

«cute» Creator

Alain MENU – octobre 2014 – version 3 (Qt 5.3.x)
projet support : DemoEditor.light

Cette création est mise à disposition selon le Contrat *Attribution-NonCommercial-ShareAlike* 2.0 France disponible en ligne <http://creativecommons.org/licenses/by-nc-sa/2.0/fr/> ou par courrier postal à Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.

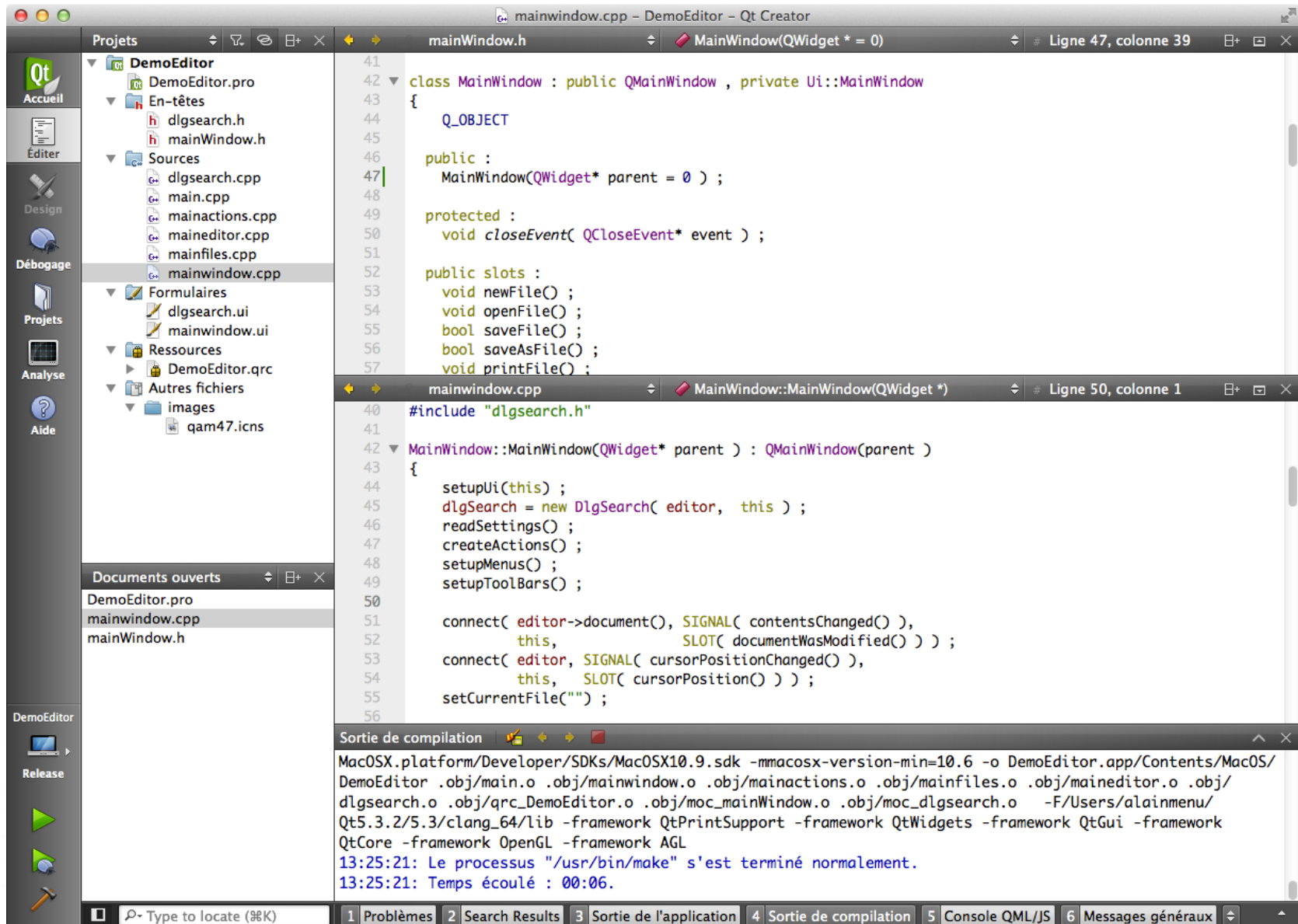


Présentation de l'EDI	3	Création d'une fenêtre fille	15
Création d'un nouveau projet	4	Classe d'interface du dialogue	16
Première compilation	6	Exécution du dialogue	18
Interface UI → code	7		
Création des actions	8	Architecture finale	19
Test du menu principal	10	Pour aller plus loin	20
Ajout de ressources	11		
Zone d'édition	13		
Chargement d'un fichier	14		



Cahier des charges :

Réaliser une application de type « bureautique » avec menu de commandes, barre d'outils et fenêtre filles...



Fichier | Nouveau fichier ou projet ...

