QT: TP 03 Edition de ligne

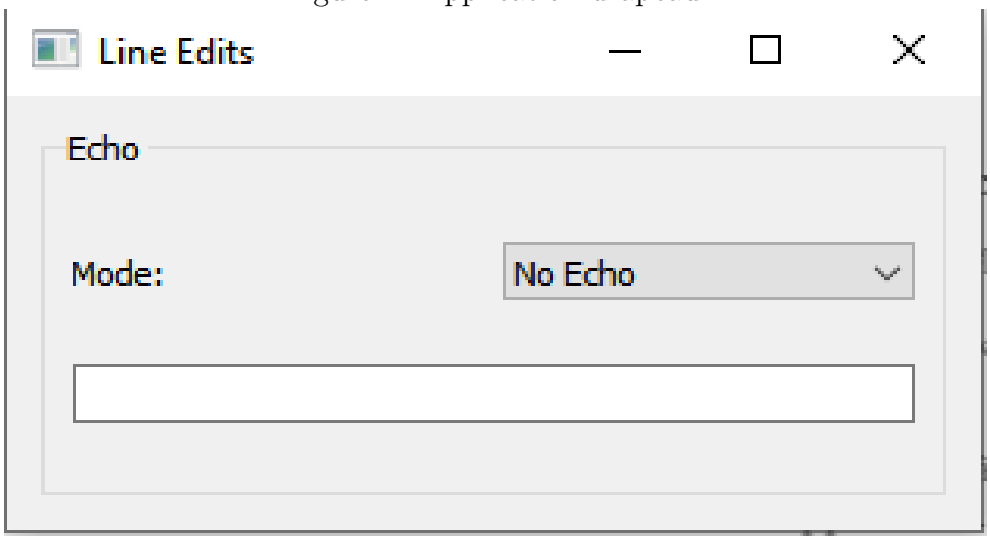
DAKKAR Borhen-eddine
Lycée le Corbusier
BTS SN

November 18, 2021

1 Objectif du TP

L'exemple consiste en une seule fenêtre utilisant un **Line Edits**, contenant une sélection de modifications de ligne avec différentes contraintes d'entrée et propriétés d'affichage qui peuvent être modifiées en sélectionnant des éléments dans les listes déroulantes.
La Figure 1 représente le résultat attendu de cette application.

Figure 1: Application drapeaux



2 Application 1

Nous allons détailler les différentes étapes pour réaliser cette application.

2.1 Etape 1: création du projet

1. Lancez Qt Creator.
2. Sur la page "Accueil" faites "New".
3. Ensuite "Application–>Qt Widgets Application–>Choose...".
4. Renommer votre projet et choisissez son emplacement (Chaque élève doit créer son pro propre répertoire dans lequel il va enregistrer ses projets).
5. Cliquez sur "suivant".
6. Sur "Define Build Syetem" choisir "Qmake" (c'est le systeme de build par défaut'.
7. Sur "Class Information" renseignez les informations des classes. Pour le moment nous allons laisser par défaut.
8. Faites "suivant" puis "suivant".
9. Sur la page "Kit Selection" vous allez choisir le compilateur. Si vous avez bien effectuer votre installation, il va vous afficher tous les compilateurs installés. Choisissez un et faites "suivant".
10. Sur "Projet Management" un récapitulatif des fichiers de projet est donné. Appuyez sur "Terminer" pour finaliser la création de votre projet.

2.2 Etape 2: Fichiers du projet

Comme vous avez pu le constater sur la page "Projet Management", un projet Qt est compsé des fichiers suivat:

- main.cpp: Un fichier source pour lancer l'application.

- window.cpp: Fichier source qui contient la définition de la classe "Window".
- window.h: Fichier entête (Header) qui contient la déclaration de de la classe "Window".
- window.ui: Ce fichier utilise un format XML pour représenter les formes et leurs caractéristiques.
- Mon_premier_projet.pro: Les fichiers ".pro" contiennent toutes les informations requises par qmake pour créer votre application, bibliothèque, ...[1].

Une fois le projet Qt est créé, le fichier main.cpp contient le code suivant:

```
1  #include "mainwindow.h"
2
3  #include <QApplication>
4  int main(int argc, char *argv[])
5      {
6          QApplication a(argc, argv);
7          MainWindow w;
8          w.show();
9          return a.exec();
10     }
```

- La classe QApplication est necéssaire pour le fonctiennement de toute application.
- La fonction show() est necessaire pour afficher l'application.
- La fonction exec() est appelée boucle d'évènements, elle démarre l'application.

