5

Communication entre C++ et QML

Dans le chapitre précédent, nous avons vu les principes essentiels et simples des interfaces graphiques en QML. Cela est suffisant pour comprendre le QML et jouer avec, mais peut s'avérer rapidement insuffisant si vous devez développer des applications conséquentes avec beaucoup de données et parfois connectées à tout un tas d'autres briques comme par exemple l'utilisation de bibliothèques C++ bas niveau. Vous pouvez par exemple avoir une bibliothèque C++ capable de lire des données provenant d'un capteur C++. Ensuite, vous créez une application C++/Qt qui dépend de cette bibliothèque. Une fois les données lues et stockées dans vos conteneurs C++, vous souhaitez peut-être les filtrer, les trier pour enfin les visualiser dans l'interface graphique en QML. La question se pose de savoir comment envoyer ces données du C++ au QML. C'est ce que j'explique dans ce chapitre, de l'envoi d'une simple valeur à l'envoi de listes de données, filtrées et triées.

5.1. Contexte QML

Lorsque l'on développe avec Qt une application C++/QML, il faut indiquer le point d'entrée du fichier QML (souvent appelé `main.qml` par analogie avec le fichier `main.cpp`, point d'entrée du programme C++). Les composants QQuickView ou QQmlApplicationEngine sont les deux alternatives permettant de spécifier l'entrée QML.

Cet exemple, repris de la documentation de Qt, montre comment démarrer une application QML à partir d'une QQuickView.

```
int main(int argc, char *argv[])
{
  QGuiApplication app(argc, argv);

  QQuickView *view = new QQuickView;
  view->setSource(QUrl::fromLocalFile("myqmlfile.qml"));
  view->show();
  return app.exec();
}
```

Cet extrait provient du livre Qt 6 en pratique
écrit par Mériadeg Perrinel - © 2025 Éditions D-BookeR

Après avoir déclaré une application de type QGuiApplication et avant d'exécuter sa boucle d'événements (voir Section 1.3, Création d'un Hello World via Qt Creator et CMake), une vue de type QQuickView est déclarée. Il suffit alors de lui indiquer l'URL du fichier d'entrée QML à utiliser, puis de l'afficher avec la méthode `show`.

L'autre façon de faire, vue également dans le premier chapitre, utilise le composant QQmlApplicationEngine et sa méthode `load`.

```
QQmlApplicationEngine engine;
const QUrl url(u"qrc:/Qt6Book/main.qml"_qs);
engine.load(url);
```

QQUICKVIEW OU *QQMLAPPLICATIONENGINE* ?

Le composant QQmlApplicationEngine, plus récent que le composant QQuickView, est doté de fonctionnalités pratiques. Par exemple, il connecte automatiquement son signal `quit` au même signal de l'application, impliquant aussi la possibilité de fermer correctement et simplement l'application depuis une page QML. Par ailleurs, il permet de charger automatiquement les fichiers de traduction situés dans le répertoire `i18n`, s'il est placé juste à côté du fichier QML principal. Il met en place un incubateur qui permet de charger de manière asynchrone les composants QML, évitant ainsi des freezes de l'application QML. Enfin, c'est la seule façon d'utiliser un fichier QML qui utilise un composant Window et de surcroît un composant ApplicationWindow comme élément racine de l'arbre QML, ce qui est le cas de l'application Qt6BookApp accompagnant ce livre.

Le composant QQuickView ne permet de charger une page QML que si son Item racine hérite de QQuickItem.

Pour une application de type bureau avec un menu, une barre d'outils ou une barre d'état, je recommande d'utiliser le composant ApplicationWindow comme composant d'entrée du fichier QML principal, ce qui implique l'utilisation du QQmlApplicationEngine côté C++ pour charger le fichier d'entrée QML. À partir du QQmlApplicationEngine instancié, il va être possible d'établir les signaux, les types à faire connaître au QML mais aussi d'envoyer toutes sortes de variables et d'objets.

Cet extrait provient du livre Qt 6 en pratique
écrit par Mériadeg Perrinel – © 2025 Éditions D-BookeR

5.2. Envoi de valeur simple

En plus de déterminer le point d'entrée du fichier QML, il est possible de préparer des variables simples depuis le C++ et de les envoyer au QML.

```
// main.cpp
QString qt6BookName  = "Qt6BookApp";
BookMainApplication::setApplicationName(qt6BookName);
QQmlApplicationEngine engine;
auto context = engine.rootContext();
context->setContextProperty("_qt6BookName", qt6BookName);

// main.qml
ApplicationWindow {
  title: _qt6BookName
}
```

Par exemple, la méthode statique `setApplicationName` de QApplication sert à définir le nom de l'application. Si vous souhaitez attribuer ce même nom au titre de l'application, qui se définit côté QML, il faut envoyer sa valeur du C++ au QML. Pour cela, il suffit de créer une variable QML contenant le nom de l'application avec la méthode `setContextProperty` appliquée sur l'instance de QQmlApplicationEngine. Cette méthode a plusieurs signatures, mais pour envoyer une copie d'un objet simple, il faut utiliser la signature prenant en compte un QVariant en deuxième paramètre. Cela signifie que le type de l'objet que vous souhaitez envoyer doit être compatible avec un QVariant. C'est le cas ici, puisque la valeur qt6BookName envoyée est de type QString.

Note › *Une QString C++ sera transformée en type string côté QML. De nombreux types primitifs C++ ou basiques Qt ont leur équivalent en QML. On parle alors de QML Value Types car ces valeurs C++ sont utilisées pour créer des valeurs QML. Un objet passé par référence du C++ au QML est vu comme un QML Object Type.*

La liste des QML Value Type est détaillée dans la documentation de Qt 6. Certains types sont légèrement plus avancés comme quarternion, vector2D ou matrix4x4 et sont fournis par des modules QML (en l'occurrence Qt Quick).

5.3. Envoi d'objet avec Q_GADGET

La section précédente indique comment envoyer une valeur simple du C++ au QML et uniquement dans ce sens. Dans cette section, nous allons voir comment définir un objet en C++ qui n'est pas un QObject et l'envoyer au QML, qui peut alors questionner ses propriétés. Voici un exemple d'objet C++ qui illustre cela :

Cet extrait provient du livre Qt 6 en pratique
écrit par Mériadeg Perrinel – © 2025 Éditions D-BookeR