➤ [Qt Widgets Application](#)

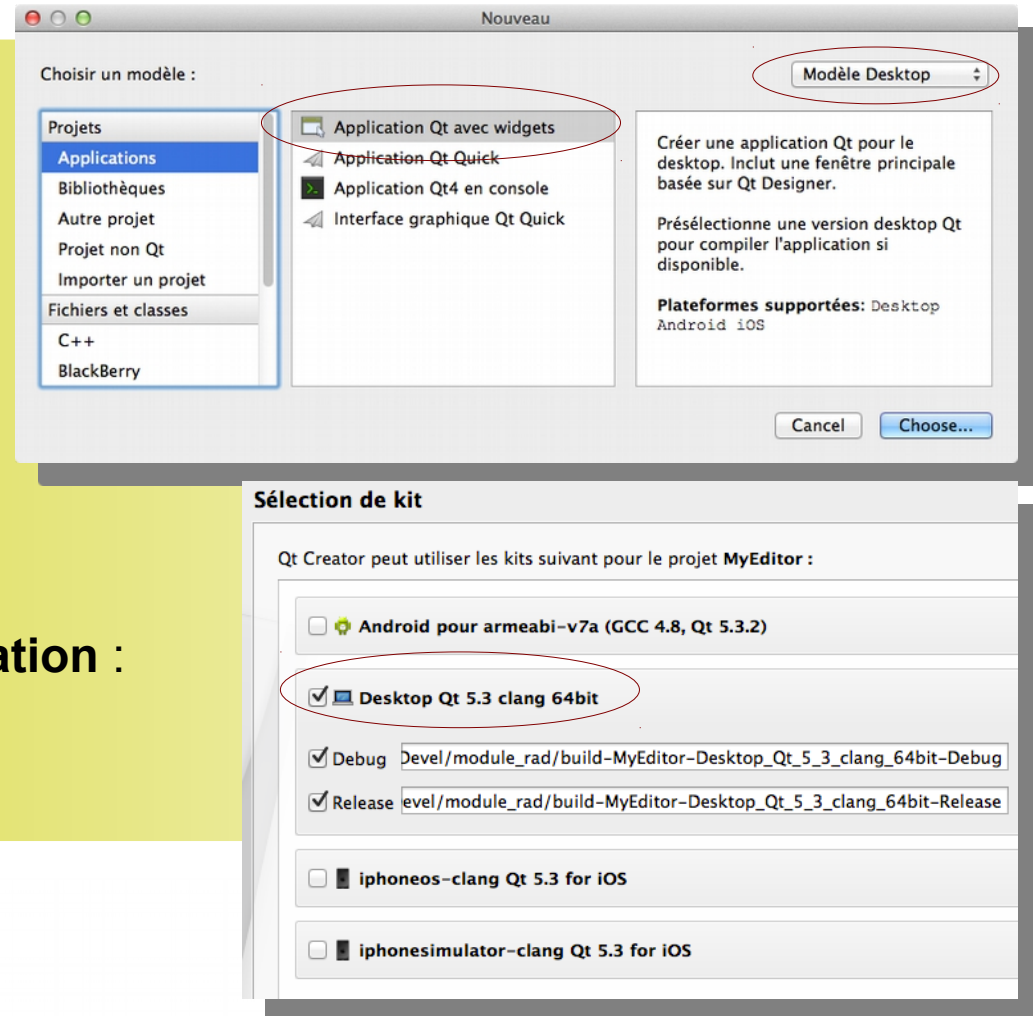
Choisir un nom de projet :

➤ [MyEditor](#)

et un répertoire de travail...

Sélectionner au moins un **kit de fabrication** :

➤ [Desktop](#) (debug et/ou release)



Choisir le nom de la classe UI principale

➡ **MainWindow** (par défaut)

et sa classe de base :

➡ **QMainWindow**

Valider le tout !

création automatique des fichiers :

➡ **MyEditor.pro**
main.cpp
mainwindow.h
mainwindow.cpp
mainwindow.ui

Information sur la classe

Définit les informations de base des classes pour lesquelles vous souhaitez générer des fichiers squelettes de code source.

Nom de la classe :

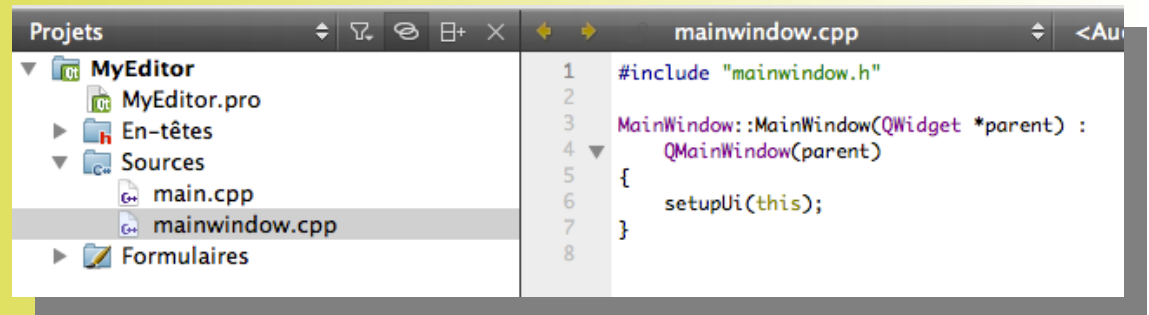
Classe parent :

Fichier d'en-tête :

Fichier source :

Générer l'interface graphique : ☒

Fichier d'interface :





Projets → désactiver « Shadow build »



personnaliser le fichier .pro ()*

sélectionner le type de cible :



Desktop **Release**

Compiler et lancer l'exécution par **Run** (Ctrl-R) !



suivi dans « Sortie de compilation »



lancement automatique....



suivi dans « Sortie de l'application »

1 Problèmes 2 Search Results 3 Sortie de l'application 4 **Sortie de compilation**

```

QT      += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

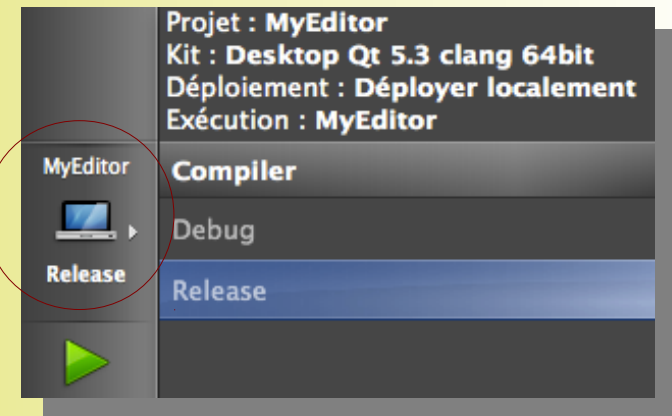
TARGET    = MyEditor
TEMPLATE  = app

UI_DIR    = .uic
RCC_DIR   = .uic
MOC_DIR   = .moc
OBJECTS_DIR = .obj

SOURCES   += main.cpp\
             mainwindow.cpp

HEADERS    += mainwindow.h

FORMS      += mainwindow.ui
  
```



(*) facultatif, pour conformité développement mode ligne de commande...

