

Laboratori IDI - Qt

Professors d'IDI, Q2-2014/15



# Índex

<b>1</b>	<b>Qt</b>	<b>7</b>
1.1	Introducció a Qt . . . . .	7
1.1.1	Components de la llibreria . . . . .	7
1.1.2	Primer programa en Qt . . . . .	8
1.1.3	Compilació . . . . .	10
1.1.4	Segon programa en Qt . . . . .	11
1.2	<i>Signals</i> i <i>slots</i> . . . . .	11
1.3	Interfícies més complexes . . . . .	15
1.4	Documentació . . . . .	17
1.5	Mini-tutorial fitxers <i>.pro</i> . . . . .	17
<b>2</b>	<b>QtDesigner</b>	<b>21</b>
2.1	Introducció . . . . .	21
2.2	Disseny de la nostra interfície . . . . .	24
2.2.1	Disseny visual . . . . .	24
2.2.2	Connexions . . . . .	25
2.2.3	Completar el disseny . . . . .	26
2.3	Compilant els fitxers creats amb el Designer . . . . .	28
2.4	Incorporant els nostres <i>widgets</i> . . . . .	29
2.4.1	Promoure components . . . . .	30
2.4.2	Crear plugins . . . . .	30
2.4.3	Creació del <i>custom widget</i> . . . . .	30
2.4.4	Creació del <i>plugin</i> . . . . .	31
2.4.5	Compilant i instalant un <i>plugin</i> . . . . .	34
2.5	Exercicis . . . . .	35
2.5.1	Colors senzills . . . . .	35
2.5.2	Colors . . . . .	36
2.5.3	Copiar i enganxar . . . . .	36
2.5.4	Botons i els seus atributs . . . . .	37
2.5.5	Més botons i atributs . . . . .	37

<b>3 QtCreator</b>	<b>39</b>
3.1 Introducció . . . . .	39
3.2 Utilitzant QtCreator . . . . .	39
3.3 Crear plugins en QtCreator . . . . .	43

# Introducció

La present documentació ha estat elaborada pels professors de l'assignatura d'Interacció i Disseny d'Interfícies.

Al llarg de l'assignatura aprendreu a dissenyar interfícies d'usuari i a implementar aplicacions basades en OpenGL. Aquest document està dedicat al disseny d'interfícies d'usuari mitjançant la llibreria de Qt, actualment en mans de Digia.

Els capítols que segueixen són un guió del que al laboratori es veurà sobre Qt i van acompanyats d'exercicis que cal resoldre. En particular, el primer capítol conté una sèrie d'exercicis de diferent grau de dificultat i diferents continguts. N'hi ha que no els resoldreu fins haver avançat més en la lectura d'aquest document, però els posem aquí com a llistat de problemes que hauríeu de ser capaços de solucionar en acabar amb aquest guió. Alguns d'ells han estat els enunciats de l'examen de Qt en quatrimestres anteriors.

Aquests guions cobreixen les sessions de Qt de laboratori de l'assignatura IDI.



# Capítol 1

## Qt

### 1.1 Introducció a Qt

En el laboratori d'aquesta assignatura haureu de construir interfícies d'usuari en C++ usant una llibreria estàndard per a aquest fi: Qt. Aquesta llibreria ha estat triada per ser suficientment general, i portable a molt diverses plataformes, però no és l'única possible. A classe de teoria es discutiran alternatives i altres filosofies de disseny per a aquesta mena de llibreries. Aquí ens centrem exclusivament en allò que necessiteu per a programar amb Qt les interfícies que necessitareu per al laboratori d'IDI.

#### 1.1.1 Components de la llibreria

Però en primer lloc, què és exactament una llibreria per a construir interfícies gràfiques d'usuari? En general aquestes llibreries ofereixen un seguit de components atòmics (alguns molt simples i altres força complexes) que es poden configurar i combinar per a construir altres interfícies. Vegem alguns exemples (Figura 1.1). En el primer cas, la Figura 1.1a ens mostra un botó. Un botó és un estri que serveix per a ser *clicat*, en aquest cas, ens interessarà programar com ha de respondre la nostra aplicació al *clic* sobre el botó. La Figura 1.1b ens mostra un *slider*. En aquest cas, ens interessarà determinar la resposta al desplaçament del cursor així com determinar sobre quins valors aquesta barra es mou. Els *radio buttons* que segueixen (Figura 1.1c) ens permeten seleccionar entre diverses opcions. Podeu veure dos exemples més de *widgets* a les figures 1.1d i 1.1e que representen una barra de *scroll* i una llista jeràrquica respectivament. Seguidament donarem més detalls dels mètodes en C++ que ens permeten gestionar les propietats més importants dels elements presentats.

El terme “widgets” prové de barrejar els termes anglesos per a finestra (*window*) i artilugi (*gadget*). Abans d'això, però, passarem a donar un cop d'ull a un programa que utilitza Qt per a crear una petita interfície.

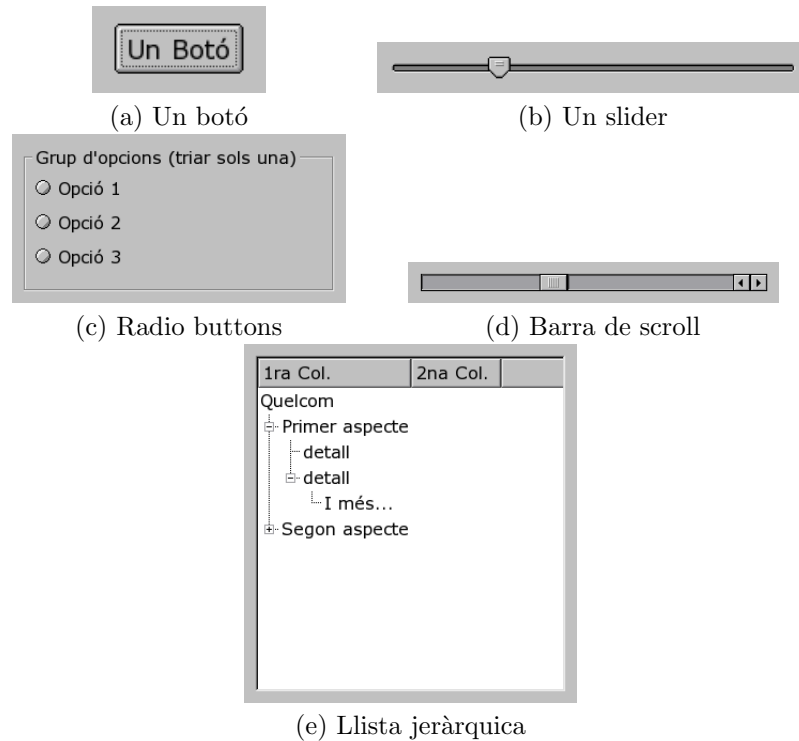


Figura 1.1: Alguns “widgets” de Qt

### 1.1.2 Primer programa en Qt

Vegem ara un exemple senzill (minúscul) d’aplicació Qt autocontinguda. Considereu el codi de la Figura 1.2.

En aquest programa tenim una petita aplicació que simplement conté un botó amb el text “Hello Qt!” que es mostrarà per pantalla. Les instruccions que permeten declarar tots els elements de l’aplicació estan contingudes en la funció principal (*main*). Però abans de passar a implementar aquesta funció hem d’incloure els fitxers que continguin declaracions d’elements que voldrem utilitzar més tard (bàsicament tipus i classes definits en llibreries o per nosaltres mateixos).

En aquesta primera sessió de Qt, utilitzarem la llibreria com una sèrie d’elements (*widgets* definits en classes) als quals podem accedir via la inclusió de fitxers que contenen la definició dels *widgets* disponibles. Per aquesta raó, haurèm de fer les inclusions necessàries per tal d’utilitzar-los al principi del nostre programa. Més endavant veurem que tot aquest procés es pot fer de forma més senzilla en el cas d’interfícies complicades mitjançant l’ús d’una eina per a l’ajuda al disseny d’interfícies: el *designer*. En el programa mencionat només s’utilitzen de forma directa dues classes de Qt: la *QApplication* i el *QPushButton*.



```

// Posem els fitxers de capçalera necessaris
// Permet declarar una QApplication
#include <QApplication>
// Permet declarar un QPushButton
#include <QPushButton>
// Programa principal
int main(int argc, char ** argv )
{
    // Aplicació en Qt que conté la meua interfície
    QApplication a(argc, argv);

    // Botó amb el text "Hello Qt!" que serà
    // la meua interfície
    QPushButton hello("Hello Qt!", 0);

    // Mètode per a determinar la mida del botó
    hello.resize( 150, 30 );

    // Mostro el botó
    hello.show();

    // El meu programa (main) executa el mètode que
    // posa en marxa l'aplicació "a" de Qt (exec) i
    // retorna el valor que generi en acabar
    return a.exec();
}

```

Figura 1.2: Un microexemple

ton. Per tal de poder-les utilitzar posem les dues línies inicials, tal com s'indica en els comentaris. És important fer notar que als laboratoris es fa servir un sistema de fitxers SAMBA que no distingeix entre majúscules i minúscules i això us podria ocasionar problemes de compilació (a casa, per exemple) si als includes no respecteu la mateixa sintaxi que en els noms de fitxers físics.

