

IHM avec Qt

Les Widgets

DAKKAR Borhen-eddine

Lycée le Corbusier

BTS SN-IR

Table des matières

- 1 Table des matières
- 2 La classe QLineEdit
- 3 La classe QTextEdit
- 4 La classe QCheckBox
- 5 La classe QSpinBox
- 6 La classe QFileDialog
- 7 La classe QFile

La classe QLineEdit

- Une **LineEdit** de ligne permet à l'utilisateur d'entrer et de modifier une seule ligne de texte avec une collection utile de fonctions d'édition.
- En changeant le **echoMode()** d'une **LineEdit**, il peut également être utilisé comme un champ "écriture seule", pour des entrées telles que des mots de passe.
- Nous pouvons modifier le texte avec **setText()** ou **insert()**.
- Le texte peut être aligné avec **setAlignment()**.
- La méthode **text()** est utilisée pour récupérer le text écrit sur la **LineEdit**.
- Lorsque le texte change, le signal **textChanged()** est émis.
- Le texte peut être aligné avec **setAlignment()**. [4]

Exemple QLineEdit

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QVBoxLayout>
#include <QLabel>
#include <QLineEdit>
QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
    QVBoxLayout * Vbox = new QVBoxLayout;
    QLineEdit *Champ_text = new QLineEdit(); // Instanciation d'un champ de text
    QLabel *Label = new QLabel;
private:
    Ui::MainWindow *ui;
private slots:
    void changement_de_text();
};
#endif // MAINWINDOW_H
```

mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    setWindowTitle("Champ de text QLineEdit");
    resize(300,100);

    Champ_text->setPlaceholderText("Entrez votre text");// Text par défaut
    Champ_text->setMaxLength(20);// Définit la taille max du text
    //--- Connexion du signal textChanged avec le slot ---//
    connect(Champ_text, SIGNAL(textChanged(QString)), this,
            SLOT(changement_de_text()));

    Label->setText("QLineEdit");
    VBox->addWidget(Champ_text);
    VBox->addWidget(Label);

    QWidget *Widget = new QWidget;
    Widget->setLayout(Vbox);
    this->setCentralWidget(Widget);
}
```

mainwindow.cpp

```
void MainWindow::changement_de_text()
{
    this->Label->setText( Champ_text->text());
}

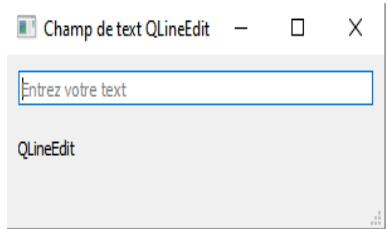
MainWindow::~MainWindow()
{
    delete ui;
}
```

main.cpp

```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```



La classe QTextEdit

- **QTextEdit** est un visualiseur/éditeur avancé de text.
- Il est optimisé pour traiter des documents volumineux et pour répondre rapidement aux entrées de l'utilisateur.
- **QTextEdit** peut afficher des images, des listes et des tableaux.
- Le signal **textChanged** est envoyé quand l'utilisateur change le texte.
- La méthode **toPlainText()** est utilisée pour récupérer le text entré par l'utilisateur. [6]

Exemple QTextEdit

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QVBoxLayout>
#include <QTextEdit>
#include <QLabel>
QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
    QVBoxLayout * Vbox = new QVBoxLayout;
    QTextEdit *txt_edit = new QTextEdit();
    QLabel *Label = new QLabel();
private:
    Ui::MainWindow *ui;

private slots:
    void changement_de_text();
};

#endif // MAINWINDOW_H
```

mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    setWindowTitle("QTextEdit");
    resize(300,200);

    txt_edit->setText("Hello, QTextEdit!");
    txt_edit->append("Ajoutez un text ici...");
    connect(txt_edit,SIGNAL(textChanged),this,SLOT());
    //--- Connexion du signal textChanged avec le slot ---//
    connect(txt_edit, SIGNAL(textChanged()), this,
            SLOT(changement_de_text()));

    VBox->addWidget(txt_edit);
    VBox->addWidget(Label);

    QWidget *Widget = new QWidget;
    Widget->setLayout(Vbox);
    this->setCentralWidget(Widget);
}
```

```
void MainWindow::changement_de_text()
{
    QString text;
    this->Label->setText( txt_edit->toPlainText());
}