Dès maintenat vous pouvez tester votre promière application:

- En bas à guache cliquez sur l'icône "marteau" (Ctrl+B) pour compiler le projet.
- S'il n'y a pas d'erreur, appyuez maintenant sur la flèche verte (Ctrl+R) pour éxecuter le projet.

2.3 Etape 3: Création de la classe Window

La classe Window hérite de la classe QWidget. Elle est utilisée pour la création des différents éléments constituants l'application. Remplacer le code généré de cette classe par le code suivant :

```
#ifndef WINDOW_H
#define WINDOW_H

#include <QWidget>

QT_BEGIN_NAMESPACE
class QLineEdit;
QT_END_NAMESPACE

//! [0]
class Window : public QWidget
{
    Q_OBJECT

public:
    Window(QWidget *parent = nullptr);

public slots:

private:
    QLineEdit *echoLineEdit;
};
//! [0]

#endif
```

La classe Window contient un contructeur et un objet de QlineEdit appellé **echoLineEdit**.

2.4 Etape 4: Définition du constructeur de la calsse window

Allez dans le fichier **Window.cpp** et ecriver le code suivant :

```
#include "window.h"
#include "ui_window.h"

#include <QComboBox>
#include <QGridLayout>
#include <QGroupBox>
#include <QLabel>
#include <QLineEdit>


Window::Window(QWidget *parent)
    : QWidget(parent)
{
//! [1]
    QGroupBox *echoGroup = new QGroupBox(tr("Echo"));

    QLabel *echoLabel = new QLabel(tr("Mode:"));
    QComboBox *echoComboBox = new QComboBox;
    echoComboBox->addItem(tr("Normal"));
    echoComboBox->addItem(tr("Password"));
    echoComboBox->addItem(tr("PasswordEchoOnEdit"));
    echoComboBox->addItem(tr("No Echo"));

    echoLineEdit = new QLineEdit;
    echoLineEdit->setPlaceholderText("Entrer votre texte");
//! [1]
```

- echoGroup : est un objet de la classe **QGroupBox**. Il fournit un cadre avec un titre en haut pour afficher divers widgets à l'intérieur. La fonction **setLayout** permet de rajouter un layout.

```
echoGroup->setLayout(echoLayout);
```

- echoLabel : est un objet de la classe **QLabel**. Il est utilisé pour afficher un texte, ici "Mode:".
- echoComboBox : est un objet de la classe **QComboBox**. Il permet de présenter une liste d'options à l'utilisateur. Nous avons utilisé la fonction **addItem** pour ajouter les différentes options à ce combobox.
- echoLineEdit : est un objet de la classe **QLineEdit**. Il permet à l'utilisateur de saisir et de modifier une ligne de texte. La fonction **setPlaceholderText** permet d'afficher un texte dans l'arrière plan de **QLineEdit**.
-

2.5 Etape 5: Disposition des widget

Pour positionner nos widgets, nous allons créer un **QGridlayout** comme suit:

```
1 //! [2]
2     QGridLayout *echoLayout = new QGridLayout;
3     echoLayout->addWidget(echoLabel, 0, 0);
4     echoLayout->addWidget(echoComboBox, 0, 1);
5     echoLayout->addWidget(echoLineEdit, 1, 0, 1, 2);
6     echoGroup->setLayout(echoLayout);
7 //! [2]
```

Ajoutez ce code à votre programme.

- echoLayout : est un objet de la classe **QGridLayout**. Il divise un espace en lignes et colonnes, et place chaque widget dans une cellule.
- La fonction **addWidget** est utilisaée pour ajouter un widget donné à la grille de cellule à la ligne, colonne. La position en haut à gauche est (0, 0) par défaut.
- La fonction **setLayout** est une fonction de la classe **QWidget** qui permet de lancer un widget.

2.6 Etape 6: Mettre echoGroup deans un nouveau layout

Nous allons maintenant positionner notre **echoGroup** dans un nouveau layout appelé **layout** .

```
//! [3]
    QGridLayout *layout = new QGridLayout;
    layout->addWidget(echoGroup, 0, 0);// 0,0 définit la position de notre layout
    setLayout(layout);

    setWindowTitle(tr("Line Edits"));// définit le titre de l'application
}
//! [3]
```

Compilez (Ctrl+B) et exécutez (Ctrl+R) votre application. Testez votre application.