Per tal d'esbrinar quines capçaleres necessitem incloure, podem referir-nos al manual (que s'executa amb la comanda *assistant*, veure apartat 1.4) i buscar allà el *widget* necessari. El manual, a més d'informar-nos quin fitxer de capçalera conté les definicions que volem, també ens indica quins mètodes té disponibles la classe buscada, el seu comportament, de qui hereda, etc.

### 1.1.3 Compilació

Com ja sabeu, el procés de compilació en C++ es divideix en dos passos, la compilació en sí, que genera els fitxers en codi objecte, i el muntatge (o *link* en anglès) que genera l'executable. Pel primer pas és necessari que el compilador pugui accedir a totes les definicions de tipus i classes que utilitzem en el nostre programa. Supposant que el programa anterior l'hem guardat a un fitxer *ex1.cpp*, necessitem crear un fitxer *helloQt.pro* amb el següent contingut:

```
TEMPLATE=app
DEPENDPATH+=.
INCLUDEPATH+=.
#Input
SOURCES+=ex1.cpp
```

Una vegada creat aquest fitxer, per a compilar i crear un executable, utilitzeu les comandes:

```
qmake
make
```

es generarà un executable anomenat **helloQt** en el directori on estiguem. Aquest fitxer podrem executar-lo amb `./helloQt`.

#### Exercici 1.1.3.1

Comenteu la línia:

```
#include <QPushButton>
```

i intenteu compilar. Llegiu atentament els missatges d'error que us presenta el compilador. Això us permetrà familiaritzar-vos amb els errors que podeu tenir en compilar.

L'execució del programa ensenyarà doncs un únic botó com el de la Figura 1.3 Naturalment voldrem fer servir interfícies més complexes i amb un comporta-



Figura 1.3: El resultat d'executar el programa de la Figura 1.2

ment més complexe. Aquest serà el següent punt a tractar, però abans d'això farem un altre exemple.

### 1.1.4 Segon programa en Qt

En aquest apartat veurem un exemple una mica més complet. El codi de la Figura 1.4 crea una finestra amb dos botons i un quadre de text. Per a fer-ho utilitza un *widget* que serveix per a contenir i distribuir altres *widgets* (un contenidor), el *QHBoxLayout*, que, en aquest cas, agrupa els elements que conté de forma horitzontal.

Compileu i executeu el codi. Comproveu que els botons ara com ara no fan res, perquè no els hem definit cap comportament. El programa us hauria de crear un *widget* com el de la Figura 1.5.

#### Exercici 1.1.4.1

Experimenteu amb els paràmetres del constructor del *widget* molla (*spacer*). Aconseguiu que sigui més ample i més estret.

#### Exercici 1.1.4.2

Canvieu el codi de forma que tots els *widgets* apareguin un a sota de l'altre. Per a fer-ho caldrà que utilitzeu el contenidor que permet arranjar els objectes en posició horitzontal.

#### Exercici 1.1.4.3

Canvieu el codi de forma que els botons apareguin un a sota de l'altre però l'espai i el quadre d'entrada de text estiguin com en el codi original.

Ara ja hem vist com crear programets senzills que utilitzen *widgets* però fins ara no sabem com programar la resposta a les possibles accions que es poden realitzar amb aquests *widgets*. En la següent secció ens encarreguem d'explicar el mecanisme que utilitzarem.

Per tal de saber quin és el comportament de cada *widget* recordeu que només cal que consulteu el manual (*assistant*).

## 1.2 *Signals i slots.*

Evidentment, a part de tenir una àmplia col·lecció de components, cal tenir maneres fàcils de connectar entre sí i amb la nostra aplicació aquests components. En Qt existeixen dos mecanismes per a fer-ho, segons que es tracti de connexions “d'alt nivell” (corresponents a les abstraccions associades amb els components, com el prémer un botó o canviar el text a una casella d'un formulari) o de “baix nivell” corresponents a events bàsics associats a las parts del propi computador (com tecles premudes, moviments del ratolí, ...).

Seguidament tractarem els d'alt nivell. Qt fa servir un mecanisme especial per aquests. Els components Qt, i els que nosaltres desenvolupem per a comunicar-se amb elements Qt, implementen **signals** i **slots**. Els *slots* són

```
#include <QApplication>
#include <QPushButton>

// Per a poder utilitzar el contenidor QFrame
#include <QFrame>
// Per a poder utilitzar el 'layout'
#include <QLayout>
// Per a poder utilitzar QLineEdit
#include <QLineEdit>

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

    QString fontFamily = "Arial";
    a.setFont(fontFamily);

    // Crea un frame
    QFrame F(0, NULL);

    // Crea un contenidor horitzontal
    QHBoxLayout* cH = new QHBoxLayout(&F);

    // Afegeix una caixa de text
    QLineEdit* le = new QLineEdit(&F);
    cH->addWidget(le);

    // Afegeix un espai (horitzontal, vertical)
    QSpacerItem *sp = new QSpacerItem(100,20);
    cH->addItem(sp);

    // Afegeix un boto
    QPushButton* ok = new QPushButton("D'acord", &F);
    cH->addWidget(ok);

    // Afegeix un altre boto
    QPushButton* surt = new QPushButton("Surt", &F);
    cH->addWidget(surt);

    F.show();
    return a.exec();
}
```

Figura 1.4: Segon exemple

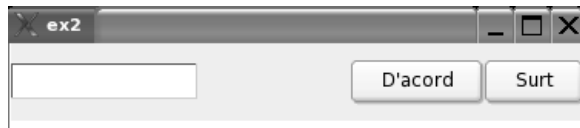


Figura 1.5: El resultat d'executar el programa de la Figura 1.4

mètodes especials d'una classe. Els implementem com qualsevol altre mètode, però el fet de declarar-los com *slots* fa que Qt els prepari per a ser connectats a *signals*.

Els *signals* també es declaren com un mètode, però no s'implementen. Allà on calgui, el codi inclou sentències `emit` per a “disparar un signal” (penseu-hi en una espècie de crida a funció, però el que es crida no és la implementació del *signal* sinó tots els *slots* que estiguin connectats (o subscrits) a aquest signal.

Vegem un exemple. Al directori `/assig/idi/Qt/S1-IntroQt` trobareu un fitxer `lab0.cpp`. Copieu-lo a un directori de treball vostre i mireu-lo amb l'editor. Veureu un seguit de declaracions dels components necessaris. En particular, la primera línia del `main()` diu:

```
QApplication app(argc, argv);
```

que declara un objecte “aplicació Qt” que conté alguns serveis comuns, i en particular el bucle de control de l'aplicació. Una aplicació Qt comença habitualment per declarar un objecte com aquest, i una sèrie d'altres recursos, i acaba entregant el control a aquest objecte aplicació amb una crida de la forma

```
app.exec();
```

de la que només tornarà quan es tanqui el *widget* principal o es cridi a `exit()`. En l'exemple l'aplicació simplement passa el retorn d'aquesta crida a qui l'ha cridat.

Però en el que cal que ens fixem ara és en dues línies cap al final del fitxer que tenen una crida `app.connect(...)`. Aquestes són les línies que indiquen quins *signals* volem connectar amb quins *slots* (i evidentment hi poden haver tantes com calgui). En aquest petit exemple, són:

```
app.connect(quitButton, SIGNAL(clicked()),
            w, SLOT(close()));
```

que connecta el *signal* `clicked()` del botó `quitButton` amb el *slot* `close()` del *widget* principal `w`. Quan l'usuari prem el botó (etiquetat “Quit”), el *widget* emet un *signal* `clicked()`, i aquest *signal* és alimentat als *slots* subscrits, en aquest cas el mètode `close()` del *widget* `w`. El resultat és que `w` es tancarà, i com l'havíem declarat com a *widget* principal de l'aplicació, aquesta plegarà.

L'altre `connect` que trobem diu:

```
app.connect(txtLine, SIGNAL(textChanged(const QString&)),
            label_mostra, SLOT(setText(const QString &)));
```

i per tant demana que cada canvi del text contingut al *widget* `txtLine` es propagui al slot `setText(const QString&)` del *widget* `label_mostra`. Per tant aquesta etiqueta, inicialment buida, anirà reflectint en tot moment el text que hi hagi a la línia de text `txtLine`.

Observeu que les signatures dels *slots* i *signals* connectats són compatibles (és a dir, `textChanged` i `setText` ambdós tenen com a paràmetre una `const QString&`). Aquest és un requeriment, i el principal avantatge d'aquesta arquitectura, ja que a diferència d'un sistema basat en *callbacks*, permet una verificació de tipus en temps de compilació alhora flexible i completa.

Ara podeu copiar-vos del directori `/assig/idi/Qt/S1-IntroQt` el fitxer `lab0.pro` i construir un executable amb les comandes

```
qmake
make
```

La comanda `qmake` llegeix una descripció abstracta del projecte, continguda en aquest cas en el fitxer `lab0.pro`, i crea un `Makefile` adequat. Tot i que en aquest cas el `Makefile` és molt senzill, veurem més endavant que l'ús del `qmake` ens pot estalviar bastanta feina. Aquesta comanda `qmake` sols cal executar-la la primera vegada, ja que després el propi `Makefile` creat es cuidarà de mantenir-se al dia: si feu modificacions a `lab0.pro`, la següent execució de `make` automàticament farà primer de tot un `qmake -o Makefile lab0.pro`.

