TP – Premières applications fenêtrées

Désormais, la programmation est de type évènementielle !

- ⇨ Commençons par créer une fenêtre simple

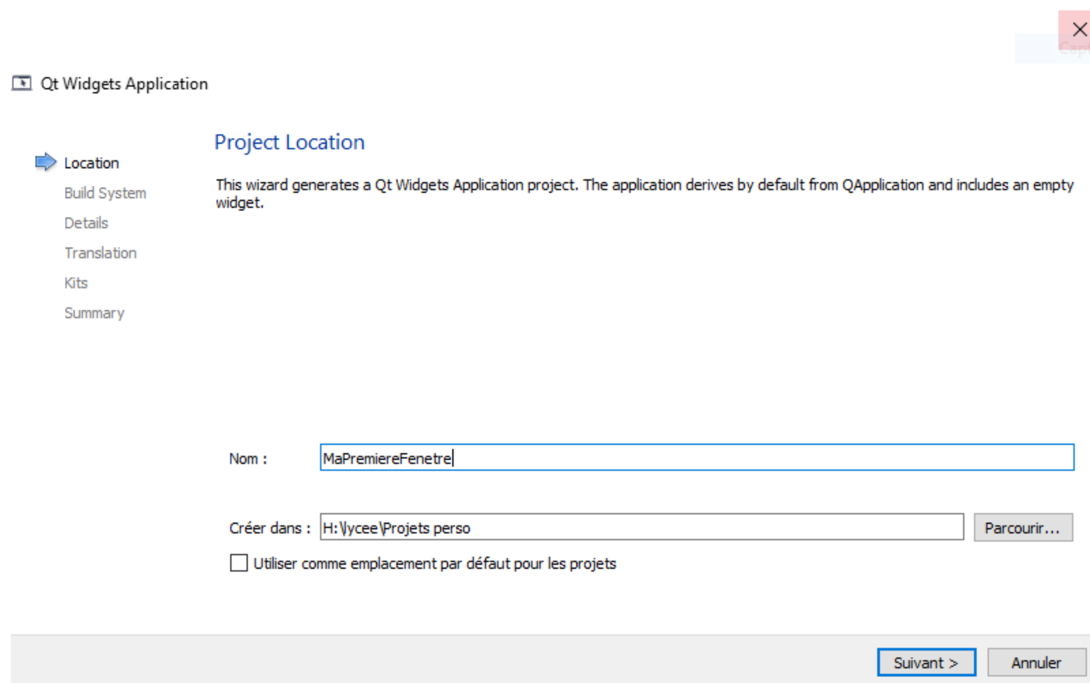
Création d'une application Qt

Remarque : les captures écran ont été faites à partir de QtCreator et Qt 5.9.

Lancer QtCreator : c:\Qt\Qt5.8.0\tools\qtcreator\bin\qtcreator.exe

Créer un nouveau projet Qt en cliquant sur *Nouveau Projet/Application Qt avec Widgets*

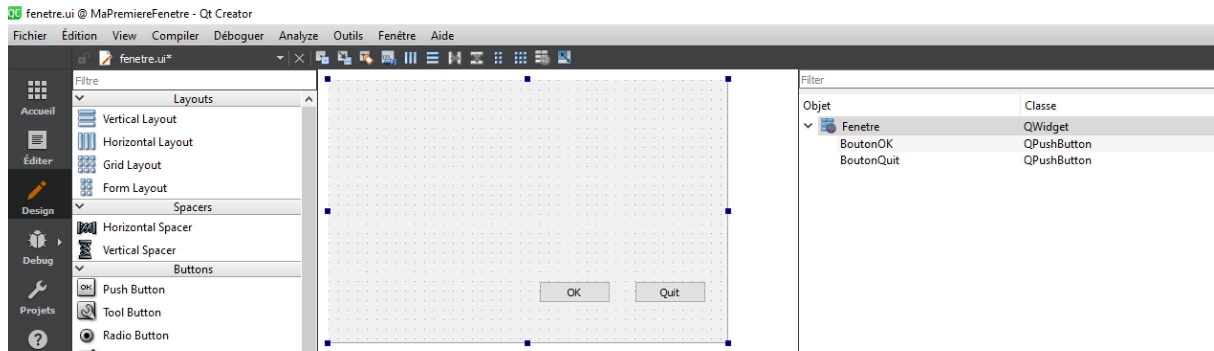
Appelez le *MaPremiereFenetre :*



Vous pouvez créer, pour commencer, une application *QWidget* : c'est l'application la plus simple possible.

