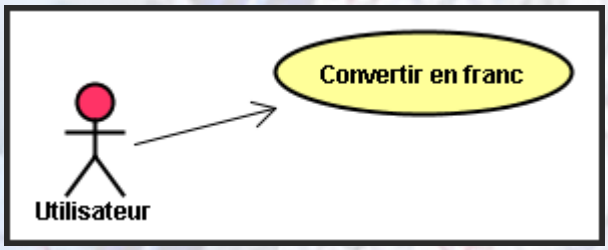
Ce troisième projet nous permet d'écrire nos premières lignes de code en mode Graphique. Une grande partie de la programmation en mode fenêtrée consiste à renseigner spécifiquement les gestions d'événements. C'est ce que nous faisons ici. Dès que l'utilisateur clique sur le bouton de conversion (ou appuie sur le bouton « Entrée »), nous demandons le traitement de conversion en franc d'une valeur saisie en €uro. Ce projet nous permet :

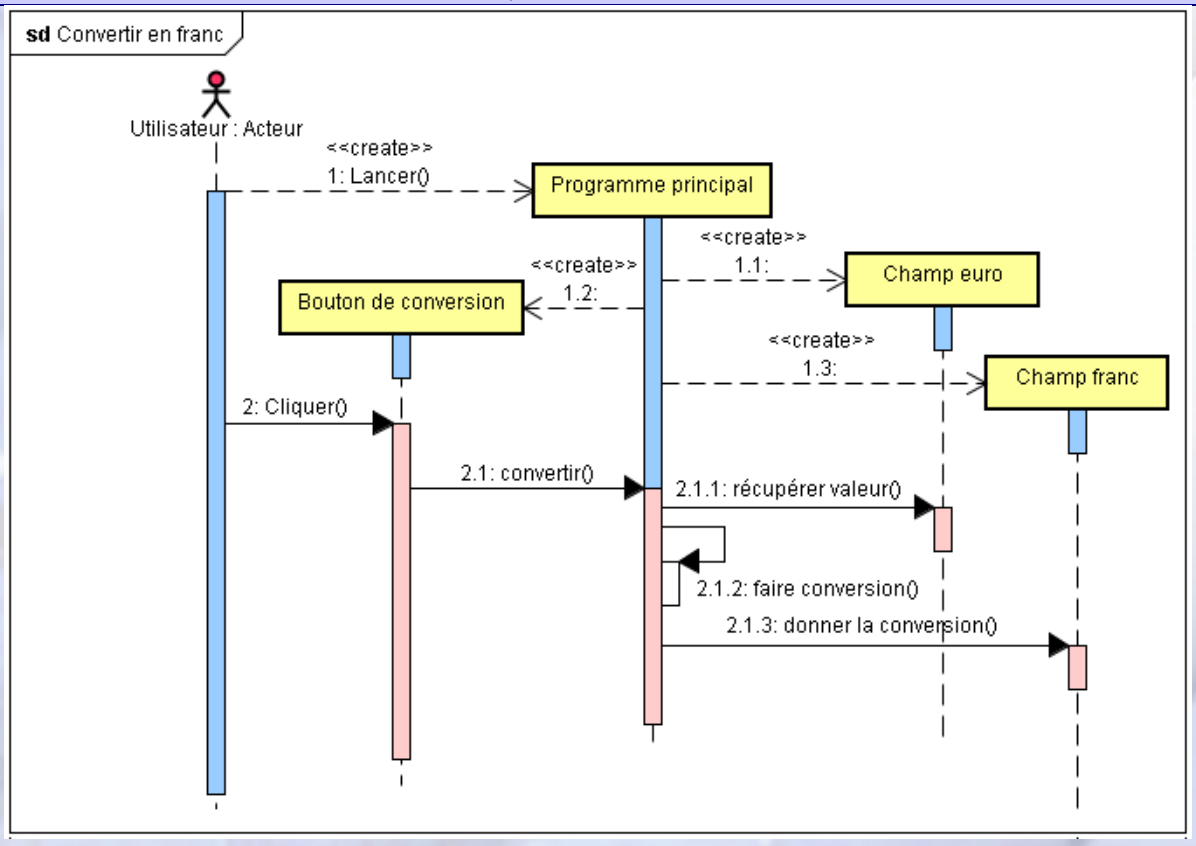
- *De mettre en œuvre une gestion d'événement personnalisée.*
- *De découvrir les fonctionnalités très poussées des zones de saisie prévues pour les réels (**QDoubleSpinBox**) qui permettent, entre autre, de placer des préfixes ou des suffixes supplémentaires.*