Ara podeu executar el programa i comprovar que té el comportament descrit: si premeu el botó “Quit” acaba l'execució, i si modifiqueu el text al camp etiquetat “Entra el nou text:” automàticament l'etiqueta de la primera línia s'actualitza amb el nou text. L'aspecte de la finestra creada pel programa, amb un text introduït per un usuari, es mostra a la Figura 1.6. Observeu que el botó

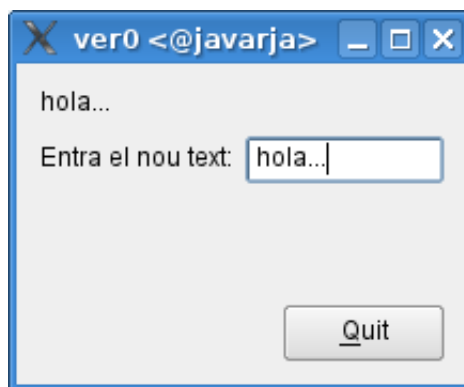


Figura 1.6: Exemple d'execució de `ver0`

té la ‘Q’ subratllada, suggerint que existeix un accelerador ‘`Alt+Q`’ per a aquest botó. Si premeu `Alt+Q` veureu que efectivament l'aplicació plega. L'únic que ha calgut per aconseguir aquest comportament ha estat precedir la ‘Q’ d'un ‘&’ a l'etiqueta del botó (vegeu la seva declaració).

**Exercici 1.2.1**

Mireu d'experimentar canviant l'ordre de crides i traient "spacers". Compareu el comportament en redimensionar la finestra.

**Exercici 1.2.2**

Genereu una aplicació que permeti inicialitzar l'hora i el minut d'un rellotge digital. La interfície ha de contenir 2 Labels, 2 LCD Numbers i 2 Dials, a més del botó de sortir. L'hora i el minut s'hauran d'especificar girant els dials i el corresponent valor apareixerà en els LCD. Recordeu que l'hora és un enter que va de 0 a 23 i el minut de 0 a 59. Cap dels elements de la interfície ha de desaparèixer quan es fa més petita la finestra. Els LCD Numbers i els dials han de expandir-se quan la finestra es fa més gran. (Veure figura 1.7).

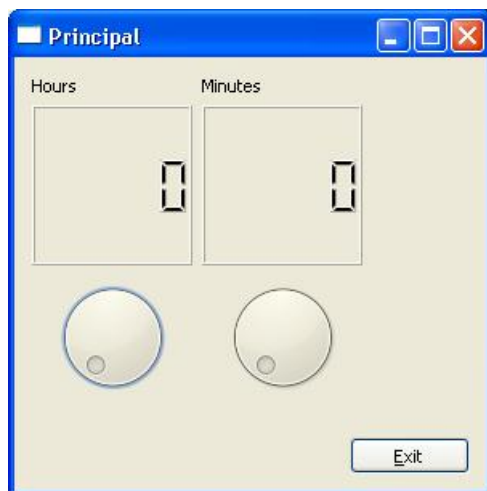


Figura 1.7: Exemple d'execució de l'exercici 1.2.2

## 1.3 Interfícies més complexes

Tots els components de Qt (i d'altres nostres que haguem construït per a comunicar-se amb components Qt d'aquesta manera) deriven d'una classe base (`QObject`). Aquesta classe base proporciona els mecanismes necessaris per a suportar el sistema de *signals* i *slots*. Tanmateix quan les nostres classes implementin *signals* i *slots* caldrà atendre a d'altres passes.

En general per construir interfícies més complexes i útils caldrà poder connectar *signals* i *slots* dels *widgets* de la llibreria amb codi nostre. Per a fer-ho dissenyarem classes que deriven de `QObject` (quan els objectes que modelen no es representen en pantalla) o de `QWidget` (quan volem dissenyar un nou *widget*

nostre), o d'algun altre component de la llibreria (quan volem especialitzar o modificar el comportament d'algun component ja existent).

En general escriurem coses de l'estil de:

```
class LaMevaClasse : public QObject {

    Q_OBJECT    // Macro de Qt

public:
    // Mètodes públics
    LaMevaClasse();
    int value() const { return val; }

public slots:
    // Slots
    void setValue( int );

signals:
    // Signals
    void valueChanged( int );

private:
    // Part privada de la classe
};
```

La macro `Q_OBJECT` s'ha d'invocar necessàriament al començament d'una classe que implementi *signals* o/i *slots*. Observeu també que l'anterior no s'ajusta completament a la sintaxi C++. Per exemple, “`public slots:`” no és C++ correcte. Aquest codi ha de ser preprocessat per `moc` (el *meta-object compiler*) per tal d'obtenir codi C++ que el nostre compilador pugui compilar. Si hem creat un fitxer `.pro`, `qmake` s'ocupa automàticament d'incloure la crida a `moc` al `Makefile`. Naturalment si la circumstància ho requereix, podem implementar `protected` o `private slots` de la mateixa forma. No oblidem que els *slots* són al cap i a la fi, mètodes de la meva classe. Els *signals* també ho són, però el `moc` ja els implementarà completament per nosaltres. Nosaltres només haurem d'indicar els llocs en la implementació de la classe en què cal llençar aquests *signals*, inserint sentències com per exemple

```
emit valueChanged(nouValor);
```

on òbviament `nouValor` és una variable o atribut el valor de la qual rebran com a paràmetre els *slots* que estiguin subscrits a aquest *signal* com a resultat de l'execució de l'`emit`.

**Important:** Si hi ha diversos *slots* connectats a aquest *signal*, tots seran cridats, però no podem saber en quin ordre.



**Exercici 1.3.1**

Modifiqueu el programa que hem fet a la secció 1.2 de forma que, amb la mateixa interfície, quan es premi una lletra, s'actualitzi l'etiqueta que conté el text posant en minúscules els caràcters senars i en majúscules els parells. Consulteu els mètodes de les classes `QLabel`, `QString` i `QChar`. La idea serà programar una nova classe `QLabel` que tingui un mètode similar a `setText` que realitzi la tasca demanada.

**Exercici 1.3.2**

Com a exercici, podeu ara construir un programa que ensenyi una interfície idèntica a l'anterior, però que no actualitzi l'etiqueta amb el text entrat fins que l'usuari no premi el `return`. La dificultat d'aquest exercici rau en que el *widget* `QLineEdit` no té cap *signal* que sigui enviat quan es prem `return` i que tingui un paràmetre de tipus `const QString&` que puguem alimentar directament al slot `setText(const QString&)` de `QLabel` com vàrem fer a l'exemple anterior. De fet la llista completa de *signals* emesos per `QLineEdit` és:

```
void textChanged ( const QString & );
void returnPressed ();
void lostFocus ();
void selectionChanged ();
```

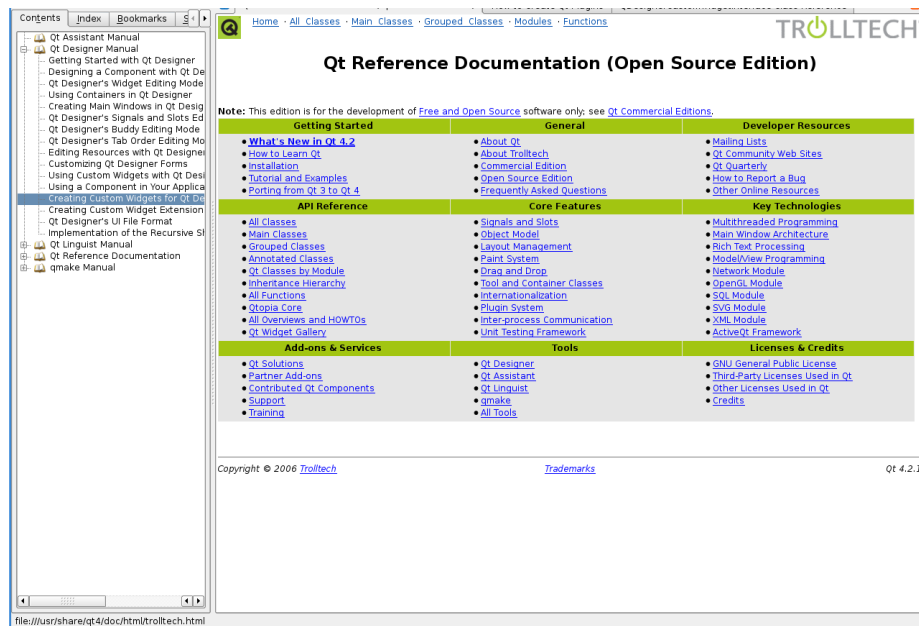
L'únic candidat vàlid és doncs `returnPressed()`. Ara bé, `QLineEdit` té un mètode `QString text()` que permet llegir el text que el camp del formulari conté en un moment donat. Amb aquests ingredients, podeu construir un nou *widget* que sí incorpori un *signal* que es llenci quan s'ha premut `return`, però que a més porti la informació del text que l'usuari hagi entrat fins el moment. Pots executar `/assig/idi/Qt/S1-IntroQt/ver1` per a provar el comportament que hauria de tenir la vostra solució.

## 1.4 Documentació

Naturalment en general us caldrà tenir informació detallada dels *widgets* i altres classes d'ajuda disponibles a la llibreria, i de tots els seus mètodes, *signals* i *slots*. Aquesta informació està disponible *on-line* a <http://doc.trolltech.com>. Sols heu de picar la comanda `assistant&`, i se us obrirà una finestra com la de la Figura 1.8. Des de les opcions de la columna de l'esquerra, podeu navegar per tota la informació que us pugui caldre, incloent tutorials d'ús, documentació detallada del `qmake`, i del propi `QtAssistant`.

