



Qt : Construire des GUI en C++

Desgenetez Charles / Reiter Maxime / Canta Thomas

07 Février 2019





- 1 Petit rappel
- 2 Parlons de Qt
- 3 Notre première interface
- 4 Introduction aux Widgets
- 5 Utilisation des signaux et slots
- 6 Un petit programme utile
- 7 Documentation Qt
- 8 Conclusion



- 1 Petit rappel
- 2 Parlons de Qt
- 3 Notre première interface
- 4 Introduction aux Widgets
- 5 Utilisation des signaux et slots
- 6 Un petit programme utile
- 7 Documentation Qt
- 8 Conclusion



- ▷ GUI est un **acronyme** anglais signifiant : Graphical User Interface ou même Interface Graphique en français.
- ▷ C'est est un dispositif de **dialogue homme-machine** permettant l'interaction entre l'humain et la machine via des boutons par exemple.

Quel intérêt ?

Il permet de **remplacer** l'interface en ligne de commande qui était moins accessible et moins intuitive dû au grand nombre de lignes de commandes complexes à apprendre.



- 1 Petit rappel
- 2 Parlons de Qt
 - Introduction
 - Installation
- 3 Notre première interface
- 4 Introduction aux Widgets
- 5 Utilisation des signaux et slots
- 6 Un petit programme utile
- 7 Documentation Qt
- 8 Conclusion

Introduction

Qu'est-ce que Qt ?



- *Qt* est une **bibliothèque** très complète qui permet notamment la création d'interface. Sa grande **diversité** et sa **portabilité** la place comme une des interfaces graphiques la plus utilisée dans le monde.

Introduction

Qu'est-ce que Qt ?



- Qt est une **bibliothèque** très complète qui permet notamment la création d'interface. Sa grande **diversité** et sa **portabilité** la place comme une des interfaces graphiques la plus utilisée dans le monde.
- Qt permet le développement de logiciel sur **Windows**, **Linux**, **Mac** ou même **Android**. Il est possible d'utiliser des options et packages pour que le logiciel soit porté sur toutes les plate-formes à la fois (utilisé notamment par *Teamspeak*, *Google*, *Skype*)

Introduction

Qu'est-ce que Qt ?



- Qt est une **bibliothèque** très complète qui permet notamment la création d'interface. Sa grande **diversité** et sa **portabilité** la place comme une des interfaces graphiques la plus utilisée dans le monde.
- Qt permet le développement de logiciel sur **Windows**, **Linux**, **Mac** ou même **Android**. Il est possible d'utiliser des options et packages pour que le logiciel soit porté sur toutes les plate-formes à la fois (utilisé notamment par *Teamspeak*, *Google*, *Skype*)
- En réalité, Qt est tellement complet en termes de bibliothèques et d'outils pour l'aide au développement que l'on appelle ça un **framework**.



- On peut coder avec n'importe lequel des IDE (Sublime Text, Emacs, Code : :Blocks...) mais on peut utiliser **Qt Creator** qui est un IDE fait spécialement pour Qt qui est optimisé et dispose d'une documentation exhaustive sur Qt.



- On peut coder avec n'importe lequel des IDE (Sublime Text, Emacs, Code : :Blocks...) mais on peut utiliser **Qt Creator** qui est un IDE fait spécialement pour Qt qui est optimisé et dispose d'une documentation exhaustive sur Qt.
- Dans ce cours, nous allons voir comment utiliser Qt avec le langage C++ mais sachez qu'il est possible de programmer en Java, Python et bien plus encore.

Introduction

Qu'est-ce que Qt ? (suite)



- On peut coder avec n'importe lequel des IDE (Sublime Text, Emacs, Code : :Blocks...) mais on peut utiliser **Qt Creator** qui est un IDE fait spécialement pour Qt qui est optimisé et dispose d'une documentation exhaustive sur Qt.
- Dans ce cours, nous allons voir comment utiliser Qt avec le langage *C++* mais sachez qu'il est possible de programmer en Java, Python et bien plus encore.

Ce qu'il faut retenir

Le framework Qt est :

- Complète
- Portable
- Facile d'accès



1988 - Haavard Nord (DG TrollTech) est chargé de créer une bibliothèque logicielle C++ pour gérer une interface graphique.

Il rencontre Eirik Chambe-Eng (Président de TrollTech) au *Norwegian Institute of Technology*.

1991 - Qt a commencé à être développé par la société dans laquelle travaille nos deux protagonistes, *Trolltech* (plus tard racheté par *Nokia*).