***Nota** : ce projet nous permet d'écrire nos premières lignes de code en relation avec une **IHM** (**I**nterface **H**omme **M**achine).*

MODÉLISATION - DIAGRAMME DE CAS D'UTILISATION

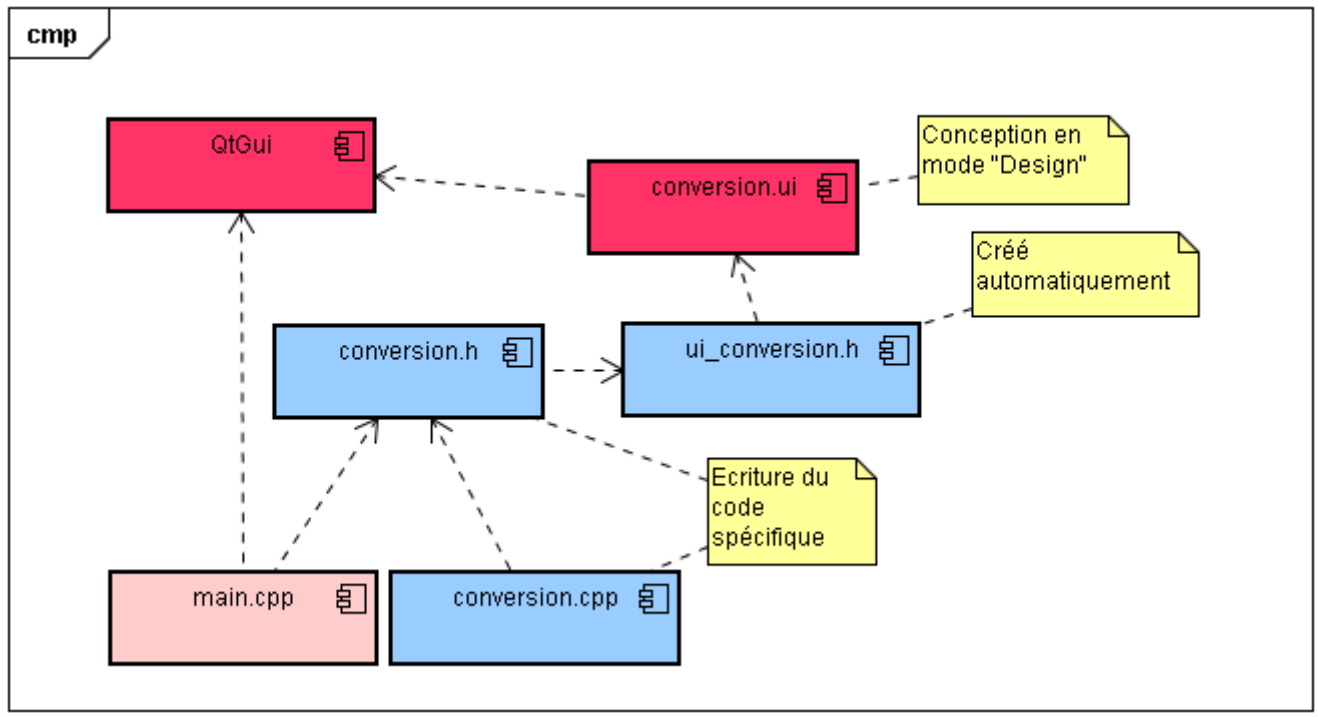


MODÉLISATION – DIAGRAMME DE SÉQUENCE DE CONVERTIR EN FRANC



MODÉLISATION – DIAGRAMME DE COMPOSANTS

Le diagramme de composant permet de décrire l'architecture d'une application en terme de composants logiciels. En d'autres termes, nous pouvons spécifier à l'aide de ce schéma l'ensemble des fichiers sources concernés par l'application globale.



