Introduction à la Programmation d'Interfaces Utilisateurs Graphiques ***avec Qt*** *(et C++)*

C. Ernst,
Y. Hmamouche
L. Lakhal

Mars 2019

1

Plan

- ***Qt*** et les interfaces graphiques
- l'approche ***Qt Widgets***
- un programme ***Qt*** basique (1er exemple)
- structure d'un programme ***Qt*** à interface graphique (2ème exemple)
- plus loin avec les widgets : signaux et slots (3ème exemple)
- intégration des données d'un fichier dans un programme ***Qt*** à interface graphique
- le projet

2

Les boîtes à outils pour le développement d'IHM

Qt ←	multiplateforme	C++
GTK+	multiplateforme	C
MFC puis WTL	Windows	C++
WPF (⊂ WTL)	Windows	(langages .Net)
FLTK	multiplateforme	C++
AWT / Swing	multiplateforme	Java
Cocoa	MacOs	Objective C
Gnustep	Linux, Windows	Objective C
Motif	Linux	C
JQuery UI	Web	Javascript

Ces boîtes à outils comportent
- une bibliothèque d'objets interactifs (les « widgets ») que l'on assemble pour construire l'interface désirée
- des fonctionnalités permettant la programmation évènementielle

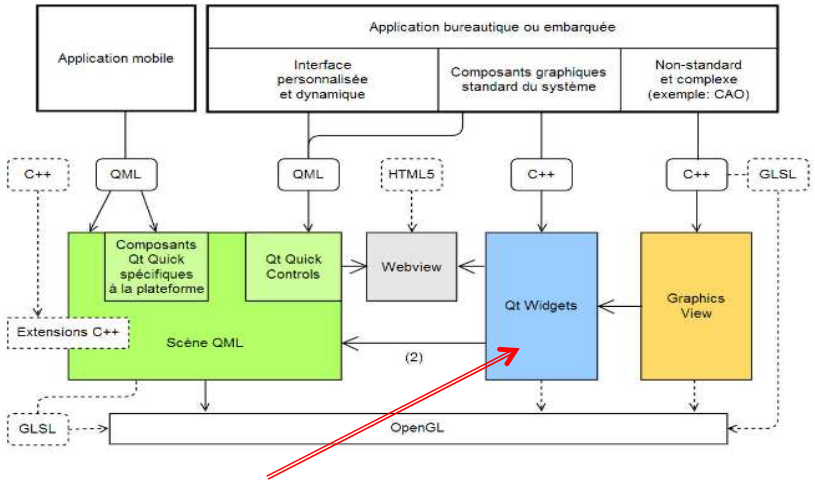
3

Que sait faire (globalement) *Qt* ?

- *Qt* est basé autour d'un modèle d'objets :
  - Son architecture est entièrement fondée sur la classe QObject et l'outil **moc** (Meta Object Compiler)
- En dérivant des classes de QObject, on obtient un certain nombre d'avantages :
  - Gestion facilitée de la mémoire
  - Signaux et Slots
  - Propriétés des objets → c'est uniquement du C++ standard avec quelques macros (d'où la portabilité)

4

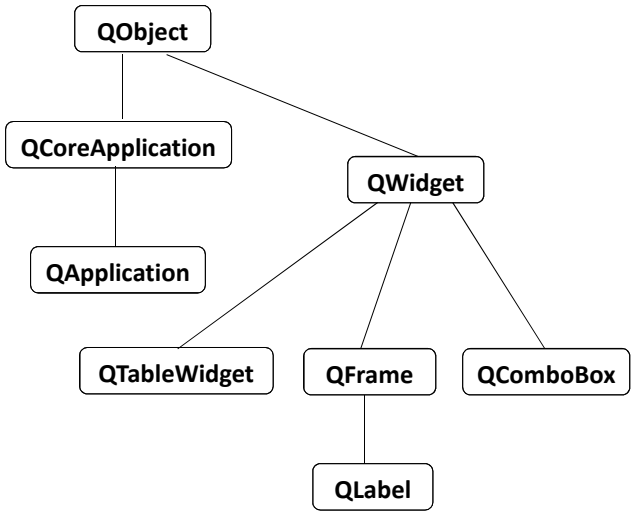
Qt et les GUIs …



Approche utilisée (manuelle, sans QtDesigner)