## 1.5 Mini-tutorial fitxers *.pro*

Per a compilar els nostres fitxers utilitzem l'eina *qmake* que llegeix un fitxer de projecte i genera un *makefile*. En aquest apartat us expliquem breument el

Figura 1.8: La documentació *on-line* de Qt

format dels fitxers que llegeix el *qmake* i que porten extensió *.pro*. Tot això ho farem seguint un petit exemple.

Suposem que tenim els següents fitxers a compilar:

- hola.cpp
- hola.h
- main.cpp

El que hem de fer és configurar el fitxer de forma que sàpiga què ha de fer amb les fonts. Aquest nou fitxer l'anomenarem *hola.pro*.

Primer de tot posarem els fitxers font en l'apartat *SOURCES*, això es fa afegint una línia com la que segueix:

```
SOURCES += hola.cpp
```

Per a afegir més fitxers, afegim més línies com aquesta, al final tindrem:

```
SOURCES += hola.cpp
SOURCES += main.cpp
```

o equivalentment:

```
SOURCES += hola.cpp main.cpp
```

Un cop els fitxers d'implementació estan afegits, cal afegir les capçaleres. Això es fa mitjançant la variable *HEADERS* i de la mateixa forma, el fitxer hauria de quedar aproximadament:

```
HEADERS += hola.h
SOURCES += hola.cpp
SOURCES += main.cpp
```

Automàticament es genera un objectiu amb el nom del fitxer *.pro*, tot i que el podem fixar nosaltres amb la variable *TARGET*:

```
TARGET = holaMon
```

Una vegada fet això cal que establim els valors de la variable *CONFIG*. Com que tractarem amb una aplicació Qt, caldrà que tingui el valor 'qt' per tal que les llibreries adients es muntin amb l'executable.

L'aspecte final hauria de ser quelcom similar a:

```
TARGET = holaMon

CONFIG += qt

HEADERS += hola.h
SOURCES += hola.cpp
SOURCES += main.cpp
```

Una vegada fet això el fitxer *makefile* es pot generar amb:

```
qmake -o Makefile hola.pro
```

o bé fent només *qmake* si no tenim cap fitxer més de projecte en el directori. Quan el *makefile* s'ha generat correctament, podem compilar amb *make*.

Tot i que la sintaxi dels fitxers de projecte és molt rica i ens permet fer moltes coses, només mencionarem els perfils de configuració que podem necessitar de forma més habitual:

- *release*: L'aplicació es genera en mode d'execució normal, sense informació de *debug*.
- *debug*: L'aplicació es genera amb informació de *debug*.
- *warn\_on*: El compilador ha de mostrar tants avisos com sigui possible.
- *warn\_off*: El compilador ha de mostrar el mínim d'avisos possible.
- *qt*: El compilador ha d'incloure les llibreries de Qt per a compilar.
- *thread*: L'aplicació utilitza múltiples *threads*.
- *opengl*: El compilador afegeix les llibreries d'OpenGL o Mesa en compilar.

- *moc*: L'aplicació es genera utilitzant el *Meta Object Compiler (moc)* si cal.

La configuració també es pot fer traient algun perfil amb la comanda de restar `-=` en comptes de sumar `+=`. Això es pot fer servir per exemple per desactivar l'opció *moc* que per defecte està activada.

Més endavant dissenyarem interfícies d'usuari amb l'eina *designer*. Aquesta genera uns fitxers que especifiquen la interfície en format *xml* i que tenen un nom acabat en l'extensió *.ui*. Per a incloure aquests fitxers en el nostre programa i que els processi adequadament, caldrà una altra línia, amb el format:

```
FORMS += nomInterficie.ui
```

## Capítol 2

# QtDesigner

### 2.1 Introducció

En l'anterior capítol hem vist com construir interfícies senzilles amb Qt. Tot i que els mecanismes que heu après poden fer-se servir per a interfícies arbitràriament complexes, a vegades pot ser desitjable comptar amb eines que ajudin al disseny d'aquestes interfícies més abstractes.

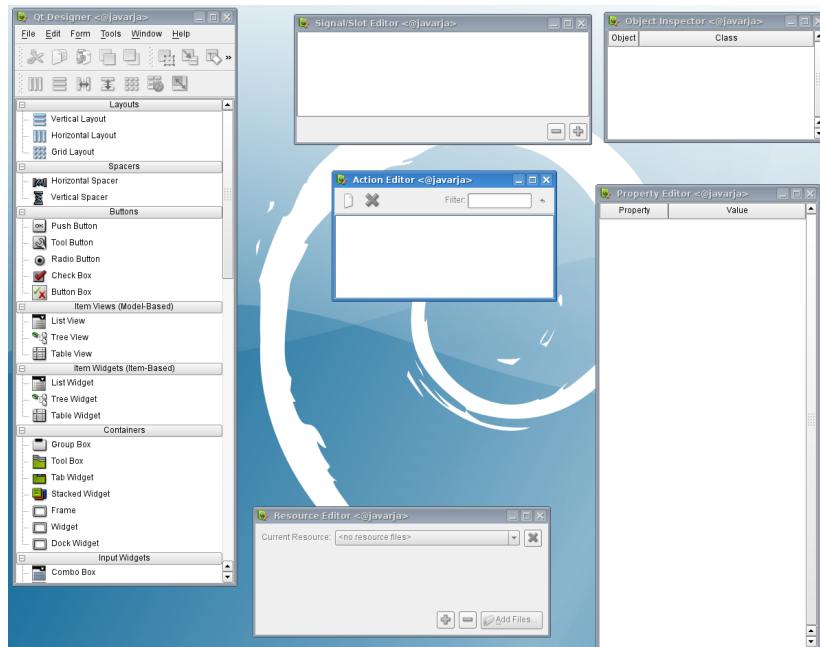


Figura 2.1: Finestra inicial del Qt Designer

Qt ens proporciona una eina de programació visual d'interfícies força potent, de la que disposeu en aquest laboratori: el *QtDesigner*. Per a executar-lo sols heu d'executar la comanda **designer**, que obrirà una interfície com la de la Figura 2.1.

S'obriran una sèrie de finestres amb diferents eines i menús.

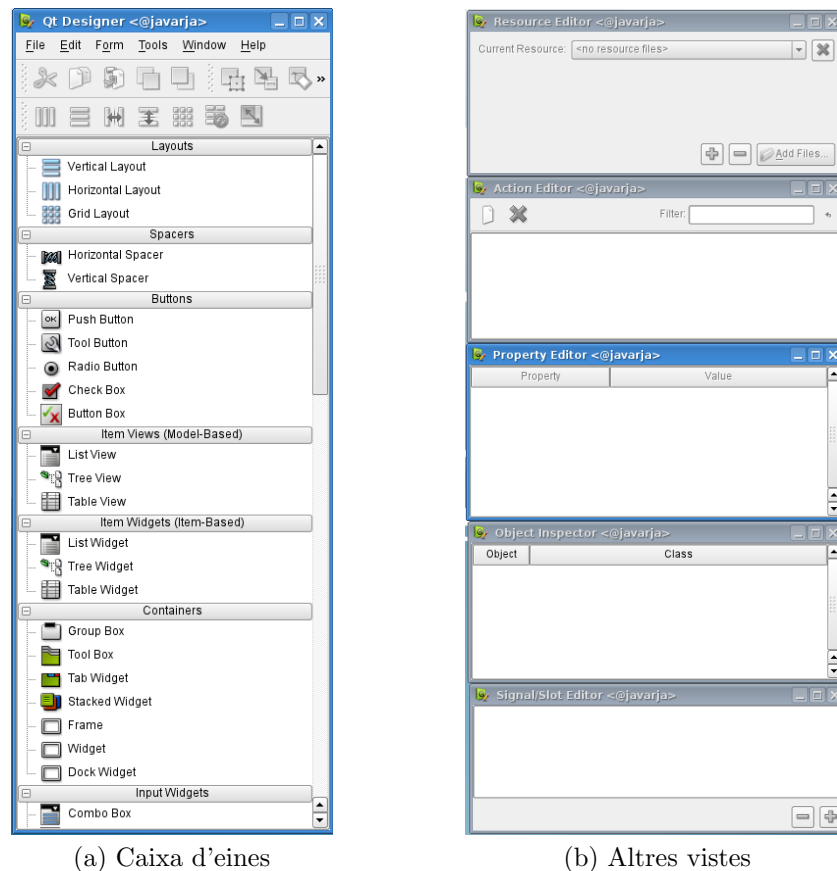


Figura 2.2:

Vegem primer de tot quines són aquestes vistes. Inicialment (si canvieu quelcom la següent vegada engegarà amb la configuració que hagueu deixat) hi ha a l'esquerra una finestra en vertical: és la caixa d'eines (*toolbox*) que serveix per a inserir diferents *widgets* a la interfície que construïu (veure Figura 2.2a). A la dreta (Figura 2.2b) hi ha cinc vistes independents. Es diuen *Resources editor*, *Action editor*, *Property editor*, *Object inspector* i *Signal/Slot editor*.