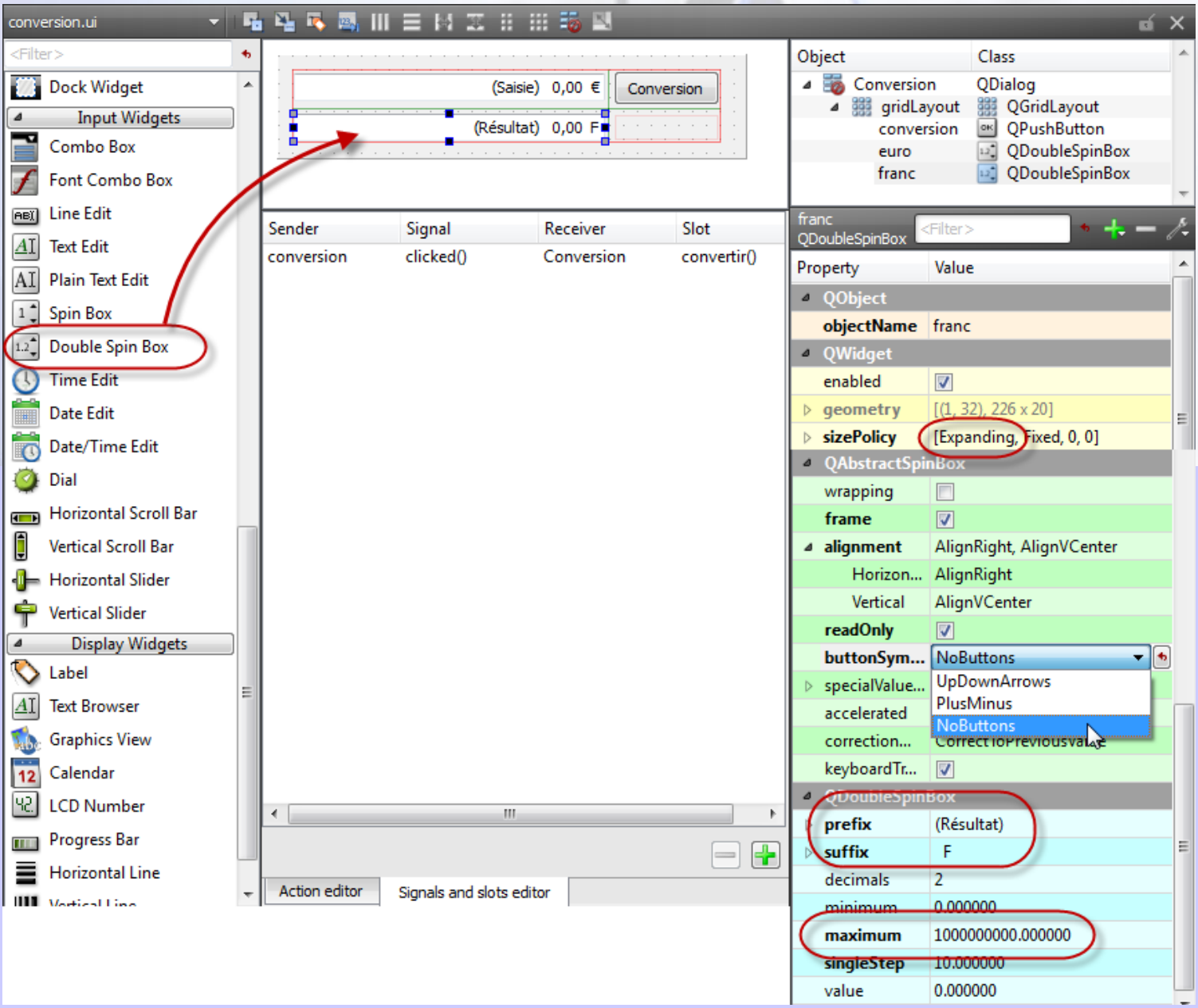
- ***QtGui*** *: est une bibliothèque qu'il est nécessaire de prendre en compte lorsque nous désirons développer en mode* ***Graphique IHM****. Cette bibliothèque contient tous les objets que nous retrouvons constamment dans une application fenêtrée classique : Fenêtre, bouton, boîte de liste, zone de saisie, etc.*
- ***conversion.ui*** *: ce fichier est la partie graphique intuitive de la conception d'une IHM, en plaçant les composants directement à l'aide de la souris, et en proposant des réglages adaptés au travers des différentes propriétés. La gestion d'événement s'effectue également à ce niveau là.*
- ***ui_conversion.ui*** *: lorsque nous exécutons le programme, tous les réglages effectués dans le fichier précédents sont pris en compte pour générer automatiquement ce fichier. Ce fichier correspond à l'écriture équivalente en langage C++ (codage), sous forme de classe et d'objets, de toutes les éléments graphiques qui ont été placés à l'aide de la souris.*
- ***conversion.h*** *: Ce fichier « en-tête » donne la déclaration de la fenêtre principale sous forme de classe. Ce fichier, devra être modifié pour déclarer un nouveau traitement d'événement personnalisé.*
- ***conversion.cpp*** *: Ce fichier donne la définition des méthodes (ce quelles doivent réaliser) qui ont été déclarées dans le fichier « en-tête » précédent. Ce fichier, devra également être modifié pour définir le nouveau traitement d'événement souhaité, dans notre cas, prévoir la conversion des €uros vers des francs.*
- ***main.cpp*** *: Ce fichier réalise très peu de chose. Il se contente de créer la fenêtre principal de l'application, ici* ***Conversion****, et lance l'exécution du programme (Toutes les spécificités du programme sont décrites dans la classe* ***Conversion****).*