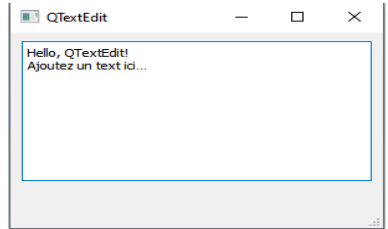
MainWindow::~MainWindow()
{
    delete ui;
}
```

main.cpp

```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```



La classe QCheckBox

- Une **QCheckBox** (case à cocher) est un bouton d'option qui peut être activé (coché) ou désactivé (non coché).
- Les cases à cocher sont généralement utilisées pour représenter les fonctionnalités d'une application qui peuvent être activées ou désactivées sans affecter les autres.
- Chaque fois qu'une case est cochée ou décochée, elle émet le signal **stateChanged()**.
- La méthode **isChecked()** est utilisée pour demander si une case à cocher est cochée ou non.
- **setCheckState** définit l'état de la case à cocher. Elle peut prendre les trois paramètres suivant:
 - **Qt::Unchecked** : 0 : L'élément n'est pas coché.
 - **Qt::PartiallyChecked** : 1 : L'élément est demi-coché.
 - **Qt::Checked** : 2 : L'élément est coché.
- Si vous n'avez pas besoin de tristate (trois états), vous pouvez également utiliser **setChecked()**, qui prend un booléen [1].

Exemple QCheckBox

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QVBoxLayout>
#include <QCheckBox>
#include <QLabel>
QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
    QVBoxLayout * Vbox = new QVBoxLayout;
    QCheckBox *case_a_cocher = new QCheckBox("Case à cocher");
    QLabel *Label = new QLabel();
private:
    Ui::MainWindow *ui;

private slots:
    void changement_d_etat();
};

#endif // MAINWINDOW_H
```

mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    setWindowTitle("QTextEdit");
    resize(300,200);
    //case_a_cocher->setChecked(false);
    //case_a_cocher->setCheckState(Qt::Unchecked); // Etat décoché
    //case_a_cocher->setCheckState(Qt::PartiallyChecked); // Etat demi-coché
    case_a_cocher->setCheckState(Qt::Checked); // Etat coché
    //--- Connexion du signal textChanged avec le slot ---//
    connect(case_a_cocher, SIGNAL(stateChanged(int)), this,
            SLOT(changement_d_etat()));

    VBox->addWidget(case_a_cocher);
    VBox->addWidget(Label);

    QWidget *Widget = new QWidget;
    Widget->setLayout(Vbox);
    this->setCentralWidget(Widget);
}
```

```
void MainWindow::changement_d_etat()
{
    QString text = QString::number(case_a_cocher->checkState()) ;
    this->Label->setText(text);
}
```

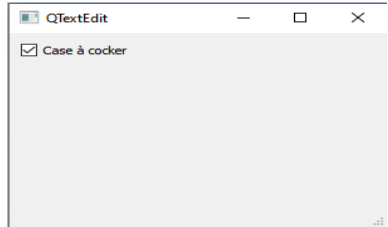
```
MainWindow::~MainWindow()
{
    delete ui;
}
```


main.cpp

```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```



La classe QSpinBox

- **QSpinBox** est conçue pour gérer des entiers et des ensembles discrets de valeurs.
- **QSpinBox** permet à l'utilisateur de choisir une valeur en cliquant sur les boutons haut/bas ou en appuyant sur haut/bas sur le clavier pour augmenter/diminuer la valeur actuellement affichée.
- Chaque fois que la valeur change, **QSpinBox** émet des signaux **valueChanged()** et **textChanged()**, le premier fournissant un entier (int) et le second une chaîne de caractère (QString).
- La valeur actuelle peut être récupérée avec **value()** et définie avec **setValue()**.
- La valeur minimale et maximale et la taille de pas peuvent être modifiées avec **setMinimum()**, **setMaximum()** et **setSingleStep()**[5].

Exemple QSpinBox

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QVBoxLayout>
#include <QSpinBox>
#include <QLabel>
QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
    QVBoxLayout * Vbox = new QVBoxLayout;
    QSpinBox *Spin_box = new QSpinBox();
    QLabel *Label = new QLabel();
private:
    Ui::MainWindow *ui;

private slots:
    void changement_de_valeur();
};

#endif // MAINWINDOW_H
```

mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    setWindowTitle("QTextEdit");
    resize(300,200);
    Spin_box->setMinimum(-15);
    Spin_box->setMaximum(15);
    Spin_box->setSingleStep(1);