Abans de seguir, sapigheu que si tanqueu alguna d'aquestes vistes, ja no apareixerà quan sortiu i torneu a entrar a l'aplicació. Passarà el mateix si les moveu a una altra ubicació. El canvi serà recordat. Si accidentalment tanqueu alguna, podeu restaurar-la usant el menú **View** i tornant a seleccionar-la, però

fins que us hagueu familiaritzat amb QtDesigner, és potser recomanable que no les toqueu.

Si obriu un nou form **File->New Form**, s'obrirà una finestra com la de la Figura 2.3. Les diferents opcions que us proposa aquesta finestra corresponen

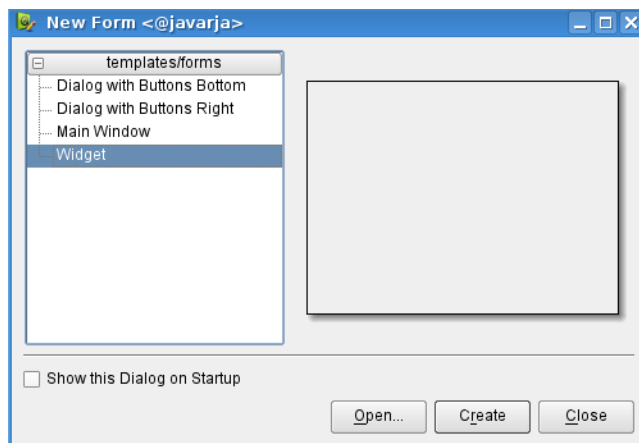


Figura 2.3: Iniciant un nou component

a diferents tipus de components que vulgueu crear. Per exemple *Main Window* us crearà la finestra principal d'una aplicació.

La resta d'opcions corresponen als diferents tipus de *widgets* i diàlegs que podeu construir. Seleccioneu per exemple *Widget*, i premeu *Create*. Al centre de la pantalla us apareixerà una finestra que representa el *widget* que esteu creant. Els puntets representen una graella útil per a dimensionar o alinear coses, tot i que en general és millor que les dimensions i posicions es determinin a través de *layouts*. Comproveu que podeu “agafar” amb el ratolí la finestra, les seves vores, o les seves cantonades, per a canviar la seva posició i tamany. A més veureu que els diferents editors s'han poblat amb informació sobre el vostre projecte. El fitxer creat per al form es diu “unnamed.ui” i és un fitxer `xml` que guarda la representació de la interfície que esteu dissenyant en un format adequat per a ser processat pel programa `uic` (sigles per *user interface compiler*), que generarà el codi C++ necessari. (Guardeu el fitxer `unnamed.ui` com a `Form.ui`).

En la vista de la finestra **Object inspector**, veiem en la pestanya d'objectes, que apareix un nou objecte: el nostre `QWidget Form`. Finalment, la finestra **Property editor** mostra les propietats de *Form*, les quals podem modificar en aquesta vista. Per exemple, seleccioneu “window title”, i entreu, en comptes del text que hi ha, “Exemple d'ús de Qt Designer”. Veureu que el títol de la finestra a l'àrea central canvia pel que heu introduït. Amb això ja hem encetat el disseny de la nostra interfície, per a fer-la útil caldrà afegir els elements necessaris tot pensant en l'aplicació que hagi de dur a terme. De la part de disseny se n'ocupa el següent punt.

## 2.2 Disseny de la nostra interfície

En aquest apartat començarem a afegir elements a la nostra interfície. Inicialment, només treballarem amb els *widgets* que vénen per defecte. Més endavant veurem com s'afegeixen nous *widgets* que ens haguem creat nosaltres.

### 2.2.1 Disseny visual

Comencem per usar les diferents eines de la vista de la “caixa ad'eines” per a instanciar diferents *widgets* com a fills d'aquest `QWidget`. Per exemple, seleccionem de la columna de l'esquerra la pestanya “Input Widgets”, i feu clic sobre “Horizontal Slider”. Veureu que apareix un slider com a cursor, indicant que l'eina ha estat seleccionada. Moveu el cursor (mantenint el botó esquerre pressionat) sobre la finestra del nou *widget* a l'àrea central, situeu-lo on vulgueu i deixeu anar el botó del ratolí. Quan deixeu anar el botó del ratolí, apareixerà una barra de lliscament on estava situat el cursor. La barra està envoltada per vuit punts blaus, que representen nanses per on poder-la agafar i retocar-ne la mida si s'escau. La presència d'aquests punts indica que el component es troba seleccionat. Si moveu el punter per sobre el *widget* que estem construint, ara no té aspecte de barra de lliscament, sinó que és l'habitual fletxa inclinada.

A la dreta, la vista d'objectes mostra un nou objecte fill de “Form”, dit “horizontalSlider” (aquests noms assignats automàticament per QtDesigner els podeu modificar via el camp *objectName* del *Property editor*. És bona pràctica donar-los-hi noms mnemònics de la seva funció a la interfície.) També veureu que el *Property editor* mostra ara atributs del “Horizontal Slider” i no del “Form”. El *Property editor* mostra els atributs de l'objecte seleccionat. Si seleccionem més d'un, mostra els atributs comuns a tots els seleccionats (i les modificacions que es facin afectaran a tots els seleccionats).

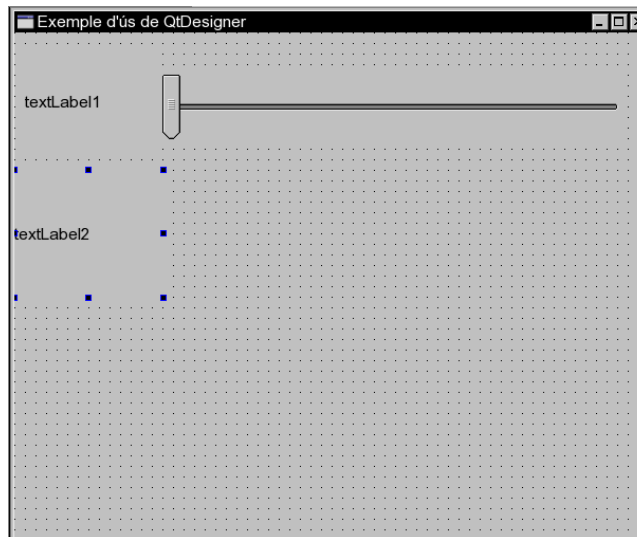
Podeu comprovar que si feu clic amb el botó esquerre del ratolí sobre el *slider*, el podeu arrossegat a una altra posició. Amb els vuit puntets blaus, en canvi, podeu canviar-ne les mides. Si feu clic fora del *slider*, el deseleccionareu, amb el qual el *Property editor* tornarà a ensenyar-vos propietats de MyForm. (Canvieu el *objectName* a MyForm). Comproveu que també podeu seleccionar el *slider* fent clic sobre la línia corresponent a l'*Object inspector*.

Anem a afegir ara un parell d'etiquetes de text. Seleccionem la pestanya “Display Widget” de la caixa d'eines. (tingueu en compte que en fer clic sobre una pestanya amb el símbol “+” a la esquerra l'obrim, però si té el símbol “-” la tanquem) Feu clic en “Label”, i arrossegueu-lo al *widget*. Creem dues a la banda esquerra del *widget*, amb el que la finestra de disseny hauria de quedar com a la Figura 2.4.

Ja estem acabant... Seleccionem ara, dins de la pestanya d'“Input Widgets”, el *widget* “SpinBox”, i afegiu un `QSpinBox` a la dreta de la segona etiqueta.

Feu clic en l'etiqueta superior, i al *Property editor* cerqueu la propietat “text”, i canvieu el valor per un rètol qualsevol, com “Control d'alçada”. Seleccionem la segona etiqueta i canvieu el seu text a un altre valor creïble com “Alçada en metres”.



Figura 2.4: El *widget* just després d'inserir les dues etiquetes

Ajusteu la mida de les etiquetes per a que es vegi tot el text que acabeu d'afegir.

### 2.2.2 Connexions

De moment hem fet una interfície que té una sèrie d'elements que no interaccionen entre ells, ara cal afegir-hi comportament. Com els dos components (el *Slider* i el *SpinBox*) controlen la mateixa quantitat, volem que romanguin sincronitzats. Seleccioneu l'eina d'editar connexions (🔗),

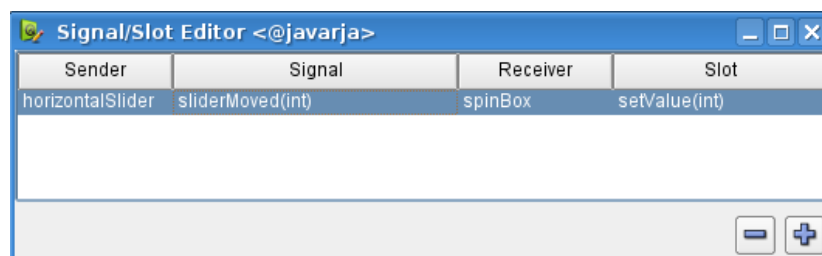


Figura 2.5: Editor de connexions


Passeu el cursor per damunt del `QSlider` i veureu com es pinta en vermell. Per a fer la connexió feu clic sobre el `QSlider` i arrossegueu el cursor sobre l'objecte amb el que volem fer la connexió. Veureu una línia que s'estén de l'objecte d'origen fins al cursor. Per a aquest exemple volem connectar-lo amb el `spinBox`. En posar el cursor sobre el `spinBox` i deixar anar el ratolí, aquest



apareixerà també en vermell i la línia l'apuntarà amb una fletxa. En aquest moment apareixerà també un diàleg com el de la Figura 2.5, que mostra els “signals” de l'objecte origen i els “slots” de l'objecte destí. Seleccionem el *signal sliderMoved(int)* i el *slot setValue(int)*. A continuació feu click en “Ok”. Podem veure com l'editor de *signals/slots* apareix aquesta connexió. També podem afegir connexions prement el botó amb el símbol “+” que apareix un a baix a la dreta i després fent doble click sobre <sender> apareix un menú. Seleccionem *spinBox*, feu doble click en <signals> i seleccionem *valueChanged(int)*. Al <receiver> poseu *horizontalSlider* i com a <slot> poseu *setValue(int)*.

