

Ajouter une 3e fenêtre



◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ 🔍 ↺

Les signaux personnalisés

Un signal peut être connecté à un autre signal. Dans ce cas, lorsque le premier signal est émis, il entraîne l'émission du second.

Les slots personnalisés

Exemple

A set of small navigation icons typically found in Beamer presentations, including symbols for back, forward, search, and other slide controls.

Utilisation

L3 Info. CDAA

R. Raffin

Présentation
générale

Programmation
événementielle

Structure
générale de Qt
Multi-plateforme

Quelques rappels
de C++

Exemple basique de
source C++

C++ et containers

C++ et compilation

C++ et classes

C++ et hiérarchie

Classes Qt

widgets

QMainWindow

Signal et Slot

Projet Qt

EDI

Docs et liens

Layouts

```
MaClasse monObjet1(1), monObjet2(2); // instancie 2 objets

//le signal et le slot doivent être compatibles (même signature)
QObject :connect(&monObjet1, SIGNAL(send(int)), &monObjet2, SLOT(receive(int)));

monObjet1.emettre();
```

*//la méthode emettre() de l'objet1 enverra le signal send avec la valeur 1,
//ce qui déclenchera l'exécution du slot receive de l'objet2 et
//cela affichera "signal reçu 1"*



Exemple de connexions

L3 Info. CDAA

R. Raffin

Présentation
générale

Programmation
événementielle

Structure
générale de Qt
Multi-plateforme

Quelques rappels
de C++

Exemple basique de
source C++

C++ et containers

C++ et compilation

C++ et classes

C++ et hiérarchie

Classes Qt

widgets

QMainWindow

Signal et Slot

Projet Qt

EDI

Docs et liens

Layouts

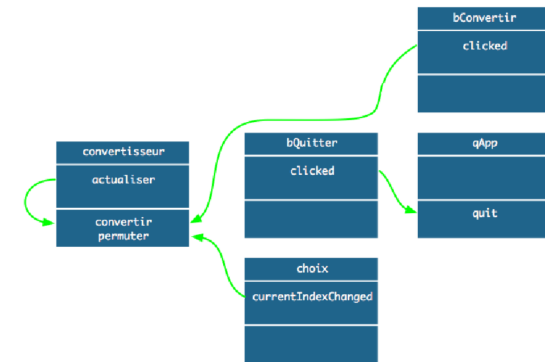


Fig. : Exemple de connexions



Plan intermédiaire

L3 Info. CDAA

R. Raffin

Présentation
générale

Programmation
événementielle

Structure
générale de Qt
Multi-plateforme

Quelques rappels
de C++

Exemple basique de
source C++

C++ et containers

C++ et compilation

C++ et classes

C++ et hiérarchie

Classes Qt

widgets

QMainWindow

Signal et Slot

Projet Qt

EDI

Docs et liens

Layouts

- 1 Présentation générale
- 2 Programmation événementielle
- 3 Structure générale de Qt
 - Multi-plateforme
- 4 Quelques rappels de C++
 - Exemple basique de source C++
 - C++ et containers
 - C++ et compilation
 - C++ et classes
 - C++ et hiérarchie
- 5 Classes Qt
 - widgets
 - QMainWindow
 - Signal et Slot
- 6 Projet Qt
 - EDI
 - Docs et liens
 - Layouts



Qu'est-ce qu'un projet Qt ?

L3 Info. CDAA

R. Raffin

Présentation
générale

Programmation
événementielle

Structure
générale de Qt
Multi-plateforme

Quelques rappels
de C++

Exemple basique de
source C++

C++ et containers

C++ et compilation

C++ et classes

C++ et hiérarchie

Classes Qt

widgets

QMainWindow

Signal et Slot

Projet Qt

EDI

Docs et liens

Layouts

Un projet Qt est défini par un fichier d'extension `.pro` décrivant la liste des fichiers sources, les dépendances, les paramètres passés au compilateur, etc.

Le fichier de projet `.pro` est fait pour être très facilement éditable par un développeur. Il consiste en une série d'affectations de variables.

Pour contrôler ses propres utilitaires (`moc`, `uic`, ...), Qt fournit un moteur de production spécifique : le programme `qmake`.

`qmake` prend en entrée un fichier de projet `.pro` et génère en sortie un fichier de fabrication spécifique à la plateforme. Ainsi, avec les systèmes UNIX/Linux, `qmake` produira un `Makefile`.



Étapes par étapes (1/2)

L3 Info. CDAA

R. Raffin

Présentation
générale

Programmation
événementielle

Structure
générale de Qt

Multi-plateforme

Quelques rappels
de C++

Exemple basique de
source C++

C++ et containers

C++ et compilation

C++ et classes

C++ et hiérarchie

Classes Qt

widgets

Qt4 et Qt5

Signal et Slot

Projet Qt

EDI

Docs et liens

Layouts

```
1 //on crée un répertoire et on se déplace à l'intérieur
2 $ mkdir exemple ; cd exemple
3
4 //on édite un simple fichier C++ main.cpp (avec vim par exemple)
5 $ vim main.cpp
6
7 //on génère le fichier de projet Qt .pro
8 $ qmake -project
9
10 $ ls
11 exemple.pro main.cpp
12
13 //le fichier .pro décrit le contenu du projet en une série d'affectations de variables
14 $ cat exemple.pro
15 TEMPLATE = app
16 TARGET = exemple
17 DEPENDPATH += .
18 INCLUDEPATH += .
19
20 HEADERS +=
21 SOURCES += main.cpp
22 //on génère le fichier Makefile à partir du fichier .pro
23
24 $ qmake
25
26 $ ls
27 exemple.pro main.cpp Makefile
```

Étapes par étapes (2/2)

L3 Info. CDAA

R. Raffin

Présentation
générale

Programmation
événementielle

Structure
générale de Qt

Multi-plateforme

Quelques rappels
de C++

Exemple basique de
source C++

C++ et containers

C++ et compilation

C++ et classes

C++ et hiérarchie

Classes Qt

widgets

Qt4 et Qt5

Signal et Slot

Projet Qt

EDI

Docs et liens

Layouts

```
1 //on fabrique l'application
2 $ make
3 g++ -c -m64 -pipe -O2 -Wall -W -D_REENTRANT -DQT_WEBKIT -DQT_NO_DEBUG -DQT_GUI_LIB
4 -DQT_CORE_LIB -DQT_SHARED -I/usr/share/qt4/mkspecs/linux-g++-64 -I. -I/usr/include/qt4/
5 QtCore -I/usr/include/qt4/QtGui -I/usr/include/qt4 -I. -o main.o main.cpp
6 g++ -m64 -Wl,-O1 -o exemple main.o -L/usr/lib/x86_64-linux-gnu -lQtGui -lQtCore -lpthread
7
8 $ ls
9 exemple exemple.pro main.cpp main.o Makefile
10
11 //on exécute l'application
12 $ ./exemple
```

À chaque fois que l'on modifie le fichier .pro, il faudra exécuter à nouveau la commande `qmake` pour que celle-ci mette à jour le fichier `Makefile`. D'autre part, le fichier `Makefile` est toujours spécifique à la plateforme. Si vous changez de plateforme (Linux, Windows, MacOS), il vous suffira d'exécuter à nouveau la commande `qmake` pour générer un fichier `Makefile` adapté à votre système.