Code généré par l'assistant :

mainwindow.h

↳ héritage multiple

mainwindow.cpp

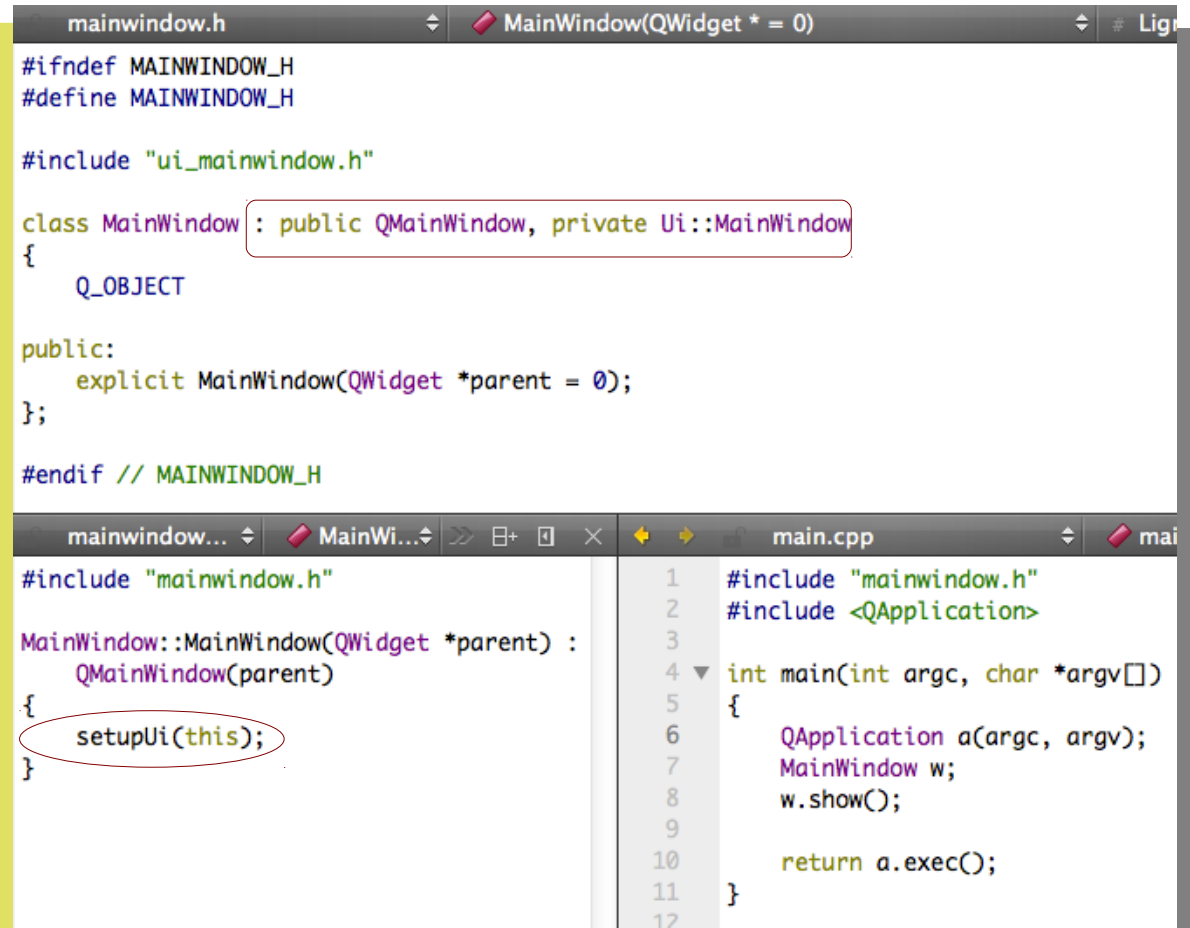
↳ constructeur → setupUi()

main.cpp

↳ création instance

↳ projection → show()

↳ boucle d'événements



```
mainwindow.h
MainWindow(QWidget * = 0)

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include "ui_mainwindow.h"

class MainWindow : public QMainWindow, private Ui::MainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
};

#endif // MAINWINDOW_H

mainwindow.cpp
#include "mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent)
{
    setupUi(this);
}

main.cpp
1 #include "mainwindow.h"
2 #include <QApplication>
3
4 int main(int argc, char *argv[])
5 {
6     QApplication a(argc, argv);
7     MainWindow w;
8     w.show();
9
10    return a.exec();
11 }
12
```

mainwindow.h

➤ Ajout de membres à la classe `MainWindow`

- entrées de menu principal (menus)
- entrées de menu déroulant (actions)
- slots d'interception

slot
« automatique »

mainwindow.cpp

- ajout de `#include <iostream>`
- implémentation provisoire des slots

private:

```
QMenu*   fileMenu ;
QAction*  openAction ;
QAction*  saveAction ;
QAction*  quitAction ;

QMenu*   editMenu ;
QAction*  searchAction ;
```