Vous devez donner un nom à votre **classe** (nouveau type à l'instar des structures). Cette classe va symboliser votre fenêtre : appelez la *Fenetre* (c'est original !!)

Lancer Qt Designer en double cliquant sur le fichier *fenetre.ui* et ajouter 2 boutons « OK » et « Quit ». Donnez leur des noms d'objets facilement repérables dans le code ; par exemple : BoutonOk et BoutonQuit.



Remarquez : le nom de l'objet, par exemple BoutonOK et son type (classe) QPushButton. il y a donc deux objets issus de la même classe.

Attention : Ne pas confondre le nom de l'objet bouton et le texte qui est affiché dedans !

Vous pouvez compiler et exécuter votre programme : il affiche la fenêtre avec ses deux boutons. Mais, bien entendu il ne se passe rien lorsqu'on clique sur les boutons : il faut mettre du code à exécuter au bout. Cela passe par les signaux et slots.

Ajout de signaux et slots

On veut exécuter une fonction lorsqu'on clique sur le bouton « OK ».

Cette fonction s'appelle un slot en Qt. Il faut donc l'ajouter dans la classe qui représente mon application.

Dans *fentre.h*, ajouter le slot correspondant au clic du bouton dans la classe:

```
class Fenetre : public QWidget
{
    Q_OBJECT

public:
    Fenetre(QWidget *parent = nullptr);
    ~Fenetre();

private slots :
    void onCliqueBouton();

private:
    Ui::Fenetre *ui;

};
```

Ajouter la déclaration du slot

Ici l'application exécutera le slot (fonction) lorsque le bouton émettera le signal *clicked*

ATTENTION : *private slots* N'EST PAS DU C++ !!!

On n'aurait pu aussi mettre une section public slots

Ajouter le code du slot dans *fenetre.cpp* :

Aide : bouton droit sur la déclaration du slot, choisir « Refactor » et « Ajouter la définition dans fentre.cpp »

Vous obtenez :

```
void Fenetre::onCliqueBouton()
{
    |
}
```

Tout ceci n'est pas suffisant pour que la fenêtre réagisse au clic du bouton OK. Il faut établir un lien entre le clic du bouton et l'exécution du slot : ceci est obtenu en exécutant la fonction *connect()* de la classe *QObject :*

```
QObject::connect(ui->BoutonOK, &QPushButton::clicked, this, &Fenetre::onCliqueBouton);
```

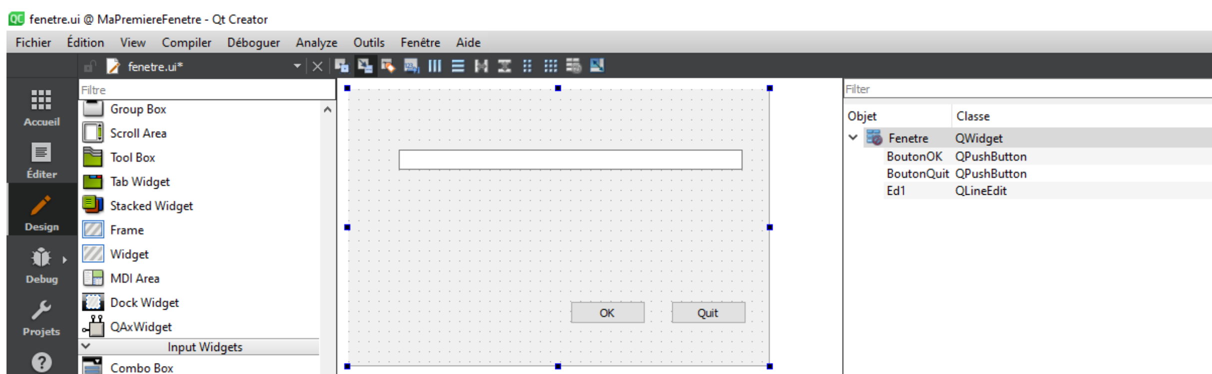
Le plus simple, c'est de l'insérer dans la méthode constructeur de la classe *Fenetre* :

```
Fenetre::Fenetre(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::Fenetre)
{
    ui->setupUi(this);

    QObject::connect(ui->BoutonOK, &QPushButton::clicked, this, &Fenetre::onCliqueBouton);

}
```

Evidemment, en cliquant sur le bouton « OK » il ne se passe rien puisqu'on n'a rien mis dans le slot !