Le fichier de projet .pro

L3 Info. CDAA

R. Raffin

Présentation
générale

Programmation
événementielle

Structure
générale de Qt

Multi-plateforme

Quelques rappels
de C++

Exemple basique de
source C++

C++ et containers

C++ et compilation

C++ et classes

C++ et hiérarchie

Classes Qt

widgets

Qt4 et Qt5

Signal et Slot

Projet Qt

EDI

Docs et liens

Layouts

Le principe de fonctionnement du fichier .pro est simple. La variable permettant d'indiquer les modules Qt à intégrer pour l'application se nomme `QT` :

```
QT += sql xml //pour activer les modules SQL et XML
QT -= gui //pour désactiver le module gui
```

D'autres variables utiles :

```
#Répertoires (cf. shadow build)
DESTDIR = bin

#pour l'exécutable
OBJECTS_DIR = build

#pour les fichiers objets (.o, .obj)
MOC_DIR = build

#pour les fichiers générés par moc
UI_DIR = build

#pour les fichiers générés par uic
FORMS = WiimoteIHM.ui # fiche(s) de Qt Designer

#Bibliothèques (multiplateforme)

linux*:LIBS += -lm -lwiuse
macx:LIBS += -framework IOKit -framework CoreFoundation
win32:LIBS += -lsetupapi -ladvapi32 -luser32
INCLUDEPATH += . # -> -I.
DEFINES += DEBUG # -> -DDEBUG
```

EDI/IDE

L3 Info. CDAA

R. Raffin

Présentation
générale

Programmation
événementielle

Structure
générale de Qt

Multi-plateforme

Quelques rappels
de C++

Exemple basique de
source C++

C++ et containers

C++ et compilation

C++ et classes

C++ et hiérarchie

Classes Qt

widgets

Qt4 et Qt5

Signal et Slot

Projet Qt

EDI

Docs et liens

Layouts

Qt Creator est l'environnement de développement intégré dédié à Qt et facilite la gestion d'un projet. Son éditeur de texte offre les principales fonctions que sont la coloration syntaxique, le complètement, l'indentation...

Qt Creator intègre en son sein les outils Qt Designer, Qt Linguist et Qt Assistant. Il intègre aussi un mode débbugage et beaucoup d'autres plugins.

Qt Creator lit en entrée un fichier de projet .pro. Il fournit aussi des assistants (*wizard*) pour créer des projets-types.

L3 Info. CDAA

R. Raffin

Présentation générale

Programmation événementielle

Structure générale de Qt

Multi-plateforme

Quelques rappels de C++

Exemple basique de source C++

C++ et containers

C++ et compilation

C++ et classes

C++ et hiérarchie

Classes Qt

widgets

Qt4 et Qt5

Signal et Slot

Projet Qt

EDI

Docs et liens

Layouts

Même si Qt Creator est présenté comme l'environnement de développement de référence pour Qt, il existe des modules pour les environnements de développement Eclipse et Visual Studio. Il existe d'autres EDI développés indépendamment de Nokia, comme QDevelop et Monkey Studio.

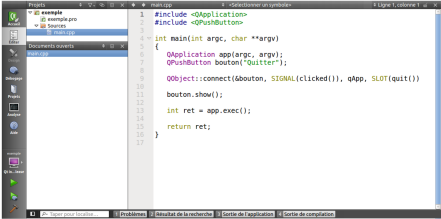


Fig. : QtCreator

L3 Info. CDAA

R. Raffin

Présentation générale

Programmation événementielle

Structure générale de Qt

Multi-plateforme

Quelques rappels de C++

Exemple basique de source C++

C++ et containers

C++ et compilation

C++ et classes

C++ et hiérarchie

Classes Qt

widgets

Qt4 et Qt5

Signal et Slot

Projet Qt

EDI

Docs et liens

Layouts

Qt Designer

QtDesigner est un logiciel qui permet de créer des interfaces graphiques Qt dans un environnement convivial. L'utilisateur, par glisser-déposer, place les composants d'interface graphique et y règle leurs propriétés facilement. Les fichiers d'interface graphique sont formatés en XML et portent l'extension .ui. Lors de la compilation, un fichier d'interface graphique est converti en classes C++ par l'utilitaire uic, fourni par Qt.

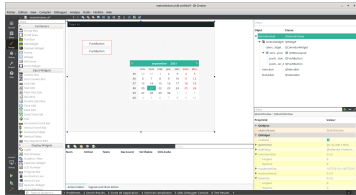


Fig. : Qt Designer

L3 Info. CDAA

R. Raffin

Présentation générale

Programmation événementielle

Structure générale de Qt

Multi-plateforme

Quelques rappels de C++

Exemple basique de source C++

C++ et containers

C++ et compilation

C++ et classes

C++ et hiérarchie

Classes Qt

widgets

Qt4 et Qt5

Signal et Slot

Projet Qt

EDI

Docs et liens

Layouts

Assistant nouveau projet

Dans QtCreator : Fichier -> Nouveau fichier ou projet...

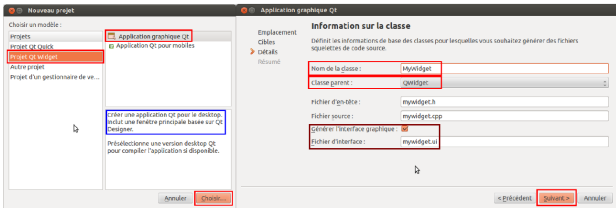


Fig. : IDE : nouveau projet

L3 Info. CDAA

R. Raffin

Présentation générale

Programmation événementielle

Structure générale de Qt

Multi-plateforme

Quelques rappels de C++

Exemple basique de source C++

C++ et containers

C++ et compilation

C++ et classes

C++ et hiérarchie

Classes Qt

widgets

Qt4 et Qt5

Signal et Slot

Projet Qt

EDI

Docs et liens

Layouts

Shadow build ?

Le *shadow build* permet de séparer les dossiers contenant les codes sources des fichiers générés (exécutable, fichiers objets .o...) par la fabrication (*build*). Il est important de ne pas les mélanger pour assurer convenablement les tâches courantes (sauvegarde, gestion de versions, portabilité multiplateforme...). Les fichiers issus de la compilation seront donc stockés dans un répertoire séparé.