private slots :

```
void open() ;
void on_saveAction_triggered() ;
void search() ;
```

```
void MainWindow::open()
{
    std::cout << "open clicked !" << std::endl ;
}

void MainWindow::on_saveAction_triggered()
{
    std::cout << "save clicked !" << std::endl ;
}

void MainWindow::search()
{
    std::cout << "search clicked !" << std::endl ;
}
```


mainwindow.cpp

ajout d'instructions dans le constructeur, après l'appel à `setUpUi()`

↳ création des actions

connexion
« manuelle »

connexion
automatique (*)

```
openAction = new QAction( tr("&Ouvrir..."), this );
openAction->setShortcut( tr("Ctrl+O") );
openAction->setStatusTip( tr("Ouverture d'un fichier") );
connect( openAction, SIGNAL( triggered() ), this, SLOT( open() ) );

saveAction = new QAction( tr("&Sauvegarder"), this );
saveAction->setShortcut( tr("Ctrl+S") );
saveAction->setStatusTip( tr("Sauvegarde du document") );
saveAction->setObjectName("saveAction");

quitAction = new QAction( tr("&Quitter"), this );
quitAction->setShortcut( tr("Ctrl+Q") );
quitAction->setStatusTip( tr("Quitter l'application") );
connect( quitAction, SIGNAL( triggered() ), this, SLOT( close() ) );

QMetaObject::connectSlotsByName( this );

fileMenu = QMainWindow::menuBar()->addMenu( tr("&Fichier") );
fileMenu->addAction( openAction );
fileMenu->addAction( saveAction );
fileMenu->addSeparator();
fileMenu->addAction( quitAction );
```

slot hérité

↳ création du menu principal

(*) `void on_<objectname>_<signalname>(<signalparameters>);`

mainwindow.cpp

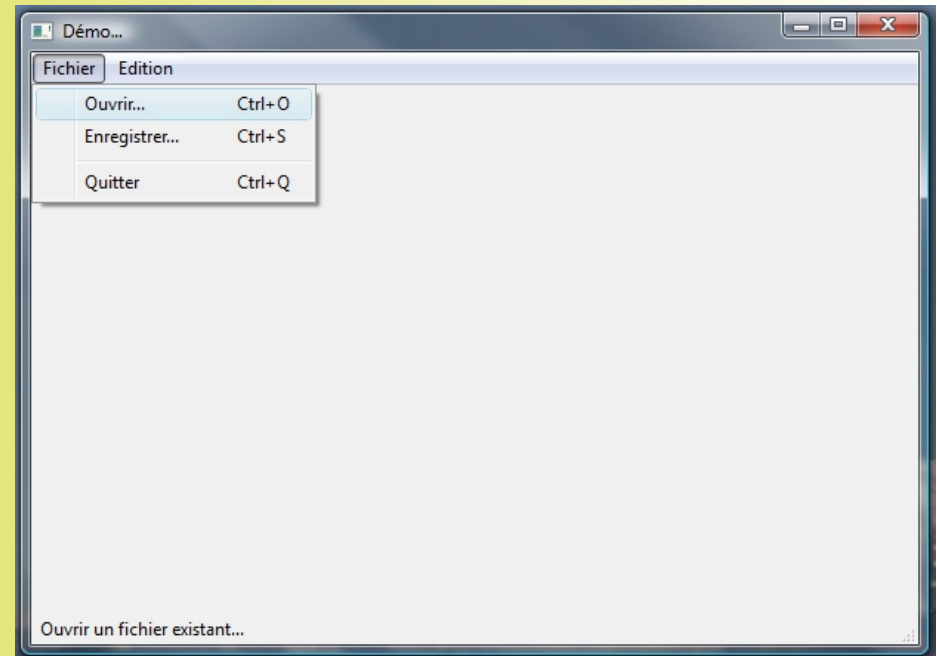
toujours dans le constructeur,

- ✚ ajout d'un titre de fenêtre et d'une invite dans la barre de statut...

```
setWindowTitle("Démonstration") ;  
QMainWindow::statusBar()->showMessage( tr("Prêt") ) ;
```

Test de compilation et d'exécution :

- ✚ la fenêtre dispose maintenant d'un menu principal avec une commande **Fichier | Quitter** opérationnelle !



*le répertoire projet contient un sous-rép.
images avec des icônes PNG...*

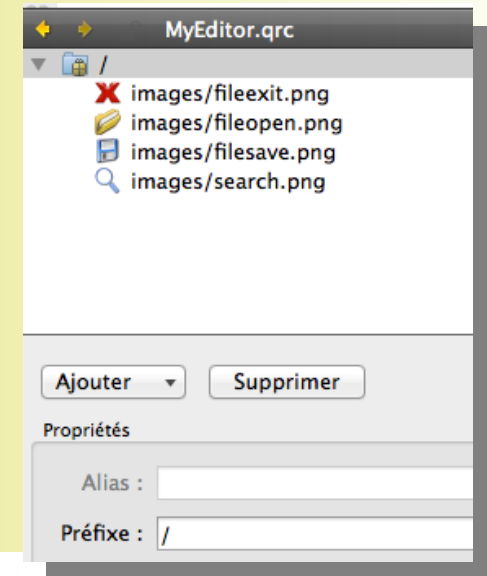
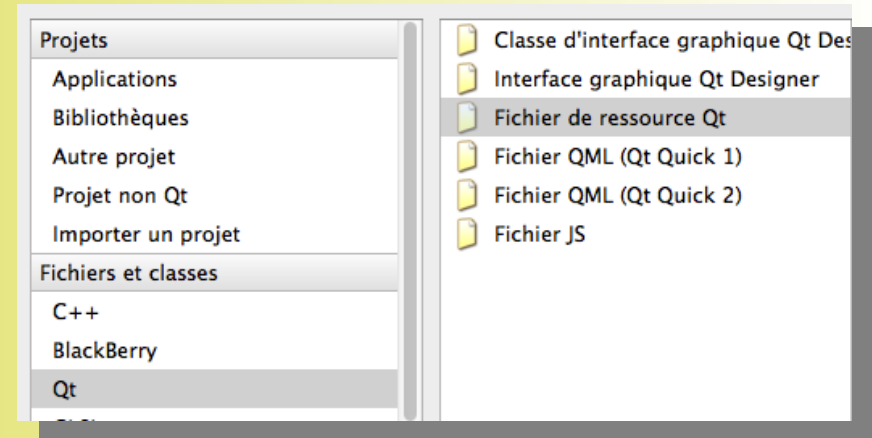
Fichier | Nouveau fichier ou projet ...