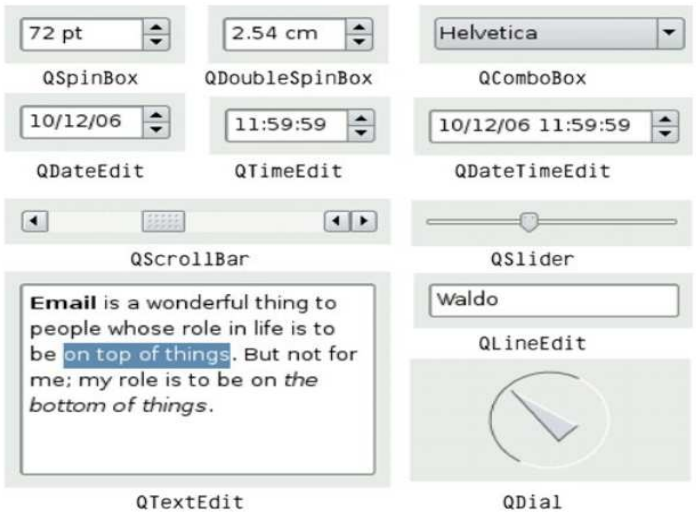
5

Arbre simplifié des principales classes *Qt*



6

… et d'autres classes *Qt* (*input* widgets)



"Le" programme *Qt* de base

Sous *Qt* , sélectionner Nouveau Projet
puis Autre Projet
et enfin Projet Vide

Ensuite, sélectionner Nouveau Fichier
puis C++ et Fichier Source. Le nommer
Main.cpp, et y saisir le code suivant :

Main.cpp

```
#include <QApplication>

int main(int argc, char* argv[])
{
    QApplication app(argc, argv);
    return app.exec();
}
```

Compiler et exécuter

Un programme *Qt* avec un widget

Un objet défini via son constructeur

Main.cpp

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

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

    QLabel label("Vive le module ECBD! Et tous ses disciples");
    label.show();

    return app.exec();
}
```

Invocation d'une méthode pour visualiser l'objet

Pour fonctionner, ajouter Qt += widgets dans le .pro

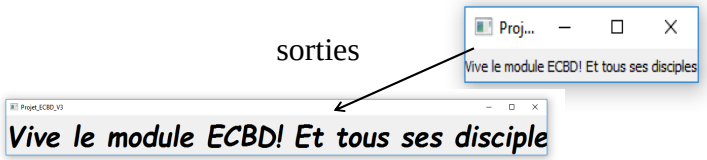
Le premier programme *Qt* "amélioré"

Modification d'une propriété du widget

Main.cpp

```
#include <QApplication>
#include <QLabel>
#include <QFont>

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

    QLabel label("Vive le module ECBD! Et tous ses disciples");
    label.setFont(QFont("Comic Sans MS", 40, QFont::Bold, true));
    label.show();

    return app.exec();
}
```

sorties



Premier exemple : problématique

- On vient de créer une « boîte » contenant du texte; automatiquement *Qt* a créé une fenêtre autour car on ne peut pas avoir une boîte qui « flotte » seule à l'écran
- On n'a pas directement prise sur cette fenêtre (taille, position, …). Et surtout, comment faire pour ajouter d'autres boîtes dans cette fenêtre? Et …

➡ Utilisation de widgets **conteneurs**

11

Premier exemple : améliorations

Un widget peut en contenir d'autres :

ainsi, pour étendre l'exemple, une fenêtre (un *QWidget*) va contenir non seulement notre label mais également un bouton poussoir (un *QPushButton*) ayant pour objectif de permettre de sortir du programme

12

Premier exemple : amélioration 1

Main.cpp

```
#include <QApplication>
#include <QLabel>
#include <QFont>
#include <QPushButton>

int main(int argc, char* argv[])
{
    QApplication app(argc, argv);
    QWidget MaFenetre;
    MaFenetre.setFixedSize(800,600);

    QLabel label("Vive le module ECBD!", &MaFenetre);
    label.setFont(QFont("Comic Sans MS", 20, QFont::Bold, true));
    label.move(100, 200);

    QPushButton bouton(“Quitter”, &MaFenetre);
    bouton.setGeometry(600, 400, 80, 40);

    MaFenetre.show();
    return app.exec();
}
```

taille de la fenêtre

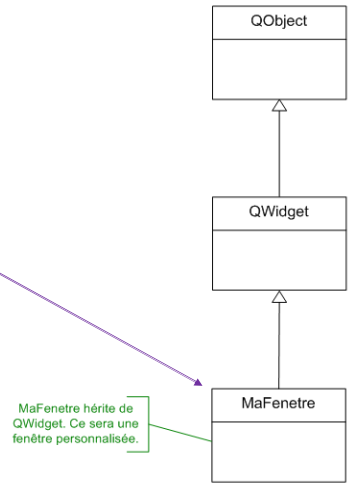
fenêtre "parent" des widgets créés

13

Premier exemple : amélioration 2 (A)

Dans la pratique (lorsque l’on développe des GUI en POO), on crée, pour représenter une fenêtre, une nouvelle classe héritant de QWidget

Nouvelle classe, donc nouveau module : créer MaFenetre.h et MaFenetre.cpp, puis ajouter ces fichiers dans le .pro



14

Premier exemple : amélioration 2 (B)

MaFenetre.h