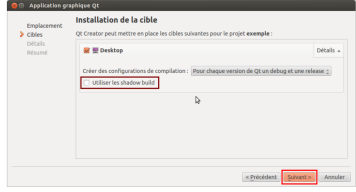


Fig. : IDE : shadow build

Les différents layouts

L3 Info, CDAA

R. Raffin

Présentation
générale

Programmation
événementielle

Structure
générale de Qt
Multi-plateforme

Quelques rappels
de C++

Exemple basique de
source C++
C++ et containers
C++ et compilation
C++ et classes
C++ et hiérarchie

Classes Qt

widgets

Qt5.10.1 online

Signal et Slot

Projet Qt

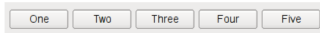
EDI

Docs et liens

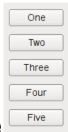
Layouts

Toutes les classes héritent de la classe abstraite QLayout :

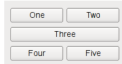
■ QHBoxLayout : boîte horizontale



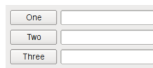
■ QVBoxLayout : boîte verticale



■ QGridLayout : grille



■ QFormLayout : formulaire



Pourquoi les Layouts ?

L3 Info, CDAA

R. Raffin

Présentation
générale

Programmation
événementielle

Structure
générale de Qt
Multi-plateforme

Quelques rappels
de C++

Exemple basique de
source C++
C++ et containers
C++ et compilation
C++ et classes
C++ et hiérarchie

Classes Qt

widgets

Qt5.10.1 online

Signal et Slot

Projet Qt

EDI

Docs et liens

Layouts

Lorsqu'un layout est défini sur un widget, il prend en charge les tâches suivantes :

- le positionnement des widgets enfants
- la gestion des tailles (minimale, préférée...)
- le redimensionnement, le déplacement
- la mise à jour automatique lorsque le contenu change

D'une manière générale, les widgets sont hiérarchiquement inclus les uns dans les autres. Ici, le principal avantage est que si le parent est déplacé, les enfants le sont aussi.

Organisation

L3 Info, CDAA

R. Raffin

Présentation
générale

Programmation
événementielle

Structure
générale de Qt
Multi-plateforme

Quelques rappels
de C++

Exemple basique de
source C++
C++ et containers
C++ et compilation
C++ et classes
C++ et hiérarchie

Classes Qt

widgets

Qt5.10.1 online

Signal et Slot

Projet Qt

EDI

Docs et liens

Layouts

Les gestionnaires de disposition (les classes QxxxLayout) simplifient le travail de positionnement :

- on peut ajouter des widgets dans un layout :
`void QLayout::addWidget(QWidget *widget)`
- on peut ajouter des layouts dans un layout :
`void QLayout::addLayout(QLayout *layout)`
- on peut associer un layout à un widget qui devient alors le propriétaire du layout et parent des widgets inclus dans le layout :
`void QWidget::setLayout(QLayout *layout)`

Combinaison de layouts

L3 Info, CDAA

R. Raffin

Présentation
générale

Programmation
événementielle

Structure
générale de Qt
Multi-plateforme

Quelques rappels
de C++

Exemple basique de
source C++
C++ et containers
C++ et compilation
C++ et classes
C++ et hiérarchie

Classes Qt

widgets

Qt5.10.1 online

Signal et Slot

Projet Qt

EDI

Docs et liens

Layouts

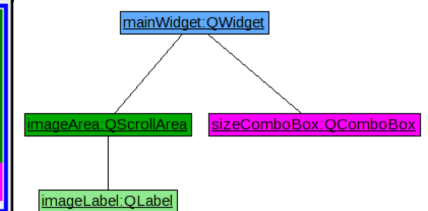
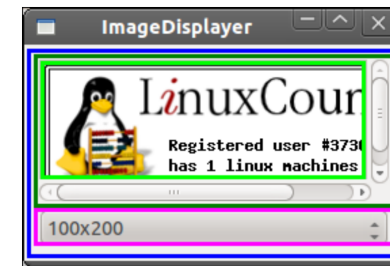


Fig. : Exemple de combinaison

Création des widgets

```
1 int main(int argc, char *argv[]) {
2     QApplication a(argc, argv);
3
4     //le widget central
5     QWidget mainWidget;
6
7     //une image
8     QLabel * imageLabel = new QLabel;
9     imageLabel->setPixmap(QPixmap("373058.png"));
10
11    //une zone de défilement
12    QScrollArea * imageArea = new QScrollArea;
13    imageArea->setWidget(imageLabel); // imageArea parent de imageLabel
14
15    //une liste déroulante
16    QComboBox * sizeComboBox = new QComboBox;
17    sizeComboBox->addItem("100x200");
18    sizeComboBox->addItem("200x400");
19 }
```



Gestion du positionnement

```

1 //un layout vertical
2 QVBoxLayout * layout = new QVBoxLayout;
3
4 //on ajoute les widgets au layout (parent de imageArea et de sizeComboBox)
5 layout->addWidget(imageArea);
6 layout->addWidget(sizeComboBox);
7
8 //on applique le layout au widget central
9 mainWindow.setLayout(layout); //parent de layout
10
11 //on affiche le widget
12 mainWindow.show(); //affichage de mainWindow et de ses enfants
13 return a.exec(); //boucle d'attente d'événements
14
15 //destruction de mainWindow et de ses enfants :
16 //-> destruction de layout
17 //-> destruction de sizeComboBox
18 //-> destruction de imageArea
19 //-> destruction de imageLabel
20 }

```

```

1 QVBoxLayout * layout = new QVBoxLayout;
2
3
4 // on ajoute les widgets au layout (parent de imageArea et de sizeComboBox)
5 layout->addWidget(imageArea);
6 layout->addWidget(sizeComboBox);
7
8 //on applique le layout au widget central
9 mainWindow.setLayout(layout); //parent de layout
10
11 //on affiche le widget
12 mainWindow.show(); //affichage de mainWindow et de ses enfants
13 return a.exec(); //boucle d'attente d'événements
14
15 //destruction de mainWindow et de ses enfants :
16 //-> destruction de layout
17 //-> destruction de sizeComboBox
18 //-> destruction de imageArea
19 //-> destruction de imageLabel
20 }

```

```

4 // on ajoute les widgets au layout (parent de imageArea et de sizeComboBox)
5 layout->addWidget(imageArea);
6 layout->addWidget(sizeComboBox);
7
8 //on applique le layout au widget central
9 mainWindow.setLayout(layout); //parent de layout
10
11 //on affiche le widget
12 mainWindow.show(); //affichage de mainWindow et de ses enfants
13 return a.exec(); //boucle d'attente d'événements
14
15 //destruction de mainWindow et de ses enfants :
16 //-> destruction de layout
17 //-> destruction de sizeComboBox
18 //-> destruction de imageArea
19 //-> destruction de imageLabel
20 }