1995 - Le framework *peine* à démarrer son activité. La première licence est achetée et ce sera la seule pendant un an.

Mars 1996 - L'**ESA** (**A**gence **S**patiale **E**uropéenne) achète *dix* licences et déclenche l'économie de Qt, fragile jusqu'alors.



Les créateurs du framework trouvaient la lettre *Q* jolie dans la police d'Emacs (un IDE).

Ils ont ensuite rajouté la lettre *t* qui correspond en fait à *toolkit* (boîte à outils). La prononciation est très souvent souillée (*triste*).

On prononce *Qt* « **Qui oute** » ("*Cute*" en anglais). En référence à la lettre *Q* qui est mignonne. (Mouais, pourquoi pas.)

Installation

Comment installer Qt Creator ?



Accéder au lien suivant : www.qt.io/download

Installation

Comment installer Qt Creator ?



The screenshot shows the Qt website's 'Download, Try, Buy' section. The page is divided into two columns. The left column is for 'Qt' and the right column is for 'Qt Open Source'. Both columns list features under four categories: Tools, Embedded tooling & solutions, Official Qt Support Helpdesk, and Close strategic relationship with The Qt Company. The 'Go open source' button in the right column is circled in red. A chat bubble on the right says: 'Hi! I can help you get the right Qt download for your development project and, if you are ready, help you with your purchase. Please give me a click to start now.'

Qt

Products Professional Services Resources Customers Company

Download, Try, Buy.

Qt is an unbelievably comprehensive framework full of features beyond the essentials.

Tools

Qt has it's own cross-platform IDE and is chock-full of tools designed for developing applications and UIs once and deploying them across multiple operating systems.

Embedded tooling & solutions

Qt has ready-made solutions that speed up your device creation with enterprise-quality features for a truly professional development experience.

Official Qt Support Helpdesk

The Qt Company's expert support team helps you solve your development challenges.

Close strategic relationship with The Qt Company

Free trial with commercial licensing.
No credit card required.

Try free

Qt is an unbelievably comprehensive framework full of features beyond the essentials. Some features are limited to GPL.

Tools

Qt has it's own cross-platform IDE and is chock-full of tools designed for developing applications and UIs once and deploying them across multiple operating systems. Some features are limited to GPL.

Embedded tooling & solutions

Qt has ready-made solutions that speed up your device creation with enterprise-quality features for a truly professional development experience.

Official Qt Support Helpdesk

The Qt Company's expert support team helps you solve your development challenges.

Close strategic relationship with The Qt Company

When developing with Qt Open Source, we encourage that you should be aware of the terms of the LGPL, and your legal obligations.

Go open source

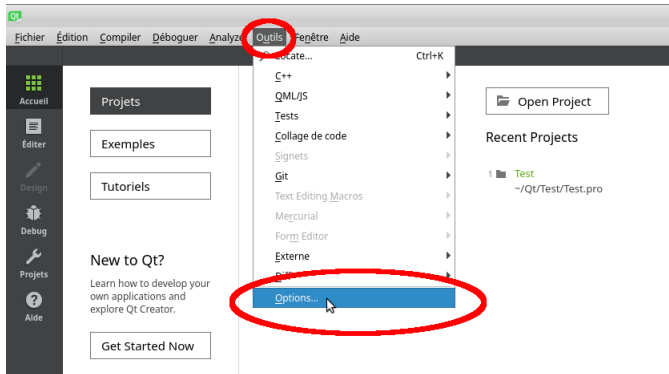
Hi! I can help you get the right Qt download for your development project and, if you are ready, help you with your purchase. Please give me a click to start now.