📁 **Fichier de ressource Qt**

choisir un nom :

📁 **MyEditor** → fichier **MyEditor.qrc**

Édition du qrc : ajouter un préfixe (/) et des fichiers →



mainwindow.h

↩ attribut(s) privé(s)

```
private:
    QToolBar*   fileToolBar ;
    QToolBar*   editToolBar ;
```

mainwindow.cpp

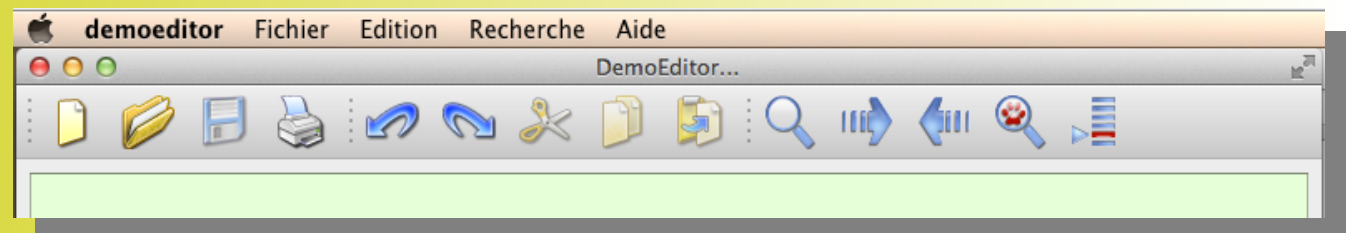
```
openAction = new QAction( QIcon(":/images/fileopen.png"), tr("&Ouvrir..."), this ) ;
/* ... */
saveAction = new QAction( QIcon(":/images/filesave.png"), tr("&Sauvegarder"), this ) ;
/* ... */
```

↩ nouvelle forme de création des actions

↩ construction de la barre d'outils

```
fileToolBar = addToolBar( tr("Fichier") ) ;
fileToolBar->addAction( openAction ) ;
fileToolBar->addAction( saveAction ) ;
/* ... */
```

Aperçu (version finale sous Mac) :



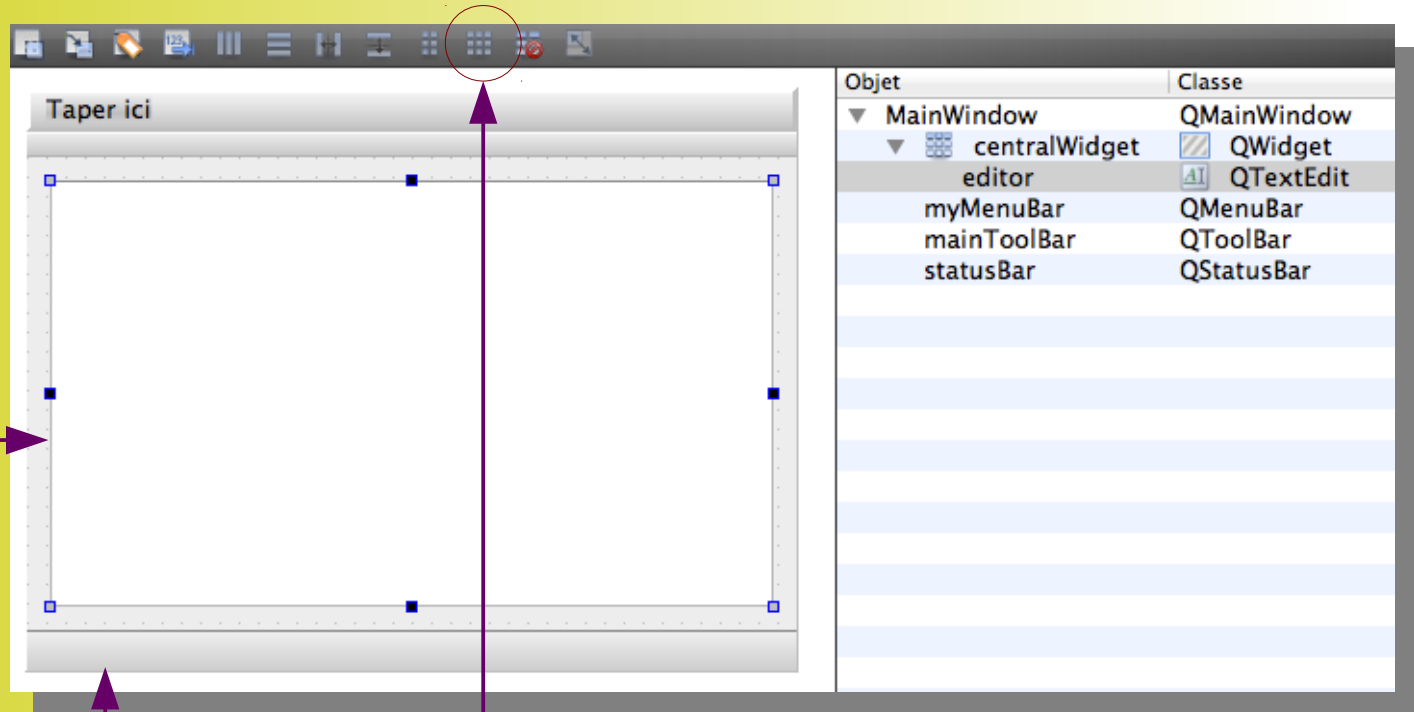
Le widget principal de la fenêtre d'application doit permettre l'édition de texte :