```

```

5 layout->addWidget(sizeComboBox);
6
7 //on applique le layout au widget central
8 mainWidget.setLayout(layout); //parent de layout
9
10 //on affiche le widget
11 mainWidget.show(); //affichage de mainWidget et de ses enfants
12 return a.exec(); //boucle d'attente d'événements
13
14 //destruction de mainWidget et de ses enfants :
15 //-> destruction de layout
16 //-> destruction de sizeComboBox
17 //-> destruction de imageArea
18 //-> destruction de imageLabel
19 }
20

```

```

8 //on applique le layout au widget central
9 mainWidget.setLayout(layout); //parent de layout
10
11 //on affiche le widget
12 mainWidget.show(); //affichage de mainWidget et de ses enfants
13 return a.exec(); //boucle d'attente d'événements
14
15 //destruction de mainWidget et de ses enfants :
16 //-> destruction de layout
17 //-> destruction de sizeComboBox
18 //-> destruction de imageArea
19 //-> destruction de imageLabel
20 }

```

```

11 //on affiche le widget
12 mainWidget.show(); //affichage de mainWidget et de ses enfants
13 return a.exec(); //boucle d'attente d'événements
14
15 //destruction de mainWidget et de ses enfants :
16 //-> destruction de layout
17 //-> destruction de sizeComboBox
18 //-> destruction de imageArea
19 //-> destruction de imageLabel
20 }

```

```

13 return a.exec(); //boucle d'attente d'évenements
14
15 //destruction de mainWindow et de ses enfants :
16 //-> destruction de layout
17 //-> destruction de sizeComboBox
18 //-> destruction de imageArea
19 //-> destruction de imageLabel
20 }

```

```
15 //destruction de rialInnvidjet et de ses enfants .
16 //-> destruction de layout
17 //-> destruction de sizeComboBox
18 //-> destruction de imageArea
19 //-> destruction de imageLabel
20 }
```

```
17 //-> destruction de sizeCoulMoulin
18 //-> destruction de imageArea
19 //-> destruction de imageLabel
20 }
```

```
17 } // destruction de ImageLabel
18
19 }
```

[EDI](#)
[Docs et liens](#)
[Layouts](#)

Navigation icons: back, forward, search, etc.

```

1 //on layout vertical
2 QVBoxLayout *layout = new QVBoxLayout;
3
4 //on ajoute les widgets au layout (parent de imageArea et de sizeComboBox)
5 layout->addWidget(imageArea);
6 layout->addWidget(sizeComboBox);
7
8 //on applique le layout au widget central
9 mainWindow.setLayout(layout); //parent de layout
10
11 //on affiche le widget
12 mainWindow.show(); //affichage de mainWindow et de ses enfants
13 return a.exec(); //boucle d'attente d'événements
14
15 //destruction de mainWindow et de ses enfants :
16 //-> destruction de layout
17 //-> destruction de sizeComboBox
18 //-> destruction de imageArea
19 //-> destruction de imageLabel
20 }

```



2e partie

7 Construction d'une application

Plan intermédiaire

The image shows a presentation slide. At the top, there is a blue header bar with the text "L3 Info. CDAA" and "R. Ratin" in white. Below the header, the main content area is white. On the left side of the main content area, there is a large blue square containing the number "7" in white. To the right of this square, the text "Construction d'une application" is written in blue. At the bottom of the slide, there is a blue footer bar containing several small white icons for navigation, including arrows, a search icon, and a refresh icon.

Construction d'une application

7 Construction d'une application

7 Construction d'une application

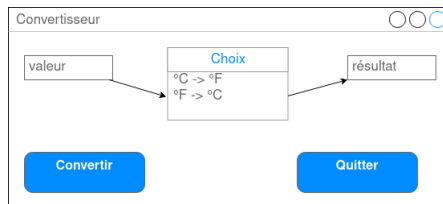
Application d'exemple

L3 Info, CDAA

R. Raffin

Construction
d'une application

On veut construire une application qui permette de convertir des degrés Celsius en Fahrenheit et vice-versa. On ajoutera 2 boutons à l'application : pour déclencher la conversion et pour quitter l'application. Un « mock-up »⁸ de l'application pourrait être le suivant :



⁸Maquette de l'interface, voir <https://www.usabilis.com/definition-mockup/>

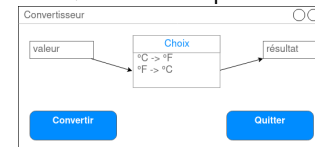
Traduction en composants

L3 Info, CDAA

R. Raffin

Construction
d'une application

À partir de cette définition et de l'ébauche d'interface, il nous faut définir les composants de Qt à mettre en place :



- 2 zones de texte pour l'entrée et le résultat
- 2 boutons pour « Convertir » et « Quitter »
- 1 bouton de choix du type de conversion

Et les connexions ?

L3 Info, CDAA

R. Raffin

Construction
d'une application

Il nous faut :

- 1 signal/slot sur le bouton « Quitter » pour fermer la fenêtre,
- 1 signal/slot sur le bouton « Convertir » pour calculer le résultat et l'afficher (2 actions),
- peut-être (cf le texte), une action lorsque le type de conversion est modifiée.

Il faut ensuite aller chercher dans la documentation quels sont les signaux/slots possibles, intrinsèques à Qt et lesquels développer :

- pour un `QPushButton` le signal `clicked()` existe,
- pour un `QWidget` le slot `close()` existe

Il nous manque donc un slot (une *action*) déclenché(e) par le bouton « Convertir ». Il faut implémenter une fonction pour cela.

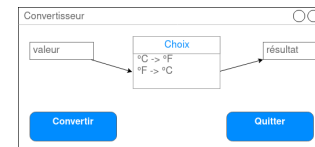
Et les positionnements ?

L3 Info, CDAA

R. Raffin

Construction
d'une application

D'après la maquette on a 2 zones horizontales, empilées verticalement. On aura donc besoin de 3 layouts :

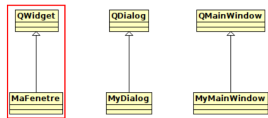


- 1 layout horizontal pour l'entrée, le type de conversion, le résultat
- 1 layout horizontal pour les 2 boutons « Convertir » et « Quitter »
- 1 layout vertical pour empiler les 2 précédents (de haut en bas)

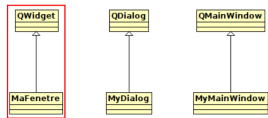
Fenêtre Qt

L3 Info. CDAA
R. Raffin
Construction
d'une application

On commence le développement par le choix du widget principal. Pour cette application nous utiliserons un `QWidget` plutôt qu'une `QMainWindow` ou un `QDialog`. Cela permettra de construire un composant, réutilisable dans une autre application.