    //connect(Spin_box, SIGNAL(valueChange(int)), this,
    //        SLOT(changement_de_valeur()));
    connect(Spin_box, QOverload<int>::of(&QSpinBox::valueChanged),
        [=](int i){ Label->setNum(i);/* ... */ });
    Vbox->addWidget(Spin_box);
    Vbox->addWidget(Label);

    QWidget *Widget = new QWidget;
    Widget->setLayout(Vbox);
    this->setCentralWidget(Widget);
}
```

```
void MainWindow::changement_de_valeur()
{
    QString str = QString::number(Spin_box->value());
    Label->setText(str);
}

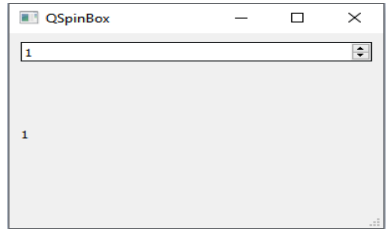
MainWindow::~MainWindow()
{
    delete ui;
}
```

main.cpp

```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```



La classe QFileDialog

- La classe **QFileDialog** permet à un utilisateur de parcourir le système de fichiers afin de sélectionner un ou plusieurs fichiers ou un répertoire [3].
- Après la création d'un **QFileDialog**, la fonction **getOpenFileName** permet d'ouvrir un fichier et récupérer son chemin.

QFileDialog

```
QString Chemin;  
Chemin = QFileDialog::getOpenFileName(this,  
    tr("QFileDialog"),  
    tr("Image Files (*.png *.jpg *.bmp)"));
```

- La boîte de dialogue affiche initialement le contenu du répertoire. Par exemple le répertoire défini par le chemin:
"C:/Users/Dell/Desktop/QFileDialog".
- Elle affiche uniquement les fichiers correspondant aux modèles donnés dans la chaîne "Fichiers d'image (* .png * .jpg * .bmp)".
- Le parent de la boîte de dialogue est défini avec **this** (c.-à-d. l'objet courant) et le titre de la fenêtre est défini par "QFileDialog".
- La variable "Chemin" stocke le chemin complet du fichier sélectionné renvoyée par **getOpenFileName**.

La classe QFile







- **QFile** est un périphérique d'E/S pour lire et écrire des textes et des fichiers binaires.
- Le nom du fichier est généralement passé dans le constructeur, mais il peut être défini à tout moment à l'aide de **setFileName()**.
- Vous pouvez vérifier l'existence d'un fichier en utilisant la méthode **exist()** et supprimer un fichier en utilisant **remove()**.
- Le fichier s'ouvre avec **open()**, se ferme avec **close()** et se vide avec **flush()**.
- Les données sont lues et écrites à l'aide de **QDataStream** ou **QTextStream**.
- Vous pouvez également appeler les fonctions héritées de **QIODevice** **read()**, **readLine()**, **readAll()** et **write()** [2].

Lecture d'un fichier text

```
QFile file("Chemin du fichier")\n\nspace{0.5cm}  
//--- Lecture d'un fichier text en utilisant Streams ---//  
if (!file.open(QIODevice::ReadOnly | QIODevice::Text))  
    return;  
  
QTextStream in(&file);  
  
while (!in.atEnd()) {  
    QString line = in.readLine();  
  
    process_line(line); // Traitement de text  
  
    Label->setText(line); // Afficher le text sur un Label  
}
```

Écriture sur un fichier text

```
QFile file("Chemin du fichier")\vspace{0.5cm}  
  
//--- Ecrire sur d'un fichier text en utilisant Streams ---//  
if (!file.open(QIODevice::WriteOnly | QIODevice::Text))  
    return;  
  
QTextStream out(&file);  
  
out << "J'apprends comment utiliser QFile \n";
```

-  *QCheckBox*. <https://doc.qt.io/qt-5/qcheckbox.html#details>. 2020.
-  *QFile*. <https://doc.qt.io/qt-5/qfile.html#details>. 2020.
-  *QFileDialog*. <https://doc.qt.io/qt-5/qfiledialog.html#details>. 2020.
-  *QlineEdit*. <http://doc.qt.io/qt-5/qlineedit.html#details>. 2020.
-  *QSpinBox*. <https://doc.qt.io/qt-5/qspinbox.html#details>. 2020.
-  *QTextEdite*. <https://doc.qt.io/qt-5/qtextedit.html#details>. 2020.