PROCESSUS DE CODAGE D’UNE APPLICATION C++/Qt/QML

**1/ Définition des prototypes dans le fichier HEADER (\*.h)**

Le fichier doit débuter avec :   
#ifndef nom\_du\_fichier.h   
#define nom\_du\_fichier.h   
  
et se terminer par : #endif   
  
on peut alors l’intégrer à le fichier c++:   
#include "nom\_du\_fichier.h"   
  
Il doit être absolument mit dans ton dossier où est situer ton projet

Le Header WManager.h:

#ifndef WMANAGER\_H

#define WMANAGER\_H

#include <QObject>

// #include <QQuickWidget>

#include <QQuickView>

#include <QGraphicsScene>

#include <QGraphicsView>

#include <QQmlEngine>

#include <QQmlContext>

#include <QDebug>

#include <QStringList>

#include <vector>

Déclaration des fonctions Q\_INVOKABLE   
Déclaration des propriétés Q\_PROPERTY

1. Q\_INVIKABLE *typeFonction nomFonction (paramètre1, paramètre2, paramètre n, …)*
2. Q\_PROPERTY (*typePropriété nomPropriété* READ *nomPropriétéLecture\_r* WRITE *SetNomPropriétéEcriture* NOTIFY *nomProprieteChanged)*

(-r, Set et Changed sont des conventions non obligatoire mais utiles et + lisibles)

**2/ Définition du champ d’application des variables et fonction dans le fichier HEADER (\*.h)**  
Les types de section ne sont pas unique et peuvent regrouper des déclarations thématique ou correspondant à un besoin précis

**Public:** sont déclarées dans cette section, les variables persistantes, accessible de toutes les fonctions de l’application

Ex : QStringList *nomObjet* ;

**Protected:**

**Signals:** Déclaration de fonctions de changement d’une propriété

**Public slots:**

**Private :**

**3/ Binding (liaison) entre le C++ et le QML dans le fichier WManager.cpp**  
3.1 Initialisation des valeurs pour le QML

Ex : *setPropString("");*

3.2 Initialisation des Modèles pour le QML **même vide**

Ex : *updateQML\_model (« nomMoldel », listeVide) ;*

Chargement du fichier QML (url du fichier) et affichage des objets

*view->setSource( QUrl::fromLocalFile(fichierQML) ) ;  
view->show();*

3.3 Exécution d’une action c++ à partir d’une fonction appelée par le QML

*void WManager::sendActionToCpp(QString nomAction, QString parametre)*

// on initialise le paramètre d’écriture

*setPropString("");*

// on test l’événement de l’utilisateur

*if ( nomAction == "ajoutEntier" )*

*{*

// Détail d'une action appelée à partir du C++

*sl << parametre;*

*updateQML\_model("UnModelARenseigner", sl);*

*qDebug() << "WManager::sendActionfromQML : entierSaisi " << parametre;*

*}*

*else if ( nomAction == "supprimeEntier" )*

*{*

*if (sl.size() >= 1)*

*{*

*sl.pop\_back();*

*updateQML\_model("UnModelARenseigner", sl);*

// Détail d'une autre action appelée à partir du C++

*}*

*else {*

*setPropString("liste vide");*

*}*

*}*

3.4 Rajout d’une propriété du Context pour le QML

*m\_propString = a;*

*qDebug() << "setPropString" << a;*

*emit propStringChanged();*

Ex: affiche liste vide si on tente de supprimer un element d’une liste vide

3.4 Rajout d’une propriété du Context pour le QML

*Ex : Lit la zone de texte*

**4/ Déclaration et gestion des objets QML dans le(s) fichier(s) \*.qml**

**Exemple :** on veut afficher et gérer deux boutons, une zone de saisie et une liste et une zone de d’affichage (gestion des erreurs de saisie).

* Le premier bouton permet à l’utilisateur d’ajouter un entier dans la liste via la zone de saisie (contrôle de saisie inclus).
* Le second bouton permet de supprimer un élément de la liste si la liste n’est pas vide.

On créer des modèles utilisés par le binding

Ex : *UnModelARenseigner*  qui sera géré par le C++ dans le fichier WManager au travers de la fonction *sendActionToCpp*

On identifie les objets par des identifiant id (ex : *id : idTextSaisi*)  
On envoie les Context au c++ selon l’intervention de l’utilisateur: *onClicked*

**Exemples :**

* on ajoute un entier à la liste: *Context.sendActionToCpp("ajoutEntier", idTextSaisi.text)*
* on supprime un entier: *Context.sendActionToCpp("supprimeEntier", idTextSaisi.text)*
* on prévient l’utilisateur via message ajouter dans la zone d’affichage : *Context.propString = "Saisir un entier"*

On utilise le rôle modeldata pour transmettre le contenu de la liste au C++

*ListView{*

*y : 200*

*width : 400*

*height : 400*

*model: UnModelARenseigner*

*delegate : Rectangle {*

*width : 80*

*height : 25*

*border.width: 1*

*border.color: "blue"*

*Text{*

*x : 10*

*width : parent.width*

*height : parent.height*

*text : modelData // mot-clef Qt à retenir*

*verticalAlignment: Text.AlignVCenter*

*}*

*}*

}