mainwindow.ui

insertion d'un
QTextEdit au milieu
de la fenêtre,

objectName = **editor**



sélection de la fenêtre et application du « Layout in a Grid »

Implémentation de la méthode `MainWindow::open()` :

```
#include <QFileDialog>
#include <QMessageBox>
#include <QTextStream>
```

Test de compilation
et d'exécution :

↪ la commande « Ouvrir... »
permet maintenant de
charger le contenu d'un
fichier texte dans la zone
d'édition de la fenêtre
d'application...

```
void MainWindow::open()
{
    QString fileName = QFileDialog::getOpenFileName( this,
        tr("Ouvrir..."),
        QDir::homePath(),
        tr("Fichiers texte (*.txt);;Autres fichiers (*.*)") );

    if ( fileName.isEmpty() ) return ;

    QFile file( fileName );

    if ( ! file.open( QFile::ReadOnly | QFile::Text ) ) {
        QMessageBox::critical( this, tr("Erreur"),
            tr("Impossible de lire le fichier %1 :\n%2")
                .arg( fileName ).arg( file.errorString() ),
            tr("Fermer") );
        return ;
    }

    QTextStream in( &file );
    editor->setPlainText( in.readAll() );
    QMainWindow::statusBar()->showMessage( tr("Lecture ok !"), 2000 );
}
```

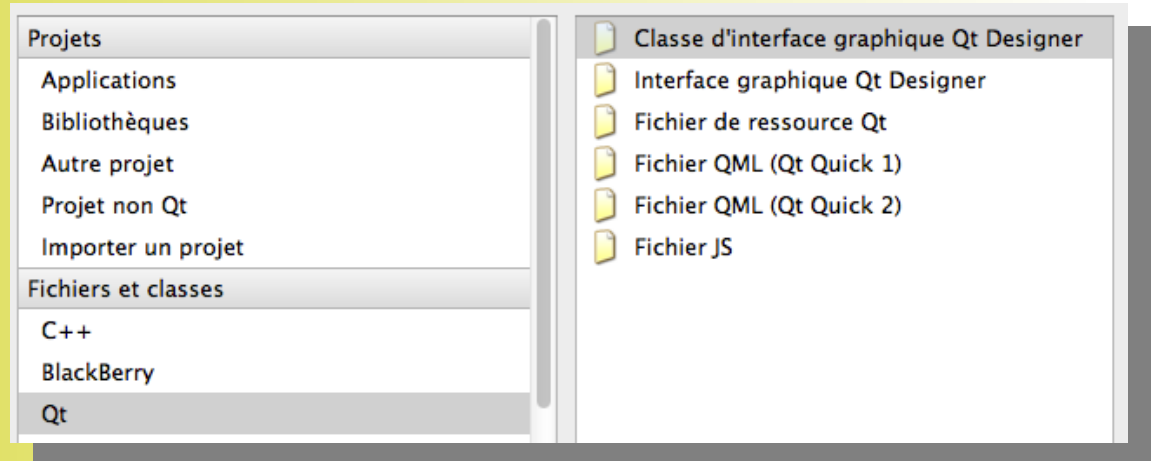
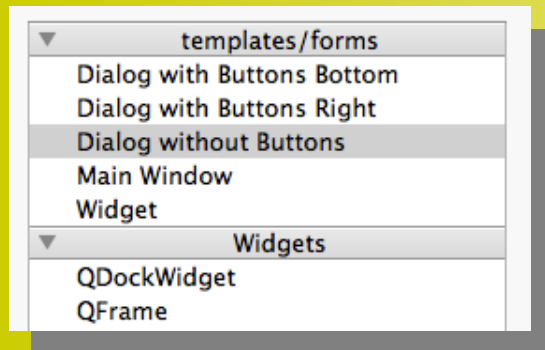
affichage
du texte

Fichier | Nouveau fichier ou projet...

📁 classe d'interface graphique

Modèle d'interface :

📁 Dialog without Buttons



l'assistant ajoute au projet les fichiers :