```
#include <QWidget>
#include <QLabel>
#include <QPushButton>

class MaFenetre : public QWidget
{
    public :
        MaFenetre();
    private :
        QLabel *m_lab;
        QPushButton *m_bou;
};
```

MaFenetre.cpp

```
#include “MaFenetre.h”

MaFenetre::MaFenetre() : QWidget()
{
    setFixedSize(800,600);

    m_lab = new QLabel (
       "Vive le module ECBD!", this);
    m_lab->setFont(QFont("Helvetica",
              20, QFont::Bold, true));
    m_lab->move(100, 200);

    m_bou = new QPushButton(
                    “Quitter”, this);
    m_bou->setGeometry(600,400,80,40);
}
```

Main.cpp

```
#include <QApplication>
#include “MaFenetre.h”

int main(int argc, char* argv[])
{
    QApplication app(argc, argv);
    MaFenetre fenetre;
    fenetre.show();
    return app.exec();
}
```

IMPORTANT : il convient de ne pas désallouer les widgets (dans le destructeur, car fait par *Qt*)

15

Premier exemple : amélioration 2 (C)

Ce nouveau programme fonctionne correctement, sauf que … rien ne se passe lorsque l’on clique sur ***Quitter***

Ceci est plutôt logique, puisque rien n’a été explicitement prévu pour …

On va donc s’intéresser maintenant aux mécanismes permettant de gérer les évènements pouvant se produire au sein d’une fenêtre

16

Signaux et Slots *Qt* : définitions

Un signal : c'est un message envoyé par un widget lorsqu'un évènement se produit

Exemple : on a cliqué sur un bouton

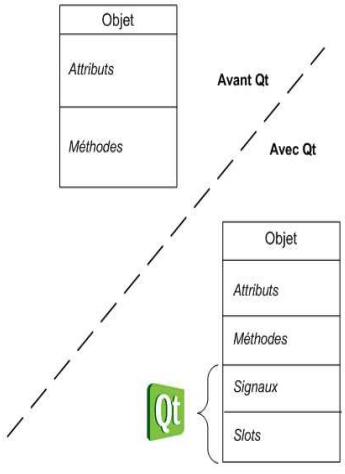
Un slot : c'est la fonction qui est appelée lorsqu'un évènement s'est produit. On dit que le signal appelle le slot. Concrètement, un slot est une méthode d'une classe

Exemple : le slot quit() de la classe QApplication provoque l'arrêt du programme

17

Signaux et Slots *Qt* : spécificités

Les signaux et les slots sont considérés par *Qt* comme des éléments d'une classe à part entière, en plus des attributs et des méthodes

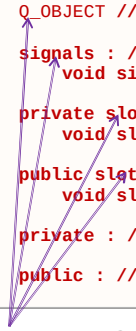


18

Signaux et Slots *Qt* : mise en œuvre

MaClasse.h

```
class MaClasse : public Q_Classe_Dérivant_de_QObject
{
    Q_OBJECT // Macro OBLIGATOIRE

    signals : // Signaux définis manuellement, non examiné ici
        void signal1( int, const char * );

    private slots : // Slots privés définis manuellement
        void slot1( … );

    public slots : // Slots publics définis manuellement
        void slot2( … );

    private : // Méthodes et attributs privés

    public : // Méthodes et attributs publics
};
```



Les macros *Qt* et l'implémentation des signaux usuels sont gérés par le **moc** (Meta Object Compiler)

19

Signaux et Slots *Qt* : fonctionnement

- Toutes les classes héritant de QObject peuvent contenir des signaux et des slots
- *Qt* intègre un grand nombre de slots prédéfinis, mais il est possible de créer ses propres slots (et signaux)
- Le mécanisme des signaux et slots est « type-safe », et les erreurs de typage sont reportées comme avertissements lors de la compilation
- La méthode **QObject::connect()** permet d'associer un slot à un signal

20

Signaux et Slots sur l'exemple

MaFenetre.h

```
#include <QWidget>
#include <QLabel>
#include <QPushButton>

class MaFenetre : public QWidget
{
    Q_OBJECT // Macro OBLIGATOIRE

    public slots :
        void setQuitter();

    public :
        MaFenetre();

    private :
        QLabel *m_lab;
        QPushButton *m_bou;
};
```

MaFenetre.cpp