Pour tester et voir si le bouton réagit, nous allons ajouter un nouvel objet à notre fenêtre :

Dans Qt Designer (ouvrir *fenetre.ui*), ajouter un *Line Edit* :



Appelez le *Ed1*

On peut désormais afficher une chaîne de caractères lorsqu'on clique sur le bouton OK. Ajouter le codes uivant dans le slot présent dans *fenetre.cpp* :

```
void Fenetre::onCliqueBouton()
{
    ui->Ed1->setText("Le bouton OK a été cliqué");
}
```

Compilez et vérifiez que le bouton « OK » fonctionne. Le texte doit normalement s'afficher dans le *Line Edit*

Explications sur le connect

```
QObject::connect(ui->BoutonOK, &QPushButton::clicked, this, &Fenetre::onCliqueBouton);
```

connect est une méthode de la classe OObject. C'est une méthode statique. Cela signifie qu'il n'est pas nécessaire de créer un objet de la classe QObject pour utiliser la méthode.

Ça se lit : Lorsque l'objet *BoutonOk* a été cliqué, il envoie le signal *clicked()* à l'objet *this* qui exécute alors *onCliqueBouton()*

Remarques :

Le pointeur *this* est le pointeur sur l'objet courant (celui qui exécute le code dans lequel se trouve *this*)

La classe QPushButton (dérivée de QAbstractButton) a été créée avec la capacité d'envoyer des signaux dont le signal de clicked()

La classe QWidget a été créée avec la capacité à recevoir des signaux.

Beaucoup de classes Qt ont, d'origine, des signaux et des slots déjà implémentés. (voir la doc de Qt)

Qu'est ce que les ui ?

ui pour « user interface ». Votre classe *Fenetre* est la structure de données logicielle correspondant à la fenêtre graphique affichée à l'écran. Elle contient un certain nombre d'objets graphiques (widget). Ces widgets sont regroupés dans un objet « ui ». En fait, la fenêtre elle-même est un widget. TOUT EST WIDGET !!

Les ui sont donc des widgets avec des fonctionnalités déjà implémentées. Par exemple, le bouton est capable d'origine lorsqu'on le clique de changer légèrement d'aspect pour donner l'illusion d'appuyer sur un vrai bouton. Votre fenêtre puisque c'est un widget a également tout un tas de comportement par défaut : pouvoir l'agrandir, la réduire, l'icônifier etc... Finalement, en ajoutant le slot *onCliqueBouton()* on a enrichi la fenêtre avec une nouvelle fonctionnalité. Bref, votre fenêtre c'est maintenant un peu plus qu'un simple widget.

Puisque votre fenêtre possède les widget BoutonOK, BoutonQuit et Ed1 regroupés au sein d'un objet « ui », on y accède si on est dans l'objet *Fenetre* par :

ui->BoutonOK   ou ui->BoutonQuit

Ajout d'un slot suite à un changement du QLineEdit

On désire maintenant réagir à la modification du contenu de *Ed1*. Ajouter selon le principe précédent, un slot *onChangeEd1()* qui sera exécuté à chaque modification du contenu de *Ed1*.

Indication : le signal sera *textChanged()*

<u>Contenu du slot :</u>

On voudrait saisir du texte en jaune sur fond vert tant que le nombre de caractères saisi est inférieur à 20 et en jaune sur fond rouge si on tape 20 caractères ou plus. On pourra utiliser une feuille de style CSS.

A chaque modification de *Ed1*, le slot est exécuté. On « mesure » alors la taille de la chaîne saisie. Si celle-ci est inférieure à 20 on modifie *Ed1* avec la feuille de style fond vert et écriture en jaune sinon en fond rouge. Par exemple, cet appel modifie l'aspect de *Ed1* en texte jaune sur fond vert :

```
ui->Ed1->setStyleSheet("color: yellow; background-color: green");
```