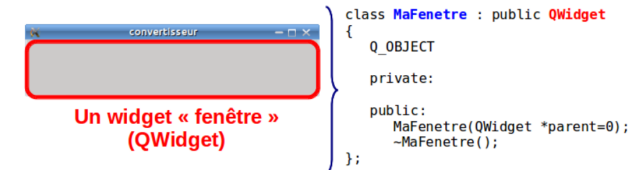


Ensuite, on composera notre fenêtre personnalisée en y intégrant des widgets :



Un widget sans parent = une fenêtre

L3 Info. CDAA
R. Raffin
Construction
d'une application



↑

```
int main( int argc, char **argv )
{
    QApplication a( argc, argv );

    MaFenetre w; // rappel : pas de parent = fenêtre !
    w.show();

    return a.exec();
}
```

Étape 1 : création de la fenêtre

L3 Info. CDAA
R. Raffin
Construction
d'une application

Dans le fichier « `maFenetre.h` » (peut-être défini par le projet Qt) :

```
1 class maFenetre : public QWidget
2 {
3     Q_OBJECT
4
5 public:
6     maFenetre(QWidget *parent = nullptr);
7     ~maFenetre();
8 };
```

Étape 1 : création de la fenêtre

L3 Info. CDAA
R. Raffin
Construction
d'une application

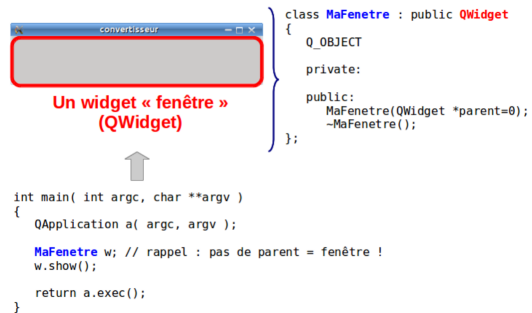
Dans le fichier « `maFenetre.cpp` » (peut-être aussi défini par le projet Qt) :

```
1 #include "maFenetre.h"
2
3 maFenetre::maFenetre(QWidget *parent)
4     : QWidget(parent)
5 {
6 }
7
8 maFenetre::~~maFenetre()
9 {
10 }
```

Composition des widgets

L3 Info. CDAA
R. Raffin
Construction
d'une application

On aura besoin de composer les widgets graphiques comme propriétés de « maFenetre » :



Étape 2 : déclaration des widgets

L3 Info. CDAA
R. Raffin
Construction
d'une application

Dans le fichier « maFenetre.h » on ajoute chaque widget :

```
1 //les include
2 //pour les widgets
3 #include <QLineEdit>
4 #include <QLabel>
5 #include <QComboBox>
6 #include <QPushButton>
7
8 class maFenetre : public QWidget
9 {
10     Q_OBJECT
11
12     private :
13     //nos widgets, par pointeurs
14     QLineEdit *valeur;
15     QLabel *resultat;
16     QLabel *unite;
17     QComboBox *choixConversion;
18     QPushButton *bConvertir;
19     QPushButton *bQuitter;
20
21 };
```

Étape 2 : instantiation des widgets

L3 Info. CDAA
R. Raffin
Construction
d'une application

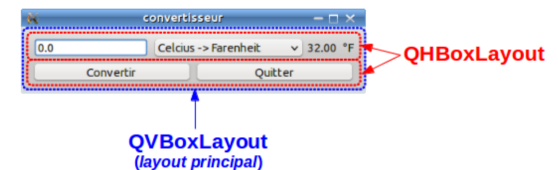
Dans le fichier « maFenetre.cpp » on instancie chaque widget :

```
1 maFenetre::maFenetre(QWidget *parent)
2     : QWidget(parent)
3 {
4     ...
5     //instanciation des widgets
6     valeur = new QLineEdit(this);
7     resultat = new QLabel("--", this);
8     unite = new QLabel(this);
9     choixConversion = new QComboBox(this);
10    bConvertir = new QPushButton("Convertir", this);
11    ...
12 }
```

Étape 3 : layouts et positionnement

L3 Info. CDAA
R. Raffin
Construction
d'une application

Comment (et pourquoi) positionner nos widgets intérieurs ?



Étape 3 : *layouts* et positionnement

L3 Info. CDAA

R. Raffin

Construction
d'une application

Dans le fichier « *maFenetre.cpp* » on crée les 3 layouts (2 horizontaux et verticaux, on leur relie les widgets et on les organise) :

```
1 ...
2 //2 widgets horizontaux
3 layout1->addWidget(valeur);
4 layout1->addWidget(choixConversion);
5 layout1->addWidget(resultat);
6 layout1->addWidget(unite);
7 ...
8 layout2->addWidget(bConvertir);
9 layout2->addWidget(bQuitter);
10 ...
11 //ajout des 2 layouts horizontaux au layout vertical
12 vlayout->addLayout(layout1);
13 vlayout->addLayout(layout2);
14 ...
15 //indication du layout de la fenêtre
16 this->setLayout(vlayout);
17 ...
```

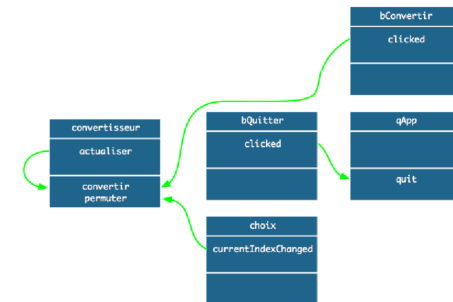
Étape 4 : signaux et connexions

L3 Info. CDAA

R. Raffin

Construction
d'une application

Comment (et pourquoi) mettre en place des signaux/slots ?



Étape 4 : signaux et connexions

L3 Info. CDAA

R. Raffin

Construction
d'une application

Dans le fichier « *maFenetre.h* » on spécifie un slot pour le calcul de la conversion :

```
1 class maFenetre : public QWidget
2 {
3 private slots:
4     void convertir();
5 ...
6 };
```

Étape 4 : signaux et connexions

L3 Info. CDAA

R. Raffin

Construction
d'une application

Dans le fichier « *maFenetre.cpp* » on spécifie les *connect* et le corps de la fonction de calcul de conversion :

Et la fonction « *convertir()* » :

```
1 void maFenetre::convertir()
2 {
3     float calcul=0.0;
4 ...
5     switch (choixConversion->currentIndex()) {
6     case 0 :
7         calcul = valeur->text().toFloat();
8         calcul = calcul*9.0/5.0+32.0;
9         resultat->setText(QString::number(calcul, 'f', 2));
10        unite->setText(QString::fromUtf8("°F"));
11        break;
12     case 1 :
13         calcul = valeur->text().toFloat();
14         calcul = 5.0*(calcul-32.0)/9.0;
15         resultat->setText(QString::number(calcul, 'f', 2));
16         unite->setText(QString::fromUtf8("°C"));
17        break;
18     }
19 }
```

Application principale ?