CONVERSION.UI

La conception de ce projet en mode « Design » est relativement rapide puisque nous avons déjà acquis pas mal d'expérience dans ce domaine. Tout porte sur les réglages particuliers des objets représentant les €uros et les francs. Ces objets sont de type ***QDoubleSpinBox***.

Cette classe est extrêmement performante puisqu'elle est capable d'interpréter des valeurs réelles saisies au clavier. Il est possible de régler finement l'aspect visuel de ce composant, en choisissant, par exemple, le nombre de chiffre après la virgule. Vous pouvez demander à inclure également un préfixe et/ou un suffixe qui entoure la valeur réelle. Ce composant dispose sur sa partie droite de petites flèches intégrées qui permettent de passer automatiquement à la valeur suivante ou à la valeur précédente. Ces flèches peuvent toutefois être cachées.

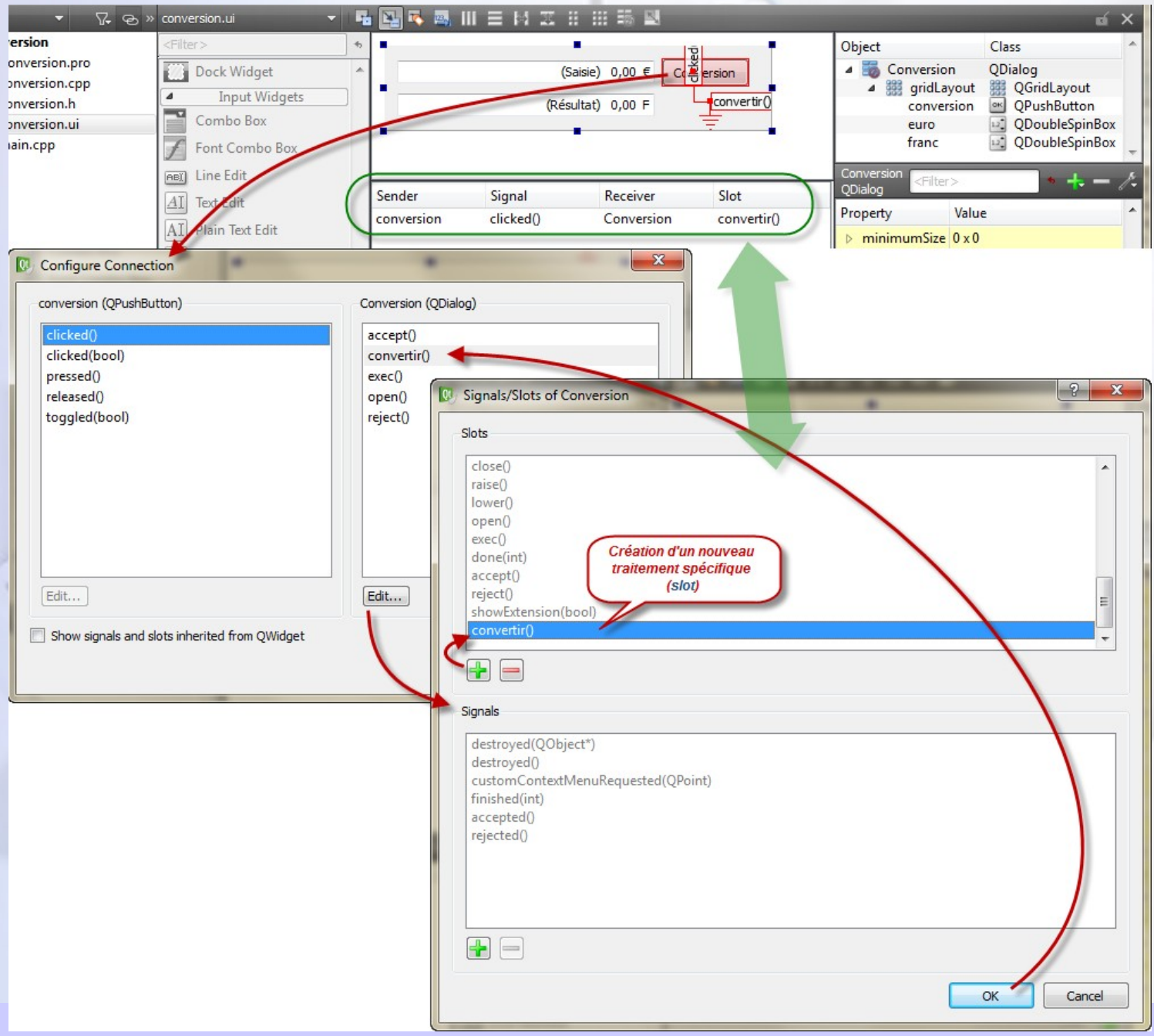
*Le placement de tous les composants dans la fenêtre est automatique grâce à la disposition **gridLayout**. C'est très intéressant, mais le problème, c'est que tous les composants disposent alors de la même largeur. Il est donc souhaitable de préciser que les objets **€uro** et **franc** prennent le plus de largeur possible, pour que le bouton de conversion ai juste la place d'écrire son intitulé, finalement dans sa taille par défaut. Cela se fait au travers de la propriété **sizePolicy**, et plus précisément de l'élément interne **Horizontal Policy**. Nous demandons à ce que le composant s'étende au maximum en horizontal en proposant la constante **Expanding**.*



GESTION ÉVÉNEMENTIELLE PERSONNALISÉE

La gestion événementielle personnalisée se fait comme nous avons l'habitude de procéder. Nous devons toutefois ajouter une nouvelle méthode personnelle (c'est vous qui choisissez le nom) qui va s'occuper du traitement à réaliser.