📁 search.h
 📁 search.cpp
 📁 search.ui

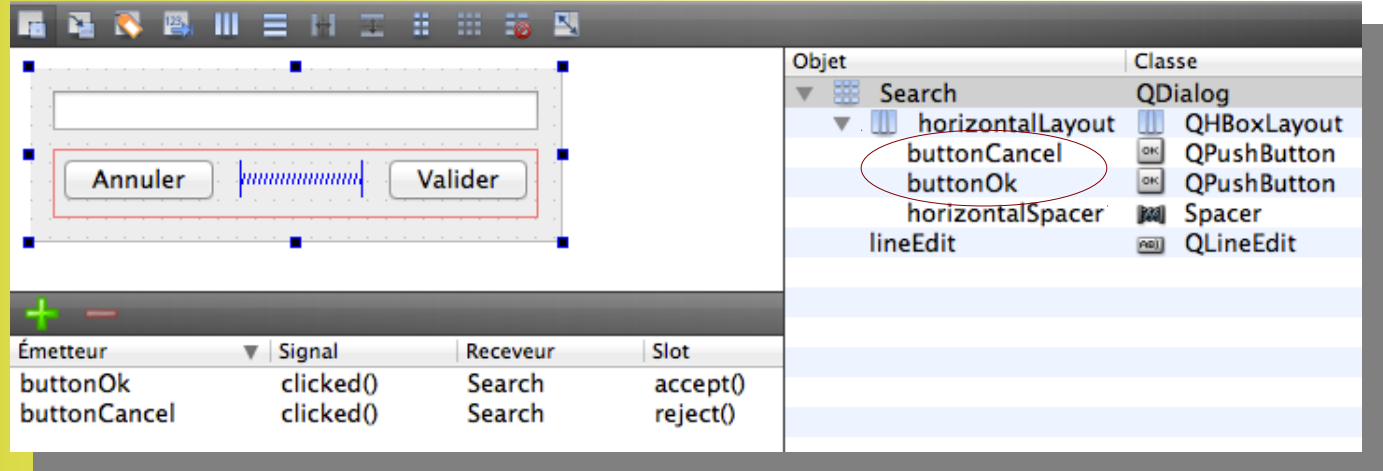
Nom de la classe : Search

Ajouter au projet : DemoEditor.pro

Construction de l'interface :

- placement et ...
- ... nommage des widgets
- connexions signal/slot

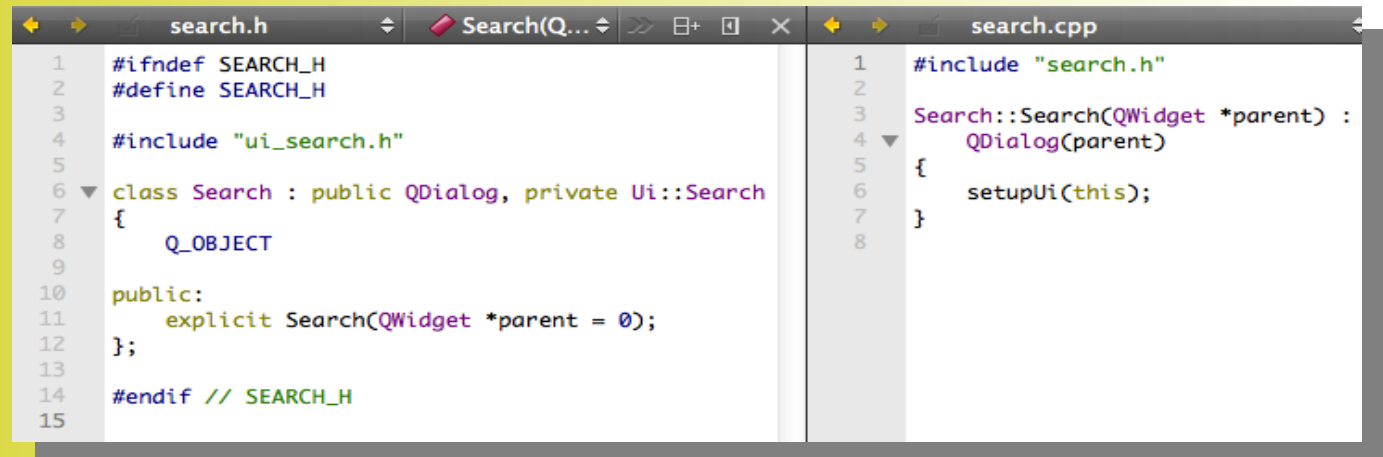
Classe associée
(héritage multiple) →



The image shows the Qt Creator interface. The top part is the UI Designer, displaying a dialog box with a text input field, an 'Annuler' (Cancel) button, and a 'Valider' (Validate) button. The bottom part is the Signal/Slot Connection Editor, showing a table of connections:

Émetteur	Signal	Receveur	Slot
buttonOk	clicked()	Search	accept()
buttonCancel	clicked()	Search	reject()

On the right, the 'Objet' (Object) and 'Classe' (Class) list shows the hierarchy: Search (QDialog) contains horizontalLayout (QHBoxLayout), which contains buttonCancel (QPushButton), buttonOk (QPushButton), horizontalSpacer (Spacer), and lineEdit (QLineEdit).



The image shows the source code for the Search dialog class. The left pane displays `search.h` and the right pane displays `search.cpp`.

```

1  #ifndef SEARCH_H
2  #define SEARCH_H
3
4  #include "ui_search.h"
5
6  class Search : public QDialog, private Ui::Search
7  {
8      Q_OBJECT
9
10 public:
11     explicit Search(QWidget *parent = 0);
12 };
13
14 #endif // SEARCH_H
15
  
```

```

1  #include "search.h"
2
3  Search::Search(QWidget *parent) :
4      QDialog(parent)
5  {
6      setupUi(this);
7  }
8
  
```

Enrichissement minimal de la classe :

search.cpp →

search.h

```
#include <QTextEdit>
#include <QTextCursor>
```

```
public:
    Search(QTextEdit* editor) ;
    bool select() ;

private :
    QTextEdit* editor ;
    QTextDocument::FindFlags flags ;
    QTextCursor* cursor ;
    QString currentSearch ;
```

editor : document de travail
 flags : mode de recherche de texte
 cursor : curseur courant dans le document
 currentSearch : motif recherché

```
Search::Search(QTextEdit* editor) :
    QDialog(0)
{
    setupUi(this) ;

    this->editor = editor ;
    cursor = new QTextCursor( editor->document() ) ;
    flags = 0 ;
}

bool Search::select()
{
    if ( ! exec() ) return false ; ← projection de la fenêtre fille

    currentSearch = lineEdit->text() ;

    flags = 0 ;
    flags |= QTextDocument::FindWholeWords ;
    flags |= QTextDocument::FindCaseSensitively ;

    *cursor = editor->textCursor() ;
    *cursor = editor->document()->find( currentSearch, *cursor, flags ) ;

    if ( !cursor->isNull() ) editor->setTextCursor( *cursor ) ;

    return true ;
}
```