Una prestació interessant del QtDesigner és que podem provar les interfícies que dissenyem sense passar per compilar i muntar el codi. Piqueu **Ctrl-R** per a veure la feina fins ara. Podreu comprovar que si modifiqueu el valor de l'alçada a algun dels dos *widgets*, canviarà també a l'altre. Piqueu **Ctrl-S** per a desar la feina

### 2.2.3 Completar el disseny

Fins ara hem dissenyat una interfície que es comporta com volíem, tanmateix el *widget in toto* apareix desordrejat. Els components no estan necessàriament ben alineats (a menys que hagueu estat molt i molt cuidadosos), i la qüestió empitjora si amb el ratolí canviem el tamany de la finestra. Proveu-ho. Comprovareu que el tamany de la finestra no afecta als *widgets* continguts, inclús si en part o completament queden fora d'ella. Per a completar aquesta interfície hem de proporcionar-li els corresponents *layouts*.

Per a tornar a editar el widget seleccionem l'eina d'editar Widgets (fent click en Edit del menú principal) o fent click sobre  Però primer afegirem un botó per tancar la interfície. Seleccionem “Push Button” de la pestanya etiquetada “Buttons”, i col·loquem a baix a la dreta un botó. En la propietat *text* poseu-hi “&Sortir”. El símbol ‘&’ davant la lletra ‘S’ indica que volem un accelerador per aquest botó que serà **Alt-S**. Amb l'editor de connexions afegiu una connexió del *signal clicked()* d'aquest botó amb el *slot close()* de *MyForm*.

Ara ja estem en condicions d'introduir els *layouts* i concloure l'exemple. Feu clic en la primera etiqueta per a seleccionar-la. A continuació premeu la tecla **Ctrl** i feu clic altre cop, ara al *QSlider*. Ara tots dos *widgets* estan seleccionats. Tots dos exhibeixen els vuit puntets al voltant. De la barra d'eines superior piqueu en el botó de *layout* horitzontal . Comprovareu que els dos *widgets* canvien instantàniament de mida. Feu el mateix amb la segona etiqueta i el *QSpinBox*. Podeu obtenir el mateix efecte amb l'accelerador **Ctrl-1**. Amb el botó “Sortir” no podem fer el mateix, perquè està sol. Però si no fem res s'estirarà fins tota l'amplada del *widget* que el conté, així que afegirem un espaiador (una mena de molla) prement el botó  de la barra de tasques i arrossegant el ratolí per a dibuixar un rectangle a l'esquerra del botó. Modifiqueu la mida de l'espaiador per a què l'aspecte de la vostra interfície sigui semblant a la Figura 2.6.

Col·loqueu l'espaiador i el botó en un *layout* horitzontal com abans. Abans d'acabar, i per a què el botó es trobi sempre a l'extrem inferior, ens caldrà una

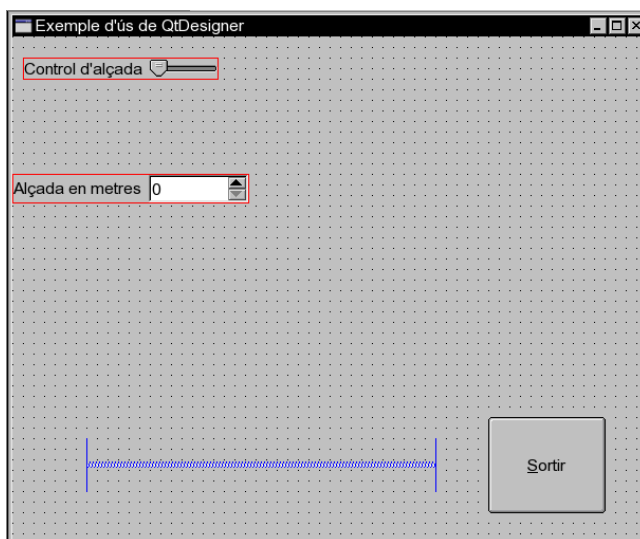


Figura 2.6: Estat després de definir una molla per a mantenir el botó a la dreta

altra molla que l'empenyi verticalment. Inserir una altra molla igual que abans, però aquest cop a l'espai vertical entre el `QSpinBox` i el botó. Per acabar, hem de donar el *layout* principal de `MyForm`. Feu clic en la finestra del `MyForm` per tal que cap *widget* estigui seleccionat i piqueu `Ctrl-2` (o equivalentment premeu `☰`). Amb el ratolí redimensioneu tota la finestra a un tamany que us sembli adequat i proporcionat, i piqueu `Ctrl-S` per a desar la feina. Com abans podeu comprovar el funcionament picant `Ctrl-R`. Veureu que aquest cop la finestra es redimensiona correctament. A més el botó “Sortir” (o l'accelerador `Alt-s` que vàrem demanar) surten l'efecte esperat. La Figura 2.7 mostra el resultat final d'aquest exemple.

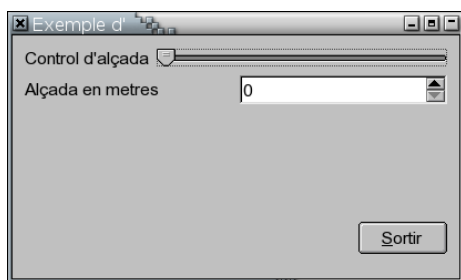


Figura 2.7: El resultat final

És molt important que dissenyeu les interfícies de forma adequada amb l'ús de *layouts* perquè sinó el seu aspecte pot variar de màquina a màquina (la resolució de la pantalla pot no ser la mateixa i fer que en conseqüència part

d'un *widget* no es vegi), cosa que dóna un to poc elegant i de vegades incòmode per treballar. Tingueu en compte que l'usuari el primer que veu de la vostra aplicació és la seva interfície, i si aquesta li resulta desagradable visualment o poc pràctica a l'hora de treballar-hi, difícilment tindrà una bona opinió del vostre programa.

## 2.3 Compilant els fitxers creats amb el Designer

Com ja hem explicat a l'apartat 2.1, el *form* que hem creat amb el *designer* anomenat `Form.ui` guarda la representació de la nostra interfície en format XML. Aquest fitxer serà processat pel programa `uic` que generarà el codi C++ necessari. Però per a poder compilar i executar el nostre *form*, necessitem crear amb un editor de textos els fitxers `MyForm.h` i `MyForm.cpp` com s'indica a la Figura 2.8.

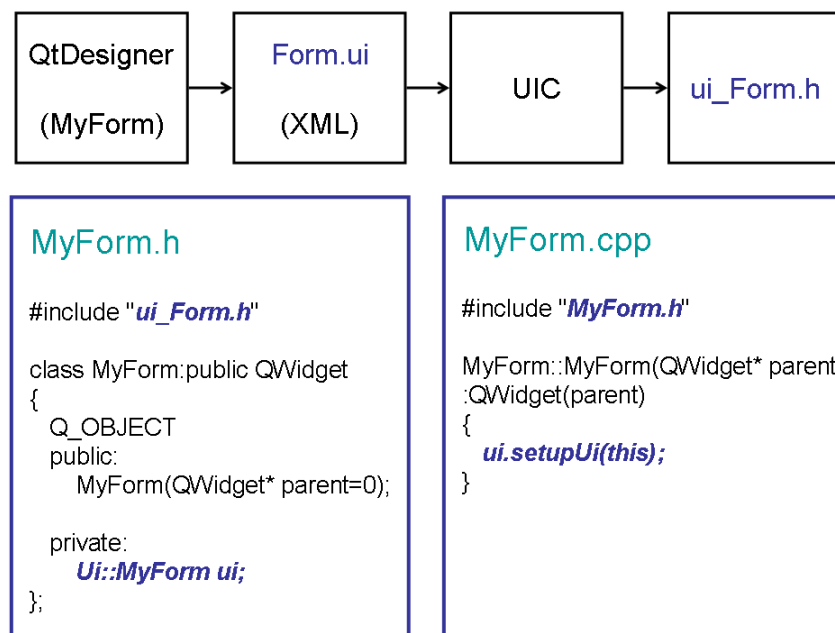


Figura 2.8: Components necessàries per a generar un *form* amb *designer*

Una vegada tenim aquests dos fitxers creats, necessitem crear el fitxer del projecte `myform.pro`:

```
TEMPLATE = app
TARGET =
DEPENDPATH += .
```

```

INCLUDEPATH += .

# Input
HEADERS += MyForm.h
FORMS += Form.ui
SOURCES += main.cpp MyForm.cpp

```

La variable `FORMS` li diu a `qmake` quins fitxers necessita processar amb `uic`. El fitxer `Form.ui` s'utilitza per a crear el fitxer `ui_Form.h`. Per a assegurar-nos que podem utilitzar el *form* que acabem de crear, necessitem incloure aquest fitxer de capçalera generat amb `uic` (`ui_Form.h`) abans de declarar l'atribut privat `Ui::MyForm ui` que cal incloure en la classe `MyForm` (vegeu la Figura 2.8). Aquest atribut `ui` ens proporciona el codi necessari per a utilitzar la interfície d'usuari que hem creat amb el *designer*. La classe `MyForm` té com a objecte privat `ui`, que ens proporciona el codi per a utilitzar la interfície d'usuari que hem creat amb el *designer*.

