4

Premiers programmes (avec Qt Widgets)

Ce chapitre a pour but d'aborder Qt 5 au travers de trois exemples d'application en C++ : deux mini-applications d'introduction et un client graphique pour un service web. Certains concepts importants du framework seront introduits dans cette partie, mais il ne s'agit pas d'une présentation exhaustive, plutôt d'un premier contact.

Le prérequis pour une bonne compréhension du chapitre est une connaissance de base du langage C++. La connaissance d'un autre langage avec des classes d'objets et une syntaxe similaire, comme Java, est suffisante pour une appréhension générale du code. Pour mettre en œuvre les exemples, il faut avoir au préalable installé Qt 5.

L'interface graphique de ces applications utilise le module Qt Widgets, et ce chapitre peut donc aussi faire office d'introduction à ce module. Qt Widgets, ainsi que les principes liés à QObject, sont des éléments issus de Qt 4 qui ont une grande maturité. Ils sont déjà traités par une littérature abondante, à laquelle nous vous invitons à vous reporter pour plus d'informations ; ils ne seront pas davantage détaillés dans ce livre centré sur Qt 5.

Si vous avez déjà pratiqué Qt 4 ou une version précédente mais pas Qt 5, vous pourrez trouver dans ce chapitre des illustrations de l'usage de fonctions nouvelles, ainsi qu'une introduction à Enginio, un service web disposant de fonctions d'intégration aux applications Qt.

4.1. Bonjour Qt

Note › *Le code de cet exemple est disponible dans le projet* `hello`.

S'agissant d'un premier contact, la règle est de se présenter. C'est ce que va faire notre première application, constituée de deux (petits) fichiers, que vous pouvez créer avec votre éditeur de texte préféré.

Le fichier main.cpp

```
#include <QApplication>
#include <QMainWindow>
#include <QLabel>

int main(int argc, char *argv[])
{
  QApplication a(argc, argv);

  QMainWindow w;
  QLabel hello ( "Bonjour, je suis une application Qt !", &w );
  w.show();

  hello.setAlignment(Qt::AlignCenter);
  hello.resize(w.size());

  return a.exec();
}
```

Nous commençons par créer un objet de type QApplication, qui va se charger de tout le travail d'initialisation et de la gestion de base d'une application graphique basée sur Qt Widgets.

L'objet w est la fenêtre principale de l'application. QMainWindow est un widget Qt complexe, mais ici nous nous contentons de le créer avec ses caractéristiques par défaut.

L'objet hello est un QLabel, qui est un autre type de widget Qt, un des plus simples, puisque qu'il se contente d'afficher un texte (mais il le fait bien). Son premier paramètre définit son contenu, le deuxième paramètre son *widget parent*. Le widget parent va contenir visuellement ses widgets enfants, mais va aussi être responsable de leur destruction le cas échéant.

La relation hiérarchique parent-enfants, qui peut s'étendre sur plusieurs niveaux (générations ?), est en fait héritée de la classe QObject, dont QWidget dérive. Un des intérêts de cette relation est de faciliter la gestion de la mémoire (les parents détruisant automatiquement leurs enfants). La gestion parent-enfants est une des nombreuses fonctions de QObject, une des classes les plus importantes de Qt, et dont dérivent une bonne partie de autres classes de l'API.

Les instructions `setAlignment` et `resize` sont purement cosmétiques, elles centrent le contenu du QLabel dans la fenêtre au moment de son ouverture.

Pour finir, `a.exec()` lance la boucle d'événements principale de Qt, qui va permettre l'affichage et les interactions avec l'utilisateur.

Note *› Une boucle d'événements est un mécanisme de messagerie à l'intérieur d'un programme. La structure de contrôle associée est une boucle (d'où le nom) qui attend les événements pour les transmettre à la fonction qui va les traiter. La boucle d'événements principale de Qt, lancée par `exec()`, traite les événements générés par le système (souvent par l'interface graphique) ainsi que ceux générés par l'application elle-même.*

Le fichier hello.pro

```
QT       += core gui widgets

TARGET = hello
TEMPLATE = app

SOURCES += main.cpp
```

Ce fichier décrit le contenu de votre projet à l'outil de gestion de projets intégré à Qt et va lui permettre de générer les bonnes configurations des compilateurs et éditeurs de liens pour créer votre exécutable (voir aussi chapitre Aller plus loin avec Qt Creator).

La ligne commençant par `QT` déclare les modules de Qt que vous utilisez. Ici, nous utilisons l'interface graphique Qt Widgets, et donc en plus de `core` nous déclarons `gui` et `widgets`.

Les lignes `TARGET` et `TEMPLATE` décrivent respectivement le titre et le type du projet.

La ligne `SOURCES` liste les fichiers sources, soit ici notre fichier `main.cpp`.

Le code source de notre application est maintenant complet, il reste à créer un exécutable. Vous pouvez le faire en ligne de commande ou en utilisant Qt Creator.

Avec Qt Creator, ouvrez `hello.pro` (Ctrl+O), cliquez sur CONFIGURER LE PROJET, puis Ctrl+R pour l'exécuter.

En ligne de commande, tapez `qmake && make` sous Linux et Mac OS. Sous Windows `make` est à remplacer par `mingw32-make` ou `nmake` suivant que vous utiliserez MinGW ou MSVC.