir

En testant chaque fonctionnalité une par une.

A. **Générer automatiquement tous les corps vides des slots et méthodes :**
- Sélection des déclarations des méthodes, puis
- clique-droit / Refactor / Create Implementations for Member Functions




Résultat :

Les définitions sont créées, vides, dans le fichier .cpp

Du coup, vous pouvez compiler.

```
16  float MainWindow::fois2(float f)
17  {
18
19  }
20
21  float MainWindow::fois5(float f)
22  {
23
24  }
```

Vous écrirez ensuite les définitions **une par une**, en testant chaque sous-programme une fois écrit.
**Développer dans l'ordre :** Fonctionnalité Quitter, Fonctionnalité Effacer, Fonctionnalité Convertir

**B. Fonctionnalité Quitter.**
Pas de consigne particulière : à vous de faire !

**C. Fonctionnalité Effacer.**

= Écrire et tester le slot **`void effacer()`** associé au clic du bouton « **bEffacer** »

Cette action sera découpée en quatre étapes :
- Connecter le signal et le slot
- Tester le **déclenchement** du slot lorsque le signal a lieu
- Écrire le code complet du slot
- Tester le **fonctionnement** du slot développé (= la fonctionnalité Effacer)

- Connecter le signal et le slot : dans le fichier **`convertisseurtemp.cpp`**

```
QObject::connect(bEffacer, SIGNAL(clicked()), this, SLOT(effacer()));
```

- Tester le **déclenchement** du slot **`effacer()`** lorsque le bouton « **bEffacer** » a été cliqué

Le corps du slot **`effacer()`** contient juste un message indiquant qu'il réagit bien au signal :

```
void ConvertisseurTemp::effacer()
{
    qDebug() << "J'efface" << Qt::endl;
}
```

avec
#include <QDebug> en début de fichier **`convertisseurtemp.cpp`**

- Écrire le code complet du slot **`effacer()`**

**Liens / Aides** :
https://doc.qt.io/qt-5/qlineedit.html
https://doc.qt.io/qt-5/qlabel.html#text-prop

Règle des Noms des getters/setters des propriétés d'une classe QWidget
Exemple : classe `QLabel` - propriété `text` – getter : `text()` – setter : `setText()`

- Tester le **fonctionnement** du slot développé : Pas de consigne particulière

## D. Fonctionnalité Convertir.

= Écrire et tester le slot **`void convertir()`** associé au clic du bouton « **bConvertir**»

Cette action sera découpée en sept étapes :
- Connecter le signal et le slot
- Tester le **déclenchement** du slot lorsque le signal a lieu
- Écrire les sous-programmes nécessaires au slot
- Écrire un code partiel du slot : utiliser la classe QDebug pour afficher le message d'erreur
- Tester le code (partiel) développé
- Modifier le corps du slot pour écrire le message d'erreur conformément aux spécifications
- Tester le code complet développé

**Liens / Aides :**
https://doc.qt.io/qt-5/classes.html

- Utilisation de la classe QMessageBox pour la création d'un message
  https://doc.qt.io/qt-5/qmessagebox.html
  https://doc.qt.io/qt-5/qmessagebox.html#details
  https://doc.qt.io/qt-5/qmessagebox.html#setWindowTitle
  https://doc.qt.io/qt-5/qmessagebox.html#text-prop
  https://doc.qt.io/qt-5/qmessagebox.html#standardButtons-prop
  https://doc.qt.io/qt-5/qmessagebox.html#StandardButton-enum

- Transformation d'une chaîne QString en nombre décimal (float) :
  `float QString::toFloat(bool& indicateur) ;`

- Transformation d'un nombre décimal (float) en chaîne de caractères QString :
  `QString& QString::setNum(float n, char format='g',int precision = 6) ;`

This is an overloaded function.

Sets the string to the printed value of *n*, formatted according to the given *format* and *precision*, and returns a reference to the string.

The formatting always uses QLocale::C, i.e., English/UnitedStates. To get a localized string representation of a number, use QLocale::toString() with the appropriate locale.

**See also** number().

The *format* can be 'e', 'E', 'f', 'g' or 'G'

Argument Formats

In member functions where an argument *format* can be specified (e.g., arg(), number()), the argument *format* can be one of the following:

| Format | Meaning |
|---|---|
| e | format as [-]9.9e[+\|-]999 |
| E | format as [-]9.9E[+\|-]999 |
| f | format as [-]9.9 |
| g | use e or f format, whichever is the most concise |
| G | use E or f format, whichever is the most concise |

A *precision* is also specified with the argument *format*. For the 'e', 'E', and 'f' formats, the *precision* represents the number of digits *after* the decimal point. For the 'g' and 'G' formats, the *precision* represents the maximum number of significant digits (trailing zeroes are omitted).