2.7 Etape 7: Définition de la fonction echoChanged

Allez dans le fichier **Window.h** et rajoutez un nouvelle fonction publique comme suit :

```
void echoChanged(int);
```

Définissons maintenant cette fonction dans **Window.cpp** :

```
//! [4]
void Window::echoChanged(int index)
{
    switch (index) {
    case 0:
        echoLineEdit->setEchoMode(QLineEdit::Normal);
        break;
    case 1:
        echoLineEdit->setEchoMode(QLineEdit::Password);
        break;
    case 2:
```

```
12          echoLineEdit->setEchoMode(QLineEdit::PasswordEchoOnEdit);
13          break;
14      case 3:
15          echoLineEdit->setEchoMode(QLineEdit::NoEcho);
16          break;
17      }
18  }
19  //! [4]
```

Elle prend en paramètre un entier **"index"** qui definit à l'aide de **switch ... case** et de **setEchoMode** le mode de **echoLineEdit**. Le tableau ci-dessous montre les différents mode de **QLineEdit** :

- **QLineEdit::Normal** valeur "0", affiche les caractères au fur et à mesure de leur saisie. C'est la valeur par défaut.
- **QLineEdit::NoEcho** valeur "1", n'affiche rien. Cela peut être approprié pour les mots de passe où même la longueur du mot de passe doit être gardée secrète.
- **QLineEdit::Password** valeur "2", masque les caractères du mot de passe au lieu des caractères réellement saisis.
- **QLineEdit::PasswordEchoOnEdit** valeur "3", Affiche les caractères tels qu'ils sont saisis lors de l'édition, sinon affiche les caractères comme avec un mot de passe.

2.8 Etape 8: Connexion du signal activated

Dans cette étape, nous allons conneté le signal **activated** de **echoComboBox** au slot **echoChanged**.

```
1  //! [5]
2  connect(echoComboBox, QOverload<int>::of(&QComboBox::activated),
3              this, &Window::echoChanged);
4  //! [5]
```

- Le signal void QComboBox::activated(int index) : ce signal est envoyé lorsque l'utilisateur choisit un élément dans la liste déroulante. L'**index** de l'élément est envoyé. Notez que ce signal est envoyé même lorsque le choix n'est pas modifié.

Compilez (Ctrl+B) et exécutez (Ctrl+R) votre application.

3 Application 2

3.1 Application athentification

Créez une nouvelle application similaire à l'application 2 et qui :

1. Affiche un pavé numérique.
2. Affiche deux champs de texte.
   - Le primier champ est appelé **Identifiant**. Il permet de saisir un numéro d'identifiant.
   - Le deuxième champ est appelé **Mot de passe**. Il permet de saisir un mot de passe.
3. Une case à cocher (QCheckBox) qui permet d'afficher ou de masquer l'Identifiant.
4. Une case à cocher (QCheckBox) qui permet de masquer ou de ne pas afficher le Mot de passe.
5. Affiche un bouton **Connexion** qui permet de se connecter après la saisie des deux champs précédents.
6. L'application vérifie la bonne écriture de l'identifiant et du mot de passe. Ensuite, elle compare les champs saisis à des variables internes enregistées précédemment (**ID** pour Identifiant et **M_p** pour Mot de passe).
7. Si la saisie correspond aux variables internes, le bouton **Connexion** sera activé.
8. L'appui bouton provoque l'affichage d'un message indiquant la réussite de l'authentification (le message s'affiche en bas de l'appication).

3.2 Spécifications de l'application

1. Le champ **Identifiant** ne doit accepter que 9 chiffres. Cette contrainte est liée directement au champ **Mot de passe**. C.à.d, si vous ecrivez un identifiant qui contient 8 chiffres par exemple, le champ mot de passe ne doit pas être activé.

2. Si l'**Identifiant** ne correspond pas à la variable **ID**, un message d'erreur sera affcihé en bas de l'application.

3. Au cas où le **Mot de passe** est incorrect, un texte indiqant le nombre de tentative restante vous sera affiché. La saisie 3 fois successives d'un mot de passe incorrect provoque la fermeture de l'application.

4. La génération des boutons numériques doit être aléatoire à chaque lancement de l'application.

5. Chaque appui bouton sur le clavier numérique provoque la coloration en **rouge** du bouton enfoncé (la coloration dure le temps d'appui).

6. Le bouton **Connexion** sera coloré en **vert** pour chaque authentification réussite et en rouge pour chaque athentification fausse.

References

[1] pro_File. https://doc.qt.io/qt-5/qmake-project-files.html. 2020.