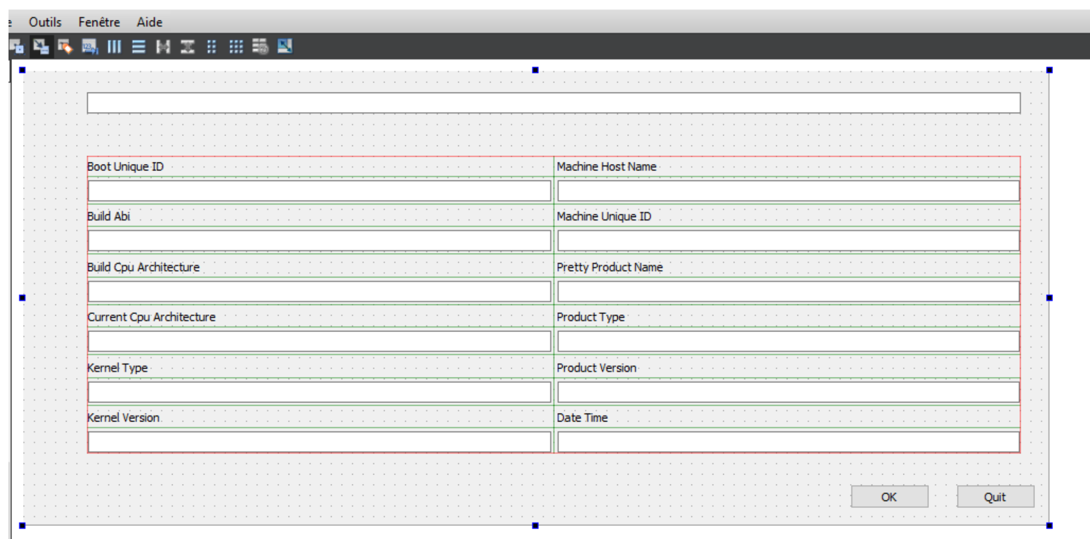
Remarquez que, si on a dépassé 20 caractères (texte en jaune sur fond rouge) et qu'on enlève des caractères pour revenir à moins de 20, le texte repasse en jaune sur fond vert.

Remarquez aussi que si on clique sur le bouton OK, le slot *onChangeEd1()* est exécuté comme si on avait tapé. Par conséquent, le signal *textChanged()* est émis si on modifie le contenu d'un QLineEdit au clavier mais aussi par programme.

Application HostZ

A l'instar des logiciels CPUZ, GPUZ etc.. donnant des informations sur les matériels et leur performance, on décide, très modestement, de réaliser une appli affichant les cractéristiques du système. Pour cela , on a utiliser la classe QSysInfo de Qt.

Celle-ci nous permet d'obtenir de base 11 informations système. Ajouter 11 QLineEdit avec 11 QLabel (texte non éditable). Vous pouvez utiliser un QLayout (dans Qt Designer) afin de placer correctement vos QLineEdit :



Conseils : aller sur le site de Qt, plus exactement sur la doc de la classe QSysInfo (dans google taper Qt QSysInfo)

https://doc.qt.io/qt-5/qsysinfo.html#machineUniqueId

Google Yahoo

Qt

Wiki Documentation Forum Bug Reports Code Review Resource Center Qt Extensions

Qt Documentation

Qt 5.15 › Qt Core › C++ Classes › QSysInfo

Contents

Public Types

Static Public Members

Detailed Description

Previous

C++ Classes

Reference

All Qt C++ Classes

All QML Types

All Qt Modules

Qt Creator Manual

All Qt Reference Documentation

Getting Started

QSysInfo Class

The QSysInfo class provides information about the system. More...

Header:	#include <QSysInfo>
qmake:	QT += core

› List of all members, including inherited members

› Obsolete members

Public Types

enum	Endian { BigEndian, LittleEndian, ByteOrder }
enum	Sizes { WordSize }

Static Public Members

QByteArray	bootUniqueId()
QString	buildAbi()
QString	buildCpuArchitecture()

Les items nous intéressant sont obtenus par l'appel aux 11 méthodes statiques (Static Public Members)

Quasiment chaque item est un QString : chaine de caractères à la mode Qt.
Les QByteArray sont des tableaux de caractères (très proches de QString)
Le douzième QLineEdit servira à afficher la date et l'heure courante : utiliser la classe QDateTime (un peu plus compliqué à utiliser que QSysInfo)
Résultat attendu :

Fenetre

HOST Z

Boot Unique ID

Machine Host Name
DESKTOP-9RQ761V

Build Abi
x86_64-little_endian-lp64

Machine Unique ID
d00a8

Build Cpu Architecture
x86_64

Pretty Product Name
Windows 10 Version 1909

Current Cpu Architecture
x86_64

Product Type
windows

Kernel Type
winnt

Product Version
10

Kernel Version
10.0.18363

Date Time
lun. déc. 7 23:56:31 2020

OK Quit