IHM avec Qt

Les Menus

DAKKAR Borhen-eddine

Lycée le Corbusier

BTS SN-IR

Table des matières

1. Table des matières
2. La classe QToolBar
3. La classe QAction
4. Les icones
5. La classe QStatusBar
6. La classe QMenu

La classe QToolBar

- La classe QToolBar fournit un panneau qui contient un ensemble de contrôles.
- Les boutons de la barre d'outils sont utilisées avec des actions. Pour les ajouter il faut utiliser addAction() ou insertAction().
- Les groupes de boutons peuvent être séparés en utilisant addSeparator() ou insertSeparator().
- Lorsqu'un bouton de la barre d'outils est enfoncé, il émet le signal actionTriggered().
- Les méthodes setMovable(), isMovable(), allowedAreas() et isAreaAllowed() sont utilisées pour fixer une barre d'outils dans une zone âriculière.Par exemple, en haut de la fenêtre, ou elle peut être déplacée entre des zones de barre d'outils.
- addToolBar permet d'insérer une berre d'outils dans la zone supérieure de la fenêtre [2].

Création d'une barre d'outils

```
QToolBar *barre_outils = new QToolBar;

addToolBar(barre_outils);
```

- La méthode addToolBar hérite de QMainWindow et elle permet d'afficher la nouvelle barre d'outils.

La classe QAction

- Dans les applications, de nombreuses commandes courantes peuvent être appelées via des menus, des boutons de barre d'outils et des raccourcis clavier. Étant donné que l'utilisateur s'attend à ce que chaque commande soit exécutée de la même manière, quelle que soit l'interface utilisateur utilisée, il est utile de représenter chaque commande comme une action.

- Nous pouvins ajouter des actions aux menus et aux barres d'outils. Par exemple, dans un traitement de texte, si l'utilisateur appuie sur un bouton de la barre d'outils Gras, l'élément de menu Gras sera automatiquement coché.

La classe QAction

- Une QAction peut contenir une icône, un texte de menu, un raccourci, un texte d'état, et une info-bulle. Ils peuvent être définis avec setIcon(), setText(), setIconText(), setShortcut(), setStatusTip(), setWhatsThis() et setToolTip().
- Les actions sont ajoutées aux widgets en utilisant QWidget :: addAction() ou QGraphicsWidget :: addAction(). Notez qu'une action doit être ajoutée à un widget avant de pouvoir être utilisée.
- Une fois qu'une QAction a été créée, elle doit être ajoutée au menu et à la barre d'outils appropriés, puis connectée à un slot qui effectuera l'action [**QAction**].

Exemple QAction

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
private slots:
    void ouvrir_fichier();

private:
    Ui::MainWindow *ui;
};
#endif // MAINWINDOW_H
```

mainwindow.cpp

```cpp
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include<QToolBar>
#include<QAction>
#include <QFileDialog>
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    //--- Création d'une barre d'outils ---//
    QToolBar *barre_outils = new QToolBar;
    addToolBar(barre_outils);
    //--- Création d'une Action ---//
    QAction *Action_ouvrir = new QAction("Ouvrir");
    Action_ouvrir->setShortcuts(QKeySequence::Open);
    connect(Action_ouvrir, &QAction::triggered, this,
            &MainWindow::ouvrir_fichier);
            barre_outils->addAction(Action_ouvrir);
}
void MainWindow :: ouvrir_fichier()
{
    //--- Le slot fait appel à QFileDialog ---//
    //--- pour ouvrir un fichier ---//
    QString fileName = QFileDialog::getOpenFileName(this);
}
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();
}
```

- L'action Action_ouvrir définie dans cette exemple fait appel à la classe QFiledialog afin d'ouvrir un répertoire.

Les icones

- Nous pouvons ajouter une icône en utilisant la fonction fromTheme.

Ajout d'une icône "fromTheme"

```
const QIcon openIcon = QIcon::fromTheme("Ouvrir document",
                                QIcon("Chemin de l'icone"));