```
#include "MaFenetre.h"

MaFenetre::MaFenetre() : QWidget()
{
    setFixedSize(800,600);

    // … code déjà défini

    m_bou->setGeometry(600,400,80,40);

    connect(m_bou, SIGNAL(clicked()),
            this, SLOT(setQuitter()));
}

void MaFenetre::setQuitter()
{
    exit(0);
}
```

Association du signal clicked(), lorsque déclenché sur le PushButton, au slot setQuitter()

21

Deuxième exemple : objectifs

- Ajouter à l'existant une liste déroulante (une QComboBox) de chaînes de caractères représentant des couleurs, et en vérifier le fonctionnement en affichant une trace de l'élément sélectionné (dans cette liste)
- Améliorer certains points de l'existant

22

Deuxième exemple : mise en œuvre (.h)

MaFenetre.h

```
#include <QWidget>
#include <QMainWindow>
#include <QLabel>
#include <QPushButton>
#include <QComboBox>
#include <QString>

class MaFenetre : public QMainWindow
{
    Q_OBJECT // Macro OBLIGATOIRE

    public slots :
        void setQuitter();
        void setCouleur();

    public :
        MaFenetre(QWidget *parent = 0);
    private :
        QLabel *m_lab, *m_tra;
        QPushButton *m_bou;
        QComboBox *m_com;
        QString couleur;
};
```

mieux que QWidget

pour l'héritage

la sélection de liste

la liste, à remplir avec des QString

23

Deuxième exemple : mise en œuvre (.cpp)

MaFenetre.cpp

```
#include "MaFenetre.h"

MaFenetre::MaFenetre(QWidget *parent) : QMainWindow(parent)
{
    setFixedSize(800,600);
    m_bou = new QPushButton("Quitter", this);
    m_bou->setGeometry(600,400,80,40);

    m_lab = new QLabel("Couleur", this);
    m_lab->setFont(QFont("Arial", 12, QFont::Bold, true));
    m_lab->move(320, 125);

    m_com = new QComboBox(this);
    m_com->setGeometry(300,150,100,30);
    m_com->addItem("Rouge");
    m_com->addItem("Vert");
    m_com->addItem("Bleu");
    m_com->addItem("Jaune");
    m_com->addItem("Orange");
    m_com->addItem("Violet");

    m_tra = new QLabel(this);
    m_tra->setFont(QFont("Arial", 12, QFont::Bold, true));
    m_tra->move(320, 300);

    connect(m_bou, SIGNAL(clicked()), this, SLOT(setQuitter()));
    connect(m_com, SIGNAL(currentIndexChanged(const QString &)),
                                        this, SLOT(setCouleur()));
}
```

24

Deuxième exemple : mise en œuvre (.cpp)

MaFenetre.cpp (suite)

```
void MaFenetre::setQuitter()
{
    exit(0);
}

void MaFenetre::setCouleur()
{
    couleur = m_com->currentText();
    m_tra->setText(">> " + couleur + " <<");
}
```

A chaque fois qu'un élément est sélectionné dans la liste, son index courant est mis à jour pour « pointer » sur cet élément ; il suffit alors d'utiliser la méthode currentText() pour récupérer la « valeur » de cet élément, puis de l'afficher en invoquant la méthode setText() sur le widget m_tra

25

Troisième (et dernier) exemple : objectifs

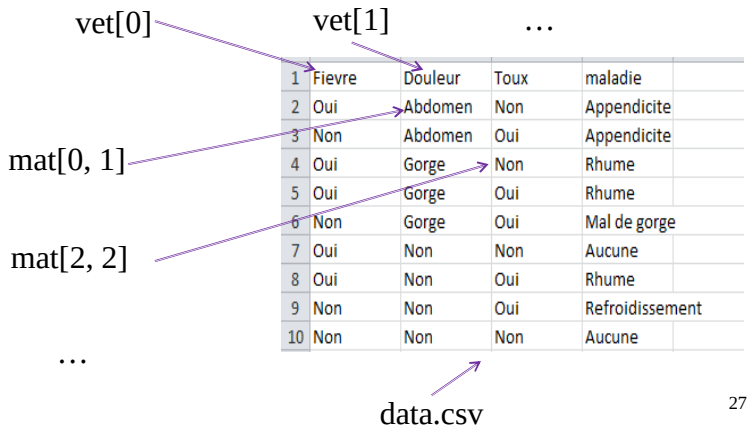
- Il **vous faut** maintenant remplir la liste non plus de façon statique, mais avec les éléments d'une colonne d'un fichier (*data.csv*), à placer dans le répertoire de votre exécutable
- On vous fournit les "éléments" de chargement des données de ce fichier (fichiers charger_csv.h et charger_csv.cpp, à ajouter dans votre .pro)
- Ces éléments consistent en 2 types et une fonction :