El constructor de la classe `MyForm` constueix i configura tots els widgets i layouts del *form* simplement fent una crida al mètode `setupUi()` de l'atribut `ui`.

Finalment el fitxer `main.cpp` contindrà el següent:

```

#include <QApplication>
#include "MyForm.h"

int main(int argc, char **argv)
{
    QApplication app(argc, argv);
    MyForm form;
    form.show();
    return app.exec();
}

```

Ara utilitzant les comandes `qmake` i `make`, podrem executar aquest exemple.

## 2.4 Incorporant els nostres *widgets*

Nogensmenys hem après al laboratori anterior que per a fer servir Qt sovint ens cal construir nous *widgets* especialitzant *widgets* de Qt o combinant-los per formar *widgets* més complexos. Evidentment, *QtDesigner* no pot saber d'aquests *widgets*. Per a poder fer servir *QtDesigner* amb aquests *widgets* que hem afegit, cal per tant donar-li informació. Això es pot fer de dues maneres: indicant a *QtDesigner* que aquest component és extern d'un que ell coneix (secció 2.4.1) o creant els nostres plugins per a *QtDesigner* (secció 2.4.2).

### 2.4.1 Promoure components

La manera més senzilla d'utilitzar un component nou (creat per nosaltres) des del *QtDesigner* és usar el **Promote**. El nou component (widget) hereta d'un widget propi de Qt, per tant, al *QtDesigner* afegirem un objecte d'aquest tipus. Per exemple, si hem creat un nou widget heredant del widget `QLabel`, afegirem al disseny un `QLabel`. Un cop el `QLabel` el tenim en el disseny, cliquem al damunt amb el botó dret del ratolí i escollim l'opció **Promote to...** Ens obre una nova finestra on ens demana el nom de la nova classe creada (`MyLabel`) i el fitxer `.h` que la defineix (`MyLabel.h`). Un cop escrits els paràmetres, cliquem al botó **Add** i després a **Promote**.

En aquest moment ja podem observar a la finestra de l'*Object Inspector* que el nostre nou widget (la label afegida) ja no és de tipus `QLabel` sinó que és de tipus `MyLabel`.

En aquesta opció de promoure components *QtDesigner* no pot conèixer com està implementada la nostra classe i per tant tota la informació de nous *signals* i *slots* que pugui tenir l'haurem d'introduir a mà al *QtDesigner*. Quan introduïm una nova connexió i se'ns obre la finestra *Configure Connection* amb els *signals* i *slots* dels components que volem connectar, haurem d'afegir (amb el botó **Edit...**) aquells *signals* i/o *slots* que la nostra nova classe (`MyLabel`) té. Quan cliquem a **Edit...** ens obre una nova finestra que ens permet afegir *signals* i *slots* de la nostra classe. Aquests *signals* i/o *slots* els hem d'afegir preservant el nom exacte i els paràmetres (només el tipus) de cada un. Si no ho introduïm correctament, ens fallarà a l'hora de compilar l'aplicació.

Aquesta és la forma més senzilla d'afegir components nous a les nostres aplicacions usant *QtDesigner*, i és al que usarem al laboratori, però requereix que li donem totes les dades entrades a mà i a més no podrem comprovar el seu funcionament amb el simulador del *QtDesigner* (amb **Ctrl-R**).

### 2.4.2 Crear plugins

Una altra manera de fer que *QtDesigner* sigui coneixedor de les classes que nosaltres hem creat és construint els nostres plugins i ficant-los en el directori on el *QtDesigner* els buscarà. El *QtDesigner* busca plugins al directori:

```
\$QT_PLUGIN_PATH/designer
```

Una vegada tinguem els plugins en aquest directori, podem obrir el *QtDesigner* i veurem que apareixen en la finestra de *Tools* els nostres *widgets*. Podrem utilitzar-los de la mateixa manera que hem fet amb la resta dels *widgets*.

### 2.4.3 Creació del *custom widget*

El *custom widget* es crearà de la mateixa manera que vàrem descriure a l'apartat 1.3, "*Interfícies més complexes*".

El fitxer de capçalera quedarà de la següent manera:

```
// Afegir aquest include a la llista de includes
#include <QtDesigner/QDesignerExportWidget>
```

```
class meuwidget : public QObject
{
    Q_OBJECT

public:
    // Constructor
    meuwidget (QWidget * parent=0);

public slots:
    // Afegir tots els slots del meuwidget

signals:
    // Afegir tots els signals
};
```

#### 2.4.4 Creació del *plugin*

Per a indicar-li al *QtDesigner* el tipus de widget que volem proporcionar, hem de crear una subclasse de *QDesignerCustomWidgetInterface* que conté la descripció de les propietats del *plugin*. La majoria d'aquestes funcions són virtuals en la classe base, ja que només té sentit que l'autor del *plugin* proporcioni aquesta informació.

A continuació mostrem el fitxer de capçalera del nostre *plugin* (*meucustomwidgetplugin.h*):

```
#include <QDesignerCustomWidgetInterface>

class meuplugin : public QObject, public QDesignerCustomWidgetInterface
{
    Q_OBJECT
    Q_INTERFACES(QDesignerCustomWidgetInterface)

public:
    meuplugin(QObject *parent = 0);

    bool isContainer() const;
    bool isInitialized() const;
    QIcon icon() const;
    QString domXml() const;
    QString group() const;
```

```

QString includeFile() const;
QString name() const;
QString toolTip() const;
QString whatsThis() const;
QWidget *createWidget(QWidget *parent);
void initialize(QDesignerFormEditorInterface *core);

private:
    bool initialized;
};

```

Finalment hem d'implementar el nostre plugin (*meucustomwidgetplugin.cpp*) com s'indica al següent exemple:

```

#include "meuwidget.h"
#include "meucustomwidgetplugin.h"
#include <QtPlugin>

meuplugin::meuplugin(QObject *parent)
    : QObject(parent)
{
    initialized = false;
}

void meuplugin::initialize(QDesignerFormEditorInterface * core)
{
    if (initialized)
        return;
    initialized = true;
}

// Torna true si el widget ha estat incialitzat, en cas contrari torna false.
bool meuplugin::isInitialized() const
{
    return initialized;
}

// Un punter a QWidget per a una instància del custom widget, construït amb
// el parent proporcionat.
QWidget *meuplugin::createWidget(QWidget *parent)
{
    return new meuwidget(parent);
}

// Nom de la classe que proporciona el nostre widget

```



```

QString meuplugin::name() const
{
    return "meuwidget";
}

// El grup dins de QtDesigner on volem que aparegui el nostre plugin
QString meuplugin::group() const
{
    return "Els meus widgets [Examples]";
}

// Podem utilitzar la icona que volguem per al nostre widget dins de la
// caixa del widget al QtDesigner.
QIcon meuplugin::icon() const
{
    return QIcon();
}

// Breu descripció per ajudar a l'usuari a identificar el widget dins de
// QtDesigner.
QString meuplugin::toolTip() const
{
    return "";
}

// Descripció més llarga per als usuaris del QtDesigner.
QString meuplugin::whatsThis() const
{
    return "";
}

// Serà true si el widget pot ser utilitzat per a contenir widgets fills,
// sinó serà false
bool meuplugin::isContainer() const
{
    return false;
}

// Descripció de les propietats del widget, com ara el seu nom, mida, i altres
// propietats estàndards de QWidget.
QString meuplugin::domXml() const
{
    return "<widget class=\"meuwidget\" name=\"atribut_botosemafor\">\n"
        "  <property name=\"geometry\">\n"
        "    <rect>\n"
        "      <x>0</x>\n"

```

```

    " <y>0</y>\n"
    " <width>100</width>\n"
    " <height>100</height>\n"
    " </rect>\n"
    " </property>\n"
    " <property name=\"toolTip\" >\n"
    " <string>The current color</string>\n"
    " </property>\n"
    " <property name=\"what'sThis\" >\n"
    " <string>The boto semafor widget displays "
    "the current time.</string>\n"
    " </property>\n"
    " </widget>\n";
}

// Fitxers que cal incloure en l'aplicació que utilitza el nostre widget.
// Aquesta informació es guardarà en els fitxers .ui i serà utilitzada per uic
// per a crear els #includes necessaris en el codi que genera per al form que
// conté el nostre widget
QString meuplugin::includeFile() const
{
    return "meuwidget.h";
}

Q_EXPORT_PLUGIN2(meucustomwidgetplugin, meuplugin)

```

La macro `Q_EXPORT_PLUGIN2()` serveix per a exportar la classe del nostre *plugin* de manera que pugui ser carregat des de *QtDesigner*.