Installation

Setup du kit

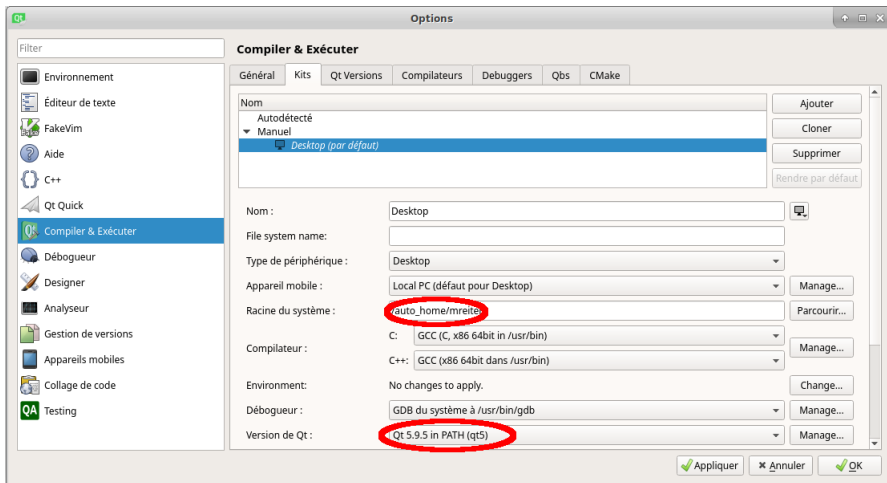


Parfois (notamment a la Fac) il faut configurer légèrement un kit de compilation.



Installation

Setup du kit

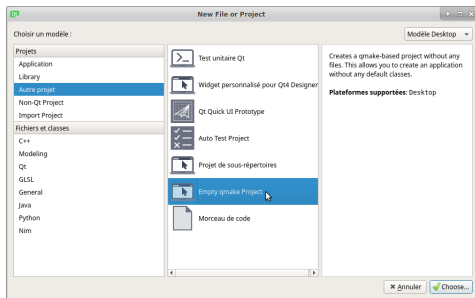




- 1 Petit rappel
- 2 Parlons de Qt
- 3 Notre première interface**
- 4 Introduction aux Widgets
- 5 Utilisation des signaux et slots
- 6 Un petit programme utile
- 7 Documentation Qt
- 8 Conclusion

- 1 Pour créer notre première fenêtre, nous allons créer un projet vide, sans fenêtre pré-crée. Pour ce faire nous allons faire :

"Fichier" → "Nouveau fichier ou projet..." → "Autre projet"
→ **"Empty qmake Project"**.



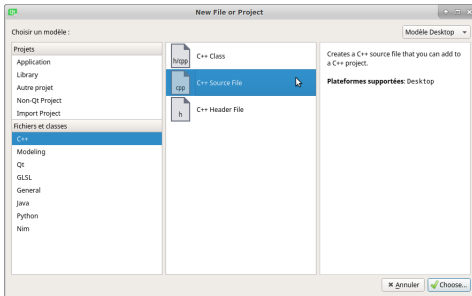


- ② On peut maintenant choisir le **nom** du projet et son **emplacement** sur l'ordinateur. Cliquez sur *suivant*, choisissez le kit que vous avez configuré au préalable et terminez la création.
- ③ Vous avez maintenant un projet vide qui contient seulement un fichier **".pro"**, il contient une configuration propre à Qt pour aider à la compilation. (On peut rajouter des widgets, des options de compilation, une icône de programme, ...)



- 4 Nous allons rajouter l'option **QT += widgets** qui va nous permettre de faire des fenêtres simples. Nous allons désormais faire "*Fichier* → "*Nouveau fichier ou projet...*" → "*C++*" → "**Source File**".

Nous allons lui donner le nom de "**main.cpp**" et terminer.



Premier fichier source

main.cpp



Remarque

Comme pour tout programme en C++, un fichier source contenant la fonction *main* est obligatoire pour pouvoir l'exécuter.

Fichier : main.cpp

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

Permet d'inclure le principe d'application qui sera reconnue par le système.



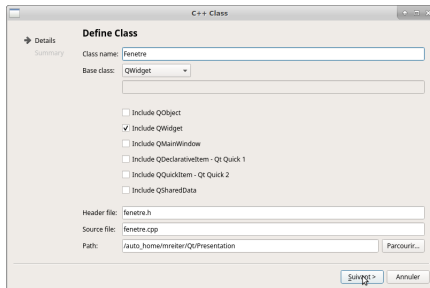
Fichier : main.cpp

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

- On initialise notre `QApplication`, à ce moment notre programme devient une application.
- **`app.exec()`** permet de dire au programme de se fermer que lorsque l'application est fermée. (souvent quand la dernière fenêtre activée et éteinte)

Nous allons créer notre propre classe *Fenêtre* afin de l'afficher.

- 1 Faire "*Fichier*" → "*Nouveau fichier...*" puis choisir "**C++ class**".



- 2 Nous allons appeler notre classe "*Fenetre*". Nous lui attribuons la *base class* "**QWidget**". Notre fenêtre va donc hériter de la classe QWidget de Qt. Puis *suivant*, puis *terminer*.