```
typedef vector <string> CVString;
typedef vector <vector<string>> CMatString;
void read_csv (CMatString&, CVString&, const string&);
```

26

Troisième exemple (mise en œuvre 1)

L'appel à read_csv (mat, vet, "data.csv");
doit produire le résultat suivant :



1	Fievre	Douleur	Toux	maladie
2	Oui	Abdomen	Non	Appendicite
3	Non	Abdomen	Oui	Appendicite
4	Oui	Gorge	Non	Rhume
5	Oui	Gorge	Oui	Rhume
6	Non	Gorge	Oui	Mal de gorge
7	Oui	Non	Non	Aucune
8	Oui	Non	Oui	Rhume
9	Non	Non	Oui	Refroidissement
10	Non	Non	Non	Aucune

27

Troisième exemple (mise en œuvre 2)

Pour que cela fonctionne, ajoutez 2 données membres dans MaFenetre.h, et appelez la fonction dans le constructeur de MaFenetre.cpp

MaFenetre.h

```
#include …

class MaFenetre : public QMainWindow
{
        …
    private :
        …
        CVString m_vet;
        CMatString m_mat;
};
```

MaFenetre.cpp

```
#include …

MaFenetre::MaFenetre …
{
    …
    read_csv (m_mat, m_vet, "data.csv");
    …
};
```

28

Troisième exemple (mise en œuvre 3)

Les données du .csv maintenant mémorisées dans votre programme, il suffit de remplir la liste déroulante avec les données d'une de ses colonnes (et de définir convenablement son en-tête)

Réaliser ce traitement (écrire pour cela une fonction membre privée recevant en paramètre le numéro de la colonne du .csv dont les éléments doivent remplir la liste)

29

Troisième exemple (mise en œuvre 4)

Vous devriez constater la présence de doublons dans la liste …

Reprendre le traitement en éliminant ces doublons, et en faisant en sorte que la valeur "NULL" soit la première des valeurs affichées (même si elle n'existe pas dans le fichier …)

30

Projet : état des lieux

Vous êtes fin prêts pour le projet …, connaissant

- les rudiments de gestion d'une fenêtre principale et de quelques widgets dans une application *Qt*
- le mécanisme de récupération des données d'un fichier .csv dans ce même programme

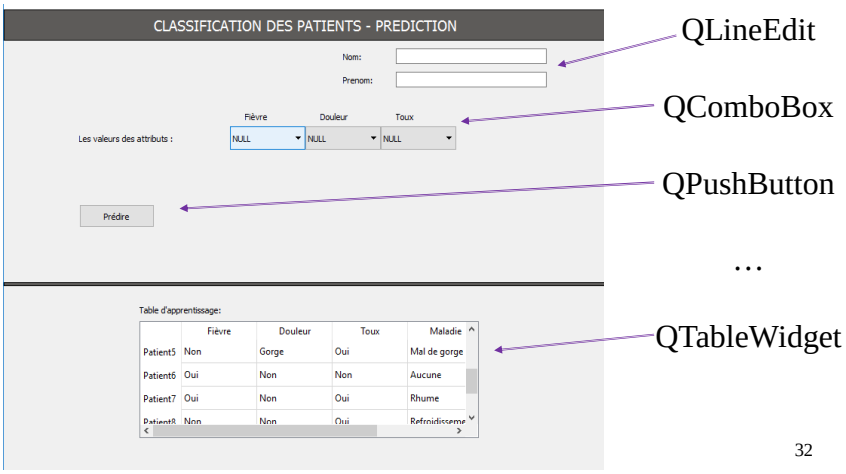
Pour manipuler d'autres widgets, documentez vous

- directement sur le Web (sous Google, tapez QT + QTableWidget pour la gestion d'une table …)
- en ligne sous *Qt* via l'assistant *Qt* d'Aide

31

Projet : objectif

Une interface possible, non "contractuelle"



32

Projet : objectif++

Vous pouvez vous fixer des objectifs plus ambitieux (même si non demandés) :

- Gestion dynamique de l'interface (fonction du nombre de colonnes du fichier)
- Sauvegarde des informations saisies pour les nouveau patients et intégration dans les calculs à venir
- …

A vous de jouer…

33