L3 Info. CDAA
R. Raffin
Construction
d'une application

Celle-ci est très générale, les actions se faisant dans « maFenetre » :

Construction d'une application

```
1 #include "maFenetre.h"
2
3 #include <QApplication>
4
5 int main(int argc, char *argv[])
6 {
7     QApplication a(argc, argv);
8     maFenetre w;
9     w.show();
10    return a.exec();
11 }
```

Améliorations

L3 Info. CDAA
R. Ruffin
Construction
d'une application

Dans la version 2.0, on pourrait :

Construction d'une application

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ 🔍 ↺

3e partie

L3 Info. CDAA
R. Raffin
Boîtes de dialogue

8 Boîtes de dialogue

Plan intermédiaire

L3 Info. CDAA
R. Ruffin
Boîtes de dialogue

8 Boîtes de dialogue

Boîtes de dialogue

- ◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ 🔍 ↺

Héritage d'un QDialog

L3 Info, CDAA

R. Raffin

Boîtes de dialogue

Application principale

Ajouts aux projets

Retour sur les widgets :

ergonomie

QtCreator

QtDesigner

La création d'une « boîte de dialogue » est réalisée à partir d'une nouvelle classe qui hérite de `QDialog` :



Squelette d'une boîte de dialogue

L3 Info, CDAA

R. Raffin

Boîtes de dialogue

Application principale

Ajouts aux projets

Retour sur les widgets :

ergonomie

QtCreator

QtDesigner

Dans « mydialog.h »

```
1 class MyDialog : public QDialog
2 {
3     Q_OBJECT
4
5     private :
6
7     public :
8         MyDialog(QWidget *parent=0);
9
10    public slots :
11
12    signals :
13    };
```

Dans « mydialog.cpp »

```
1 #include "mydialog.h"
2
3 MyDialog : MyDialog(QWidget parent) : QDialog(parent)
4 {
5     //TODO
6 }
```

La classe QDialog

L3 Info, CDAA

R. Raffin

Boîtes de dialogue

Application principale

Ajouts aux projets

Retour sur les widgets :

ergonomie

QtCreator

QtDesigner

La classe `QDialog` est la classe de base des fenêtres de dialogue. Elle hérite de `QWidget`. Une fenêtre de dialogue (ou boîte de dialogue) est principalement utilisée pour des tâches de courte durée et de brèves communications avec l'utilisateur.

Une fenêtre de dialogue (ou boîte de dialogue) peut :

- être modale ou non modale,
- fournir une valeur de retour,
- avoir des boutons par défaut,
- posséder un `QSizeGrip` (une poignée de redimensionnement) dans le coin inférieur droit

Boîte de dialogue non modale

L3 Info, CDAA

R. Raffin

Boîtes de dialogue

Application principale

Ajouts aux projets

Retour sur les widgets :

ergonomie

QtCreator

QtDesigner

Une boîte de dialogue non modale (*modeless dialog*) est un dialogue qui fonctionne indépendamment des autres fenêtres de la même application.

Exemple : rechercher du texte dans les traitements de texte. Une boîte de dialogue non modale est affichée en utilisant `show()` qui retourne le contrôle à l'appelant immédiatement.

Remarque : si la boîte de dialogue est visuellement cachée, il suffira d'appeler successivement `show()`, `raise()` et `activateWindow()` pour la replacer sur le dessus de la pile.

Boîte de dialogue modale

L3 Info. CDAA

R. Raffin

Boîtes de dialogue

Application principale

Ajouts aux projets

Retour sur les widgets :

ergonomie

QtCreator

QtDesigner

Une boîte de dialogue modale (*modal dialog*) est un dialogue qui bloque l'entrée à d'autres fenêtres visibles de la même application.

Exemple : les dialogues qui sont utilisés pour demander un nom de fichier ou pour définir les préférences de l'application (couleur, police...) sont généralement modaux.

La façon la plus commune pour afficher une boîte de dialogue modale est de faire appel à sa fonction `exec()`. Lorsque l'utilisateur ferme la boîte de dialogue, `exec()` fournira une valeur de retour utile.



Les boîtes de dialogue en « français »

L3 Info. CDAA

R. Raffin

Boîtes de dialogue

Application principale

Ajouts aux projets

Retour sur les widgets :

ergonomie

QtCreator

QtDesigner

On doit faire appel au « traducteur » et spécifier une *locale*.

```
1 #include <QApplication>
2 #include <QTranslator>
3 #include <QLocale>
4 #include <QLibraryInfo>
5
6 int main(int argc, char *argv[])
7 {
8     QApplication a(argc, argv);
9     QString locale = QLocale::system().name().section('_', 0, 0);
10    QTranslator translator;
11    translator.load(QString("qt_") + locale, QLibraryInfo::location(
12        QLibraryInfo::TranslationsPath));
13
14    a.installTranslator(&translator);
15
16    //...
17    return a.exec();
18 }
```



La classe QMessageBox

L3 Info. CDAA

R. Raffin

Boîtes de dialogue

Application principale

Ajouts aux projets

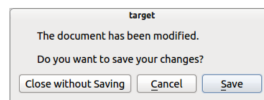
Retour sur les widgets :

ergonomie

QtCreator

QtDesigner

La classe `QMessageBox` fournit un dialogue modal pour informer l'utilisateur ou pour demander une information supplémentaire.



```
1 QMessageBox msgBox;
2
3 msgBox.setText("The document has been modified.");
4 msgBox.setInformativeText("Do you want to save your changes?");
5 msgBox.setStandardButtons(QMessageBox::Save | QMessageBox::Discard | QMessageBox::Cancel);
6
7 int ret = msgBox.exec();
```



La classe QMessageBox

L3 Info. CDAA

R. Raffin

Boîtes de dialogue

Application principale

Ajouts aux projets

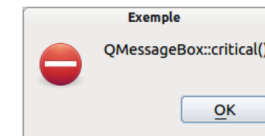
Retour sur les widgets :

ergonomie

QtCreator

QtDesigner

Elle fournit aussi quatre types prédéfinis : `QMessageBox::critical()`, `QMessageBox::information()`, `QMessageBox::question()`, `QMessageBox::warning()`.



```
1 QMessageBox::critical(0, "Exemple", "QMessageBox::critical()");
```