La classe de la fenêtre principale doit maintenir un objet de classe Search :

↩ `mainwindow.h` : nouveau membre

```
private:  
    Search*    searchDlg ;
```

↩ `mainwindow.cpp` : constructeur

```
searchDlg = new Search( editor ) ;
```

L'action « Rechercher » est ajoutée dans le menu « Edition »...
Le slot `search()` associé invoque l'ouverture du dialogue enfant :

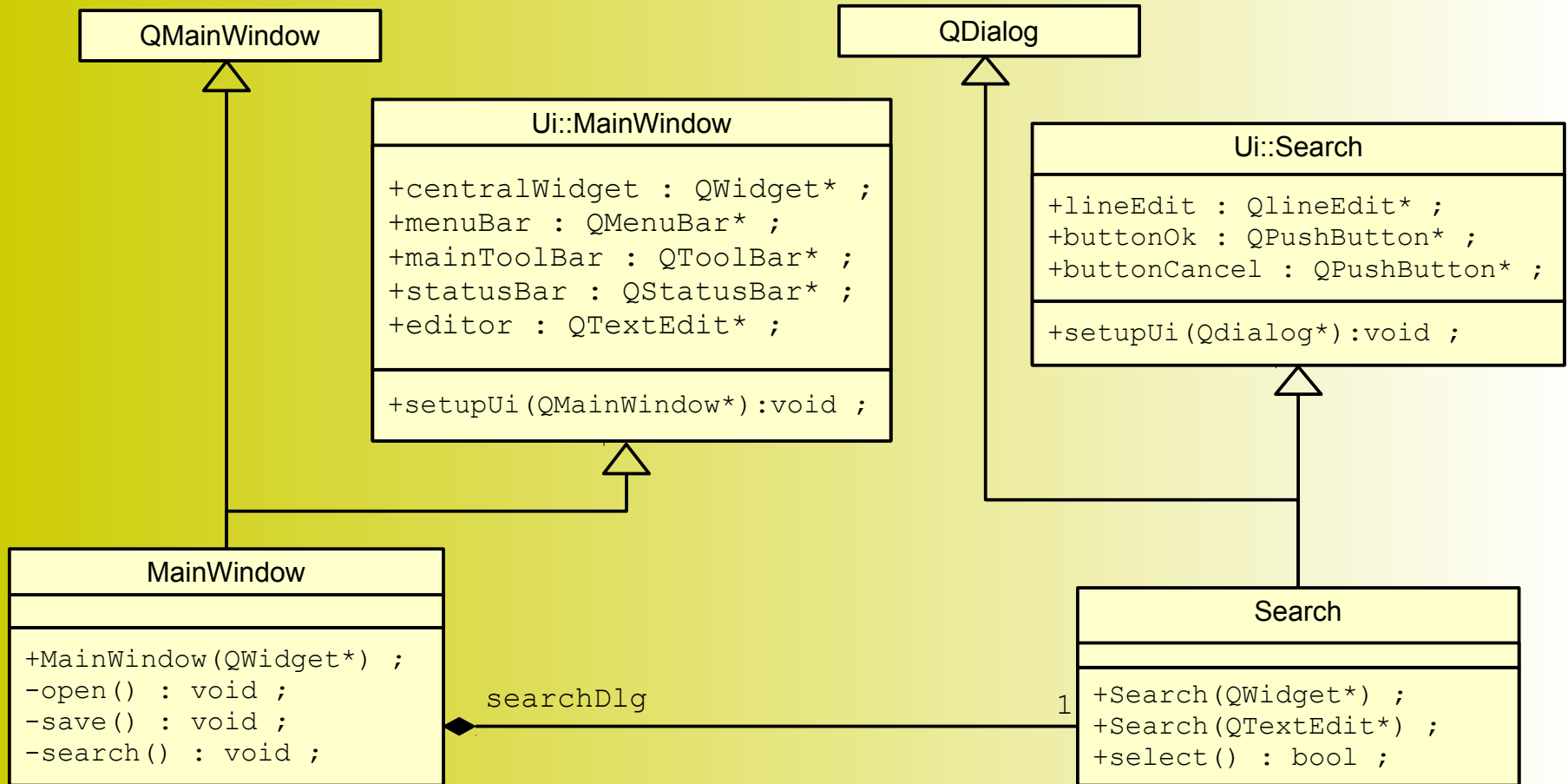
↩ `mainwindow.cpp` : méthode `search()`

```
void MainWindow::search()  
{  
    searchDlg->select() ;  
}
```

Test de compilation et d'exécution :

↩ la commande « Rechercher » ouvre maintenant la boîte de dialogue permettant d'indiquer un motif... En cas d'issue par le bouton « Valider », la première occurrence du motif dans le texte est montrée en surbrillance !

Diagramme de classes (dans l'état...) :



Étudier et tester les sources de l'archive

[DemoEditor-src-v3.tar.gz](#)

- ☞ barre d'outils, barre d'état
- ☞ affichage position curseur
- ☞ sélection de texte,
couper / copier / coller
- ☞ recherche / remplacement
d'occurrences dans le texte
- ☞ sélecteur de police de
caractères
- ☞ sélecteur de couleur
- ☞ mémoire inter-sessions
- ☞ icône d'application
- ☞ impression
- ☞