QWidget est la classe de base de tous les gadgets de Qt.
(les boutons, les barres, les textfields, ... mais aussi des fenêtres !)

- 3 Notre fenêtre est créée. Elle est vide et prend des dimensions par défaut.

Remarque

Qt Creator sépare tout seul les Headers des fichiers Sources.
Ils contiennent certaines lignes que nous détaillerons plus tard.



Nous pouvons désormais inclure la fenêtre dans notre main afin de l'afficher.

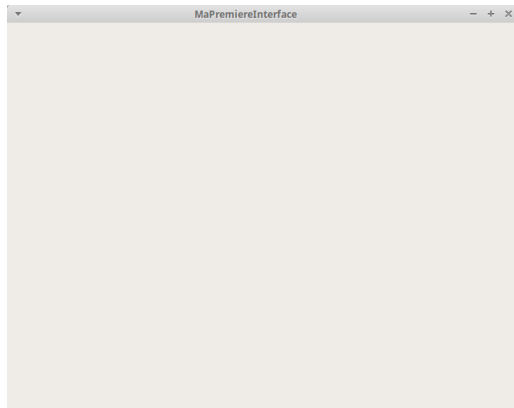
Fichier : main.cpp

```
#include <QApplication>
#include "fenetre.h" //Ne pas oublier

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    Fenetre fen; //Une instance de Fenetre
    fen.show(); //On l'affiche
    return app.exec();
}
```



Et voici le résultat :





- 1 Petit rappel
- 2 Parlons de Qt
- 3 Notre première interface
- 4 Introduction aux Widgets**
 - Découverte des widgets
 - Placer des widgets
- 5 Utilisation des signaux et slots
- 6 Un petit programme utile
- 7 Documentation Qt
- 8 Conclusion

Introduction aux différents Widgets

C'est quoi un widget ?



Definition

Un widget c'est simplement un *gadget*/un élément graphique tel qu'un bouton, une liste déroulante, etc... ou même une fenêtre !

Tous les gadgets de Qt héritent de la classe *QWidget*.

Nous, nous avons fait une fenêtre en créant un widget vide.

Tous ces objets, nous pouvons les superposer les uns sur les autres.

Nous pouvons donc placer un gadget *bouton* dans notre fenêtre vide par exemple.



Création d'un bouton

```
#include <QPushButton>
#include "fenetre.h"

Fenetre::Fenetre(QWidget *parent) : QWidget(parent)
{
    QPushButton *b;
    b = new QPushButton("Salut les enfants", this);
}
```



Création d'un bouton

```
#include <QPushButton>
#include "fenetre.h"

Fenetre::Fenetre(QWidget *parent) : QWidget(parent)
{
    QPushButton *b;
    b = new QPushButton("Salut les enfants", this);
}
```

Remarque

A chaque widget créé, le dernier paramètre est "this". Il indique que le widget appartient au widget parent. Il sera détruit en même temps que son parent. Cela évite de delete le pointeur du bouton.



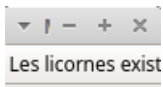
Création d'un bouton

```
#include <QPushButton>
#include "fenetre.h"

Fenetre::Fenetre(QWidget *parent) : QWidget(parent)
{
    QPushButton *b;
    b = new QPushButton("Salut les enfants", this);
}
```

Remarque

Pour modifier le texte de notre bouton on pourra utiliser :
`bouton.setText("Les licornes existent");`

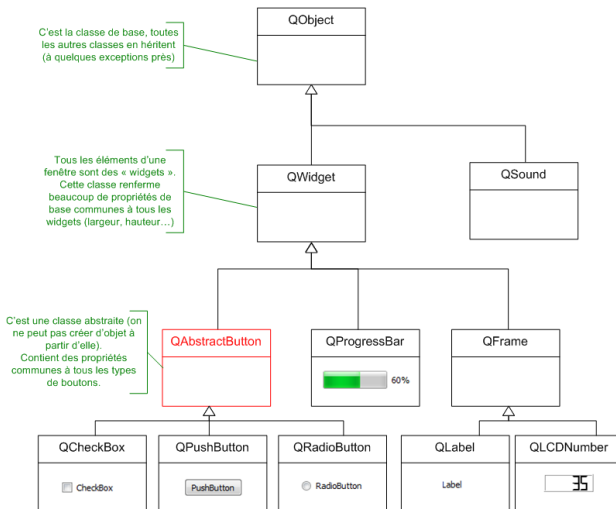


Remarque

Comme vous pouvez le voir, le bouton est coupé.
Comme nous n'avons pas défini la taille de la fenêtre alors que nous y avons placé un widget, cette dernière se rétrécit à sa taille minimum (il ne reste que les boutons pour quitter et rétrécir la fenêtre).