 QAction *Action_ouvrir = new QAction(openIcon,tr("&Ouvrir"),
                                      this);
```

- Nous pouvons ajuster la taille de l'icône à l'aide de la fonction setIconSize.

Ajout d'une icône "Ajustement de la taille"

```
QToolBar *barre_outils = new QToolBar;
 addToolBar(barre_outils);
 barre_outils->setIconSize(QSize(16,16));
 QAction *Action_ouvrir = new QAction(QIcon("Chemin de l'icone"),
                                      tr("&Ouvrir..."), this);
```

La classe QStatusBar

- La barre d'état peut être une des catégories suivantes:
  1. Temporaire: occupe brièvement la majeure partie de la barre d'état. Utilisé pour expliquer les textes des info-bulles ou les entrées de menu, par exemple.
  2. Normale: occupe une partie de la barre d'état et peut être masqué par des messages temporaires. Utilisé pour afficher la page et le numéro de ligne dans un traitement de texte, par exemple.
  3. Permanente: elle n'est jamais caché. Utilisée pour des indications de mode importantes.
- La méthode setStatusTip permet d'ajouter un text qui sera affiché sur la barre d'état.

Afficher un text sur la barre d'état

```
Action_ouvrir->setStatusTip("Ouvrir un fichier");
```

La classe QMenu

- La classe QMenu fournit un widget de menu à utiliser dans les barres de menus et les menus contextuels.
- Les menus déroulants sont affichés par la barre de menus lorsque l'utilisateur clique sur l'élément correspondant ou appuie sur la touche de raccourci spécifiée.
- Utilisez QMenuBar :: addMenu() pour insérer un menu dans une barre de menus.
- Un menu se compose d'une liste d'action (QAction).
- Il existe quatre types d'éléments d'action: les séparateurs, les actions qui affichent un sous-menu, les widgets et les actions qui exécutent une action. Les séparateurs sont insérés avec addSeparator(), les sous-menus avec addMenu() et tous les autres éléments sont considérés comme des actions[1].

Création d'un menu

Menu

```
//--- Création d'une Action ---//
QAction *Action_ouvrir = new QAction("&Ouvrir");
QAction *Action_fermer = new QAction("&Fermer");

//--- Utilisation de reccourci "Ctrl+O" pour ouvrir ---//
Action_ouvrir->setShortcut(QKeySequence(tr("Ctrl+O")));
Action_fermer->setShortcut(QKeySequence(tr("Ctrl+F")));

//--- Connexion du signal-slot ---//
connect(Action_ouvrir, &QAction::triggered, this,
        &MainWindow::ouvrir_fichier);
connect(Action_fermer,&QAction::triggered,this,&QApplication::quit);

//--- Ajouter un menu ---//
QMenu *Menu_1 = new QMenu;
Menu_1 = menuBar()->addMenu("&Fichier");
Menu_1->addAction(Action_ouvrir);
Menu_1->addAction(Action_fermer);
```

Création d'un menu

- La fonction setShortcut permet de définir le raccourci clavier à l'aide de QKeySequence.
- La fonction addSeparator() crée une nouvelle action de séparation qui permet de rajouter de nouveau menu.

Afficher un text sur la barre d'état

```
//--- Ajouter une séparation ---//
Menu_1->addSeparator();
QMenu *Menu_2 = new QMenu;
Menu_2 = menuBar()->addMenu("Edition");
Menu_2->addAction(Action_ouvrir);
Menu_2->addAction(Action_fermer);
```

Les sous-menus

- Il est possible de créer des sous-menus dans les menus.
- Pour le faire, nous faisons appel à la fonction addMenu utilisée précédemment.

Ajouter un sous-menu

```
//--- Ajouter un sous-menu ---//
QMenu *Sous_menu1 = new QMenu;
Sous_menu1 = Menu_1->addMenu("sous menu 1");
Sous_menu1->addAction(Action_ouvrir);
Sous_menu1->addAction(Action_fermer);
```

Références


QMenu. https://doc.qt.io/qt-5/qmenu.htmldetails. 2020.


QToolbar. https://doc.qt.io/qt-5/qtoolbar.htmldetails. 2020.