La classe QInputDialog

L3 Info. CDAA

R. Raffin

Boîtes de dialogue

Application principale

Ajouts aux projets

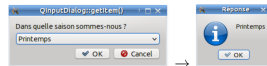
Retour sur les widgets :

ergonomie

QtCreator

QtDesigner

La classe `QInputDialog` fournit un dialogue simple pour obtenir une valeur unique de l'utilisateur. La valeur d'entrée peut être une chaîne, un numéro ou un élément d'une liste (`getText()`, `getInt()`, `getDouble()`, `getItem()`). Une étiquette (*Label*) doit être placée afin de préciser à l'utilisateur ce qu'il doit entrer.



```
1 QStringList items;
2 bool ok;
3 items << QString::fromUtf8("Printemps") << QString::fromUtf8("Été") << QString::fromUtf8("Automne") << QString::fromUtf8("Hiver");
4
5 QString item = QInputDialog::getItem(0, "QInputDialog::getItem()", "Dans quelle saison sommes-nous ?", items, 0, false, &ok);
6
7 if (ok && !item.isEmpty())
8     QMessageBox::information(0, QString::fromUtf8("Réponse"), item);
```

La classe QColorDialog

L3 Info. CDAA

R. Raffin

Boîtes de dialogue

Application principale

Ajouts aux projets

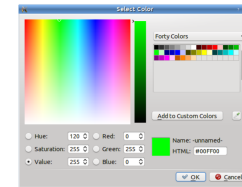
Retour sur les widgets :

ergonomie

QtCreator

QtDesigner

La classe `QColorDialog` fournit un dialogue pour la spécification des couleurs. Cela permet aux utilisateurs de choisir les couleurs (`getColor()`). Par exemple, vous pourriez l'utiliser dans un programme de dessin pour permettre à l'utilisateur de définir la couleur du pinceau.



```
1 QColor color = QColorDialog::getColor(Qt::green, 0);
2
3 if (color.isValid()) { //... }
```

La classe QFontDialog

L3 Info. CDAA

R. Raffin

Boîtes de dialogue

Application principale

Ajouts aux projets

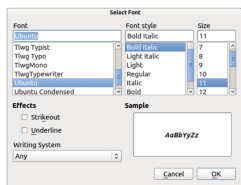
Retour sur les widgets :

ergonomie

QtCreator

QtDesigner

La classe `QFontDialog` fournit un widget de dialogue de sélection d'une police (`getFont()`).



```
1 bool ok;
2 QLabel pMonLabel("Hello world!");
3 QFont font = QFontDialog::getFont(&ok, pMonLabel.font());
4
5 if (ok) {
6     pMonLabel.setFont(font);
7     pMonLabel.show();
8 }
```

La classe QPrintDialog

L3 Info. CDAA

R. Raffin

Boîtes de dialogue

Application principale

Ajouts aux projets

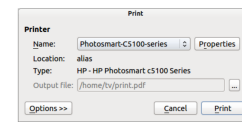
Retour sur les widgets :

ergonomie

QtCreator

QtDesigner

La classe `QPrintDialog` fournit une boîte de dialogue pour spécifier la configuration de l'imprimante et imprimer.



```
1 QPrinter printer;
2
3 QPrintDialog printDialog(&printer, 0);
4
5 if (printDialog.exec() == QDialog::Accepted)
6     { //impression...
7 }
```

La classe QFileDialog

L3 Info. CDAA

R. Raffin

Boîtes de
dialogue

Application
principale

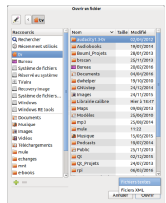
Ajouts aux projets

Retour sur les
widgets :
ergonomie

QtCreator

QtDesigner

La classe `QFileDialog` fournit une boîte de dialogue modal qui permet aux utilisateurs de sélectionner des fichiers ou des répertoires. Elle permet de parcourir le système de fichiers afin de sélectionner un ou plusieurs fichiers ou un répertoire (`getExistingDirectory()`, `getOpenFileName()`, `getOpenFileNames()`, `getSaveFileName()`).



```
1 QString fileName = QFileDialog::getOpenFileName(0, "
    Ouvrir un fichier", "/home/tv", "Fichiers textes
    (*.txt);;Fichiers XML (*.xml)");
```

La classe QMessageBox

L3 Info. CDAA

R. Raffin

Boîtes de
dialogue

Application
principale

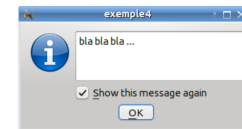
Ajouts aux projets

Retour sur les
widgets :
ergonomie

QtCreator

QtDesigner

La classe `QErrorMessage` fournit une boîte de dialogue non modale qui affiche un message d'erreur (`showMessage()`).



```
1 QMessageBox *errorMessageDialog;
2 errorMessageDialog = new QMessageBox;
3 errorMessageDialog->showMessage("bla bla bla ...");
```

Plan intermédiaire

L3 Info. CDAA

R. Raffin

Boîtes de
dialogue

Application
principale

Ajouts aux projets

Retour sur les
widgets :
ergonomie

QtCreator

QtDesigner

8 Boîtes de dialogue

9 Application principale

10 Ajouts aux projets

11 Retour sur les widgets : ergonomie

12 QtCreator

13 QtDesigner

Création d'une application « fenêtre principale »

L3 Info. CDAA

R. Raffin

Boîtes de
dialogue

Application
principale

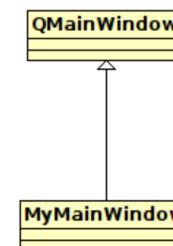
Ajouts aux projets

Retour sur les
widgets :
ergonomie

QtCreator

QtDesigner

La création d'une application « fenêtre principale » est réalisée à partir d'une nouvelle classe qui hérite de `QMainWindow` :



Squelette de l'application « fenêtre principale »

L3 Info, CDAA

R. Raffin

Boîtes de
dialogue

Application
principale

Ajouts aux projets

Retour sur les

widgets :

ergonomie

QtCreator

QtDesigner

Dans « mymainwindow.h »

```
1 class MyMainWindow : public QMainWindow
2 {
3     Q_OBJECT
4
5     private :
6
7     public :
8         MyMainWindow(QWidget *parent = 0);
9
10    public slots :
11
12    signals :
13    };
```

Dans « mymainwindow.cpp »

```
1 #include "mymainwindow.h"
2
3 MyMainWindow::MyMainWindow(QWidget *parent) :
4     QMainWindow(parent)
5 {
6     //TODO
7 }
```

La classe QMainWindow

L3 Info, CDAA

R. Raffin

Boîtes de
dialogue

Application
principale

Ajouts aux projets

Retour sur les

widgets :

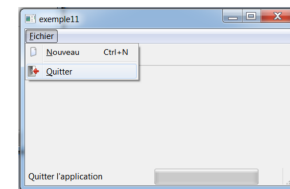
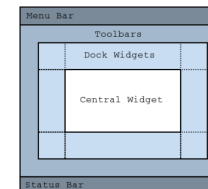
ergonomie

QtCreator

QtDesigner

La classe QMainWindow offre une fenêtre d'application principale.

Une fenêtre principale fournit un cadre pour la construction de l'interface utilisateur d'une application.