- Chaque widget peut être modifié avec des méthodes qui lui sont propres.
- Mais il hérite aussi des méthodes des widgets dont il dérive.
- Ainsi, pour notre bouton, on peut par exemple indiquer s'il est plat ou non (sans bordures). *setFlat(bool)* est une méthode propre à *QPushButton*.
- Mais *QPushButton* hérite de *QAbstractButton*. Ce dernier nous permet de modifier le texte, rajouter une icône, faire qu'il reste enfoncé, ...
- Enfin, *QPushButton* hérite de *QWidget*. On peut donc utiliser toutes les méthodes de *QWidget* comme le changement de taille, de police, de style, etc...



Source : <https://openclassrooms.com>

Les widgets principaux

Une liste non exhaustive



QPushButton

permet de faire des boutons. :)

QRadioButton

permet de faire des items à cocher (unique réponse).

QCheckBox

permet de faire des items à cocher (plusieurs réponses).


QLineEdit

permet de faire une ligne de texte à remplir.

QTextEdit

permet de faire plusieurs lignes de texte à remplir.

QSpinBox

permet de faire une boîte à numéro. 

QLabel

permet d'écrire simplement du texte.

QProgressBar

permet de faire une barre de chargement.

Je suis une transition

Aimez moi



Comment placer des widgets ?



Un problème

Actuellement, instancier des widgets ne fait que les empiler sur notre fenêtre. Il faut donc trouver un moyen de les placer sans avoir besoin de les déplacer pixel par pixel.



Un problème

Actuellement, instancier des widgets ne fait que les empiler sur notre fenêtre. Il faut donc trouver un moyen de les placer sans avoir besoin de les déplacer pixel par pixel.

Une solution

Pour résoudre ce problème, *Qt* nous offre les **Layouts**. Ce sont des objets qui nous permettent d'agencer les widgets comme bon nous semble.



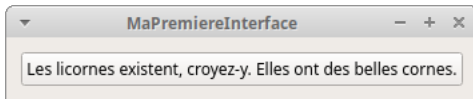
- | | |
|--------------------|--|
| QHBoxLayout | est un layout horizontal, tous les widgets sont placés automatiquement à l'horizontal à l'intérieur. (De gauche à droite) |
| QVBoxLayout | est un layout vertical, tous les widgets sont placés automatiquement à la verticale. (De bas en haut) |
| QGridLayout | est un layout en grille. Nous décidons du nombre de colonnes et de lignes et nous plaçons ensuite les widgets à l'intérieur. (La grille est invisible) |
| QFormLayout | permet de faire un agencement tel un formulaire. |



- Les layouts se superposent. On peut donc placer plusieurs layouts horizontaux dans un layout vertical par exemple.
- Le layout par défaut d'un QWidget superpose tous les widgets. Il faut donc souvent le remplacer.
- On remplace le layout d'un widget avec la méthode *setLayout(QLayout*)*.
- Pour placer des widgets dans un layout, nous utilisons la méthode *addWidget(QWidget*)*.

Exemple

```
QHBoxLayout* l = new QHBoxLayout( this );  
QPushButton* b = new QPushButton( "Envoyer", this );  
l->addWidget( b );  
setLayout( l );
```



Résultat

Le layout met en forme le bouton.

Par défaut dans les layouts, les widgets prennent toute la place qu'il peuvent.

On peut éviter ça en rajoutant des objets dans les layouts (l->addStretch()) par exemple.



- 1 Petit rappel
- 2 Parlons de Qt
- 3 Notre première interface
- 4 Introduction aux Widgets
- 5 Utilisation des signaux et slots**
- 6 Un petit programme utile
- 7 Documentation Qt
- 8 Conclusion



Nous venons de voir plusieurs widgets, comment les modifier et les placer. Mais créer une fenêtre sans interaction avec l'utilisateur n'est pas un **GUI**. Il faut donc désormais trouver le moyen d'interagir avec tous nos widgets.



Nous venons de voir plusieurs widgets, comment les modifier et les placer. Mais créer une fenêtre sans interaction avec l'utilisateur n'est pas un **GUI**. Il faut donc désormais trouver le moyen d'interagir avec tous nos widgets.

Alors, comment on fait hein ?