*Ainsi, nous demandons à lancer une nouvelle méthode **convertir()**, qui effectue le traitement de conversion entre les €uros et les francs, et qui s'exécute à chaque fois que l'utilisateur clique sur le bouton de **Conversion**.*



x CONVERSION.H

Pour prendre en compte cette nouvelle méthode ***convertir()*** qui va réaliser le traitement souhaitée, vous devez au préalable la déclarer à l'intérieur de la classe qui réceptionne l'événement, c'est-à-dire la classe ***Conversion***.

> *Vous spécifiez cette déclaration dans une zone particulière, appelée « **private slots** », qui s'occupe uniquement des méthodes de gestion événementielle. Ce type de méthode ne doit pas comporter de retour (**void**) et ne possède généralement pas d'argument. Par ailleurs, une déclaration de méthode consiste à mettre son nom et à ponctuer la fin de la méthode par un point-virgule.*

conversion.h* — Conversion

```
1   #ifndef CONVERSION_H
2   #define CONVERSION_H
3
4   #include <QtGui/QMainWindow>
5
6   namespace Ui
7   {
8       class Conversion;
9   }
10
11  class Conversion : public QMainWindow
12  {
13      Q_OBJECT
14  public:
15      Conversion(QWidge *parent = 0);
16      ~Conversion();
17  private:
18      Ui::Conversion *ui;
19  private slots:
20      void convertir();
21  };
22
23  #endif // CONVERSION_H
```

Lignes de code à rajouter.

Déclaration d'une nouvelle méthode qui s'appelle convertir() sans retour et sans argument.

CONVERSION.CPP

Pour terminer, tout le traitement de conversion souhaité se définit dans le fichier qui porte l'extension « ***cpp*** ». Il faut mettre en œuvre ce que nous appelons une définition de méthode. Effectivement, après avoir déclarée la méthode dans le fichier en-tête, nous devons préciser tout ce qu'elle doit faire en écrivant le traitement à l'intérieur des accolades, comme pour une fonction ***main()***.

*Lorsque vous définissez une méthode, vous devez également préciser à quelle classe elle appartient. Il faut alors écrire en préfixe de la méthode, le nom de la classe suivi de l'opérateur de portée « **::** »*

conversion.cpp — <Select Symbol>

```
1   #include "conversion.h"
2   #include "ui_conversion.h"
3
4   Conversion::Conversion(QWidget *parent)
5       : QMainWindow(parent), ui(new Ui::Conversion)
6   {
7       ui->setupUi(this);
8   }
9
10  Conversion::~Conversion()
11  {
12      delete ui;
13  }
14
15  void Conversion::convertir()
16  {
17    const double TAUX = 6.55957;
18    double e = ui->euro->value();
19    double f = e * TAUX;
20    ui->franc->setValue(f);
21  }
```

Vous redéfinnissez ensuite tout le traitement que la méthode devra réaliser pour effectuer la conversion.

Classe qui contient la méthode

Objet représentant la fenêtre principale de l'application.