La structure d'un QMainWindow

L3 Info, CDAA

R. Raffin

Boîtes de
dialogue

Application
principale

Ajouts aux projets

Retour sur les

widgets :

ergonomie

QtCreator

QtDesigner

QMainWindow a sa propre mise en page à laquelle vous pouvez ajouter QToolBars, QDockWidgets, un QMenuBar, et un QStatusBar.

La fenêtre possède une zone centrale (central widget) qui peut être occupée par n'importe quel type de widget.

Le widget central sera généralement un widget standard de Qt comme un QTextEdit (logiciel de type « texte ») ou un QGraphicsView (logiciel de « dessin »). Les widgets personnalisés peuvent également être utilisés pour des applications avancées.

On définit le widget central avec setCentralWidget().

SDI ou MDI

L3 Info, CDAA

R. Raffin

Boîtes de
dialogue

Application
principale

Ajouts aux projets

Retour sur les

widgets :

ergonomie

QtCreator

QtDesigner

Une application qui gère des documents a soit une interface unique (SDI pour *Single Document Interface*) ou multiple (MDI pour *Multiple Document Interface*).

Pour créer des applications MDI dans Qt, on utilisera un QMdiArea comme widget central.

Les sous-fenêtres de QMdiArea sont des instances de QMdiSubWindow.

Elles sont ajoutées à une zone MDI avec addSubWindow().

L'interface MDI

L3 Info. CDAA

R. Raffin

Boîtes de dialogue

Application principale

Ajouts aux projets

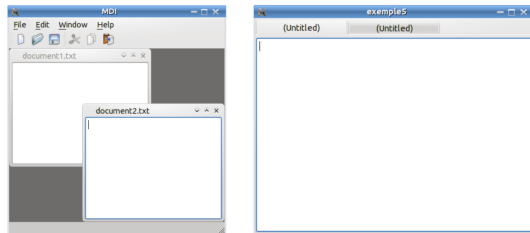
Retour sur les widgets :

ergonomie

QtCreator

QtDesigner

Avec `setViewMode()`, on peut choisir un affichage des sous-fenêtres soit indépendantes (`QMdiArea::SubWindowView`), soit regroupées en onglets (`QMdiArea::TabbedView`).



La classe QMdiArea

L3 Info. CDAA

R. Raffin

Boîtes de dialogue

Application principale

Ajouts aux projets

Retour sur les widgets :

ergonomie

QtCreator

QtDesigner

```
1 QMdiArea *mdiArea = new QMdiArea;
2
3 QTextEdit *textEdit1 = new QTextEdit;
4 QTextEdit *textEdit2 = new QTextEdit;
5
6 QMdiSubWindow *mdiSubWindow1 = mdiArea->addSubWindow(textEdit1);
7 QMdiSubWindow *mdiSubWindow2 = mdiArea->addSubWindow(textEdit2);
8
9 //ou : QMdiArea::SubWindowView
10
11 mdiArea->setViewMode(QMdiArea::TabbedView);
12 mdiArea->setCentralWidget(mdiArea);
```

La classe QAction

L3 Info. CDAA

R. Raffin

Boîtes de dialogue

Application principale

Ajouts aux projets

Retour sur les widgets :

ergonomie

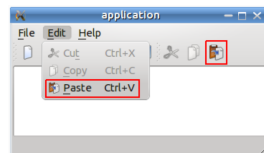
QtCreator

QtDesigner

La classe `QAction` fournit une interface abstraite pour décrire une action (= commande) qui peut être insérée dans les widgets.

Cela permet de créer des commandes communes pouvant être invoquées via des menus, boutons, et des raccourcis clavier.

Les actions peuvent être ajoutées aux menus et barres d'outils, et seront automatiquement synchronisées.



La classe QMenu

L3 Info. CDAA

R. Raffin

Boîtes de dialogue

Application principale

Ajouts aux projets

Retour sur les widgets :

ergonomie

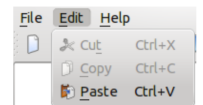
QtCreator

QtDesigner

La classe `QMenu` fournit un widget pour une utilisation dans les barres de menus et les menus contextuels. Un menu contextuel est un menu qui s'affiche lorsqu'on fait un clic droit sur un widget.

Un widget menu est un menu de sélection. Il peut être soit un menu déroulant dans une barre de menu ou un menu contextuel autonome.

Qt implémente donc les menus avec `QMenu` et `QMainWindow` les garde dans un `QMenuBar`. On utilise `QMenuBar::addMenu()` pour insérer un menu dans une barre de menu.



La classe QMenuBar

L3 Info, CDAA

R. Raffin

Boîtes de dialogue

Application principale

Ajouts aux projets

Retour sur les widgets :
ergonomie

QtCreator

QtDesigner

La classe `QMenuBar` fournit une barre de menu horizontale. Une barre de menu se compose d'une liste d'éléments de menu déroulant. On peut ajouter de nouveaux menus à la barre de menus de la fenêtre principale en appelant `menuBar()` qui retourne la `QMenuBar` de la fenêtre. On ajoute un menu avec `QMenuBar::addMenu()`.



```
1 QMenu *fileMenu = new QMenu(QString::fromUtf8("&Fichier"), this);
2 menuBar()->addMenu(fileMenu);
3
4 QMenu *editMenu = new QMenu(QString::fromUtf8("&Édition"), this);
5 menuBar()->addMenu(editMenu);
6 //...
```

Navigation icons

La classe QMenu

L3 Info, CDAA

R. Raffin

Boîtes de dialogue

Application principale

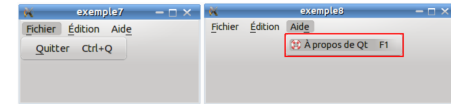
Ajouts aux projets

Retour sur les widgets :
ergonomie

QtCreator

QtDesigner

On peut soit créer une instance de `QAction` puis l'ajouter avec `addAction()` :



Solution 1 :

```
1 QAction *actionHelp = new QAction(QString::fromUtf8("À propos de Qt"), this);
2 actionHelp->setShortcut(QKeySequence(Qt::Key_F1));
3 actionHelp->setIcon(QIcon(":/help.png"));
4 connect(actionHelp, SIGNAL(triggered()), qApp, SLOT(aboutQt()));
5
6 helpMenu->addAction(actionHelp);
```

Navigation icons

La classe QMenu

L3 Info, CDAA

R. Raffin

Boîtes de dialogue

Application principale

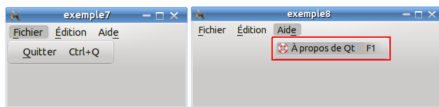
Ajouts aux projets

Retour sur les widgets :
ergonomie

QtCreator

QtDesigner

Soit créer la `QAction` directement en utilisant `addAction()` :



Solution 2 :

```
1 fileMenu->addAction(QString::fromUtf8("&Quitter"), qApp, SLOT(quit()),
   QKeySequence::Quit);
```

Navigation icons