C'est très simple, *Qt* nous offre (Il est quand même généreux) la possibilité d'utiliser un système nommé *SIGNALS and SLOTS*. Donc... des signaux émis et reçus par des slots.

Ce type de système n'est pas nouveau, on appelle ce genre d'outils des **Évènements**.



SIGNAL

Un signal est un « *message* » envoyé par le un widget lorsqu'il vient de faire une action précise.

Exemples

- 1 Un utilisateur appuie sur un bouton.
- 2 Une barre de chargement se termine.

SLOT

C'est la fonction qui est appelée lorsqu'un évènement s'est produit. On dit que le signal appelle le slot. Il peut aussi être appelée sans signal, c'est une méthode classique.



Pour connecter un objet à un slot, on utilise la fonction `Qt connect()` :

```
connect(b, SIGNAL(pressed()), this, SLOT(close()));
```

Décortilage

- **b** est le pointeur d'un *QPushButton*.
- **SIGNAL(pressed())** est un signal de *QPushButton*
- **this** est l'objet que l'on connecte au signal (ici, notre fenêtre).
- **SLOT(close())** est le slot appelé par le signal (ici, on ferme notre fenêtre).

C'est aussi simple que cela. Désormais, notre fenêtre se fermera dès lors que nous appuyons sur le bouton.



- Ce sont des pointeurs qui sont passés dans la fonction *connect()*.
- Une connexion reste toujours active, même après une première utilisation.
- Il faut donc faire attention lors de la destruction d'objet connecté.
- Il existe déjà beaucoup de signaux/slots propres à chaque objet.
- Il est possible de recevoir des paramètres des signaux (comme le texte changé d'un *QLineEdit*) et de le passer dans un SLOT.



Information

Il est possible de créer ses propres signaux et ses propres slots.
Ici, nous allons créer seulement des SLOTS étant le plus important.

Fichier : fenetre.h

```
public slots:  
    void monSlot ();
```



Fichier : fenetre.cpp

```
Fenetre::Fenetre(QWidget *parent) : QWidget(parent)
{
    ....
    connect(b, SIGNAL(pressed()), this, SLOT(monSlot()));
}

void Fenetre::monSlot() {
    setWindowTitle("Bonsoir Pariiiiiiiis");
}
```



- 1 Petit rappel
- 2 Parlons de Qt
- 3 Notre première interface
- 4 Introduction aux Widgets
- 5 Utilisation des signaux et slots
- 6 Un petit programme utile**
- 7 Documentation Qt
- 8 Conclusion



Qu'allons-nous faire ?

Pour mettre en oeuvre toutes les notions que nous avons vues, nous allons créer un petit jeu très simple en quelques lignes de code.
(Déjà commencé)

Information

Nous allons vous montrer le code directement depuis Qt Creator.
Néanmoins, le code est disponible sur le git du cours.



- 1 Petit rappel
- 2 Parlons de Qt
- 3 Notre première interface
- 4 Introduction aux Widgets
- 5 Utilisation des signaux et slots
- 6 Un petit programme utile
- 7 Documentation Qt**
- 8 Conclusion



La documentation Qt est très exhaustive, elle est disponible de plusieurs moyens :

- <http://doc.qt.io/qtcreator/index.html>,
- Dans le logiciel lui-même.

Les deux façons ramènent sur la même documentation.



- 1 Petit rappel
- 2 Parlons de Qt
- 3 Notre première interface
- 4 Introduction aux Widgets
- 5 Utilisation des signaux et slots
- 6 Un petit programme utile
- 7 Documentation Qt
- 8 Conclusion**

Conclusion

Ce qu'il faut en retenir



Qt est un framework **sophistiqué** et **efficace** allié à une documentation **organisée** et **complète**.

Il est très **accessible** même dès la première utilisation notamment dû à **l'explicitation** du nom des fonctions et de la doc intégrée.

De plus un programmeur lambda ne peut être restreint de son utilisation grâce à sa **portabilité sur toutes les plateformes**

Ce cours ne peut traiter le framework dans sa globalité. Nous vous invitons donc à essayer et de vous rendre compte des possibilités infinies qu'il propose.

Ça y est c'est la fin
mais pas celle des haricots



Merci à tous pour votre écoute !
(vous êtes vachement sympa d'avoir tout écouté quand même)

On espère que le sujet vous aura plus !
(like, commente et partage)

N'hésitez pas à poser des questions si vous n'avez pas
compris quelque chose !
(mais ne vous y sentez pas obligé)