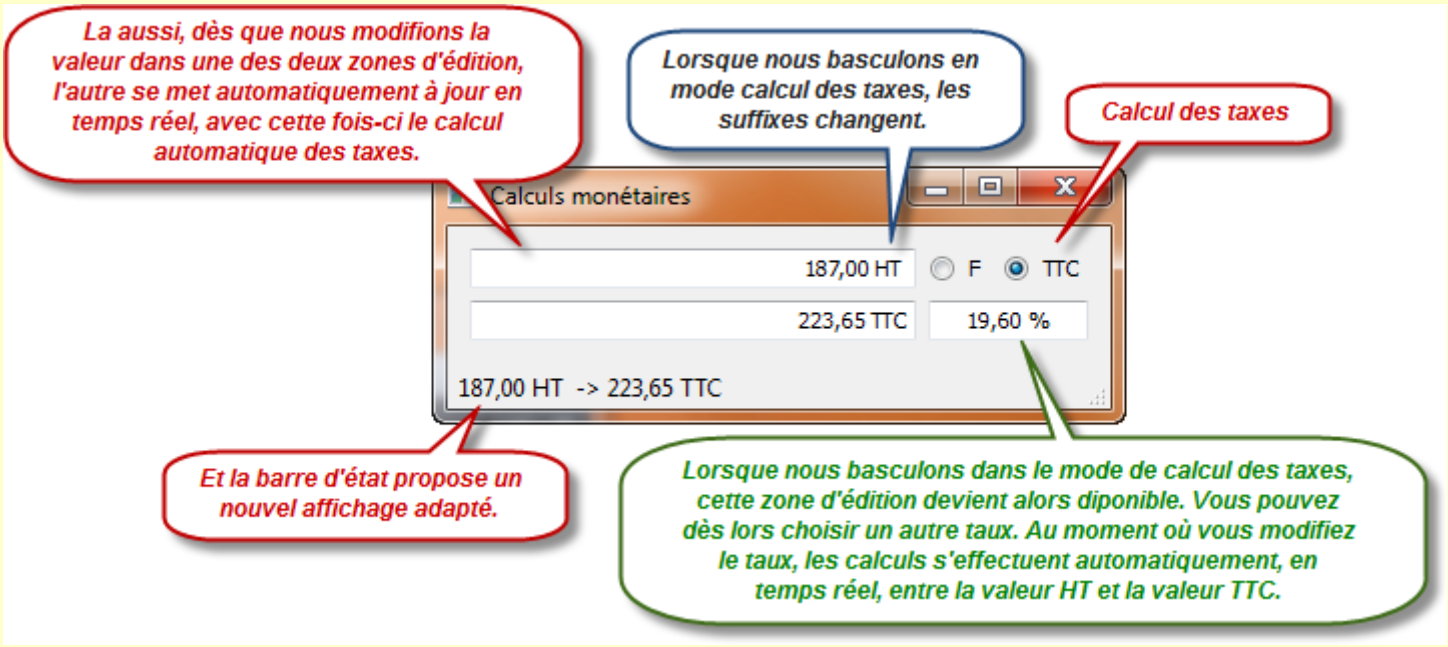
x TRAVAUX PRATIQUES EN AUTONOMIE

Vous allez maintenant valider vos nouvelles connaissances en gestion événementielle personnalisée au travers d'un projet. Le but de ce projet est de bien maîtriser le composant de saisie des réels ***QDoubleSpinBox***, et de traiter de façon fine l'ensemble des traitements adaptés aux interventions de l'interlocuteur.

*Nous profiterons de l'occasion pour créer des méthodes de traitement d'événement qui possèdent des paramètres[1] qui vont servir à récupérer directement les valeurs réelles saisie par l'utilisateur. Nous en profitons également pour mettre en place une barre d'état à l'aide de la classe **QStatusBar**[2]. Finalement, cet unique projet propose deux fonctionnalités différentes :*

x D'abord la mise en œuvre d'un convertisseur entre les €uros et les francs, cette fois-ci, dans les deux sens.

x De rajouter ensuite le calcul d'une valeur monétaire avec ou sans les taxes. Il est également possible de choisir son taux.



1 ***QDoubleSpinBox** possède un signal **valueChanged(double)** avec un paramètre de type **double**. Ce paramètre correspond à la valeur réelle saisie. Il est donc possible de créer une méthode de traitement d'événement qui possède le même type de paramètre qui va ainsi servir au calcul spécifique à réaliser.*

2 ***QStatusBar** possède une méthode **showMessage()**, bien utile, qui permet d'afficher le message souhaité dans la barre d'état.*