### 2.4.5 Compilant i instal·lant un *plugin*

El fitxer de projecte del *plugin* ha d'especificar els fitxers `.h` i `.cpp` per al *custom widget* i per a la interfície del *plugin*. Normalment el fitxer del projecte només necessita especificar que es compili com a llibreria, però amb suport específic per a *QtDesigner*. Això es pot fer amb les següents declaracions:

```

CONFIG += designer plugin release
TEMPLATE = lib

# Input
HEADERS += meuwidget.h meucustomwidgetplugin.h
SOURCES += meuwidget.cpp meucustomwidgetplugin.cpp

```

És necessari assegurar-se que el *plugin* s'instal·la amb la resta dels *custom widgets* del *QtDesigner*.

```
target.path = ~/.designer/plugins/designer
```

```
INSTALLS += target
```

A l'*assistant* podeu trobar documentació més amplia sobre *QtDesigner*, incloent exemples més extensos i detallats. Però el que s'ha dit aquí hauria d'ésser suficient per ara. Per a posar a prova els vostres coneixements, proveu de repetir els tres exemples de l'anterior sessió de laboratori, però des de *QtDesigner*. Els dos primers exercicis els hauríeu de poder fer íntegrament en *QtDesigner*, i provar-lo exclusivament amb l'eina de previsualització. El tercer, evidentment, caldrà que el compileu per a provar-lo, tal com es discuteix més amunt.

Ara disposeu de dues maneres de construir interfícies gràfiques. Programant-les directament com a la sessió anterior, o via el *QtDesigner*.

Podeu començar provant un exercici senzill, sense *custom widgets*, com el 1.2.2, i després continuar amb els exercicis 1.3.1, 1.3.2 i els que teniu a l'apartat següent.

## 2.5 Exercicis

Els següents exercicis són per a que practiqueu en la creació de nous widgets per a resoldre problemes concrets. Podeu resoldre'ls usant **Promote** si heu de crear noves components i voleu usar *QtDesigner* per al disseny.

### 2.5.1 Colors senzills

Dissenyeu una interfície que contingui un text que tingui un fons que canviï de color de la següent forma. A la interfície hi ha d'haver dos botons, un que permeti posar el text amb fons blau i un altre que permeti posar el text amb fons vermell. Per a canviar el color d'un *widget* podeu mirar-vos la funció *setStyleSheet*. La interfície hauria de quedar aproximadament com a la Figura 2.9.

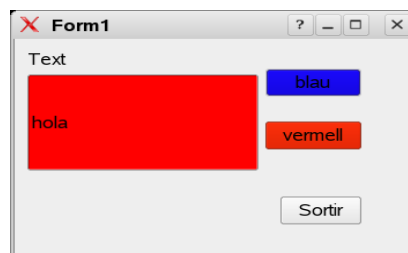


Figura 2.9: Formatejar un text

Ara proveu el mateix però canviant el color del text, en comptes del fons.

### 2.5.2 Colors

Implementeu una interfície que permeti experimentar amb colors en format RGB en concret es pretén poder veure quin color em genera una combinació RGB determinada. Per a això heu d'implementar una interfície amb un botó que es pintarà del color seleccionat, tres *sliders* que permeten canviar el color, i una etiqueta per cadascun dels *sliders* que em mostrin el color concret. En particular volem que els *sliders* puguin definir colors entre 0 i 1 amb precisió de 0.01. Per a canviar el color d'un botó podeu mirar-vos el mètode *setStyleSheet*.

**Nota:** Tingueu en compte que els *sliders* contenen només enters i que el format RGB en la classe *QColor* és en enters de 0 al 255.

El resultat hauria de ser aproximadament el que es mostra en la Figura 2.10.

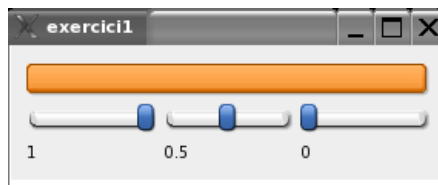


Figura 2.10: Exercici dels colors

### 2.5.3 Copiar i enganxar

Dissenyau una interfície que contingui un text editable i una etiqueta. A més, ha de tenir dos botons que faran la funció de copiar i enganxar. En el primer cas, quan es prem el botó de copiar, cal capturar el text que en aquell moment estigui al text editable. Un cop es prem el botó d'enganxar, l'etiqueta ha de rebre com a text el que s'ha copiat anteriorment, encara que el text editable hagi canviat el seu valor.

El resultat hauria de tenir un aspecte similar al de la Figura 2.11.

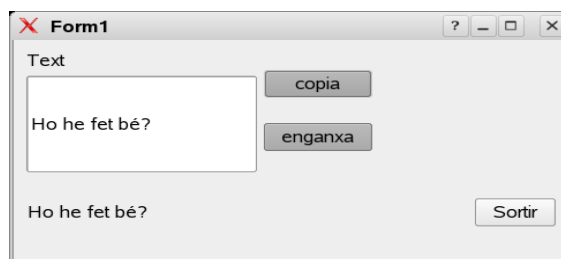


Figura 2.11: Copiar i enganxar

### 2.5.4 Botons i els seus atributs

En aquest exercici es demana que dissenyeu una interfície com la de la Figura 2.12(a) en la qual hi hagi dos botons que tenen un nom que pot ser canviat a través de les caixes de text que hi ha a la dreta. Cada vegada que el text de la caixa de text de la dreta es modifica, l'etiqueta del botó ha de canviar-se convenientment. A més, si es fa clic en algun dels botons, s'ha d'escriure “Sóc en” seguit de l'etiqueta que té el botó en aquell moment. Per a fer-ho, consulteu a l'assistant els mètodes de les classes *QPushButton* i *QLineEdit* i deriveu-ne alguna si cal. Noteu que la interfície ha d'estar construïda amb els *layouts* de forma correcta. Així, tant si es redimensiona en amplada com en alçada els elements s'organitzen adequadament (veieu la Figura 2.12(b)).

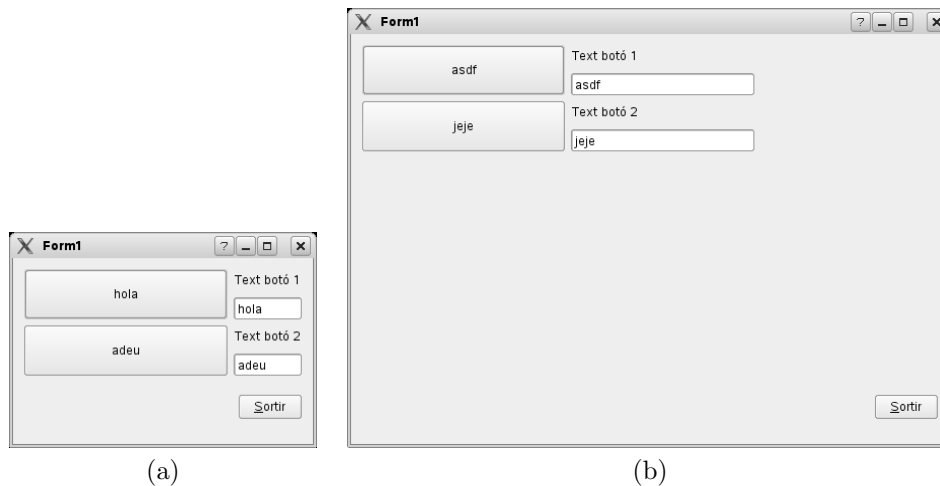


Figura 2.12: Modificació de les etiquetes dels botons

### 2.5.5 Més botons i atributs

Modifiqueu el programa anterior de forma que el botó només prengui com a nom el contingut de l'etiqueta quan es premi la tecla *return*, no mentre s'està editant.



## Capítol 3

# QtCreator

### 3.1 Introducció

*QtCreator* és una eina de desenvolupament de projectes creats amb Qt que permet la generació d'interfícies, l'edició, la compilació i la depuració de codi escrit en C++.

Es tracta d'un IDE (Integrated Development Environment) que facilita la creació de projectes més o menys complexos que fan servir la llibreria Qt. A més integra altres eines com *QtDesigner* per al disseny d'interfícies i automatitza la generació de codi per a crear *custom widgets* i *plugins*.

### 3.2 Utilitzant QtCreator

El funcionament del *QtCreator* és senzill: L'iniciem des d'una consola executant **qtcreator** i ens apareix la pantalla que permet crear un projecte nou o recuperar un que havíem tancat recentment (veure figura 3.1).

La següent pantalla ens permet seleccionar el tipus de projecte que volem crear (en el nostre cas escollirem “Qt Gui Application”) (veure figura 3.2).

A continuació introduïrem el nom i el directori del projecte que volem crear (figura 3.3).

El tipus d'aplicació que hem escollit conté un formulari per a la interfície. El formulari és una classe que deriva de **QWidget** i això ho hem d'indicar a la següent pantalla (figura 3.4). Automàticament el *QtCreator* genera el codi bàsic necessari per a la classe juntament amb el fitxer **.ui** que conté la definició en XML del formulari. En la pantalla següent (figura 3.5) es llisten els fitxers que el *QtCreator* genera de forma automàtica.

Un cop el projecte està creat, s'obre directament el designer per a crear la nostra interfície (figura 3.6). El designer està integrat dins del creator i funciona exactament igual que si l'executéssim des de consola. El fitxer **.ui** es va actualitzant automàticament.

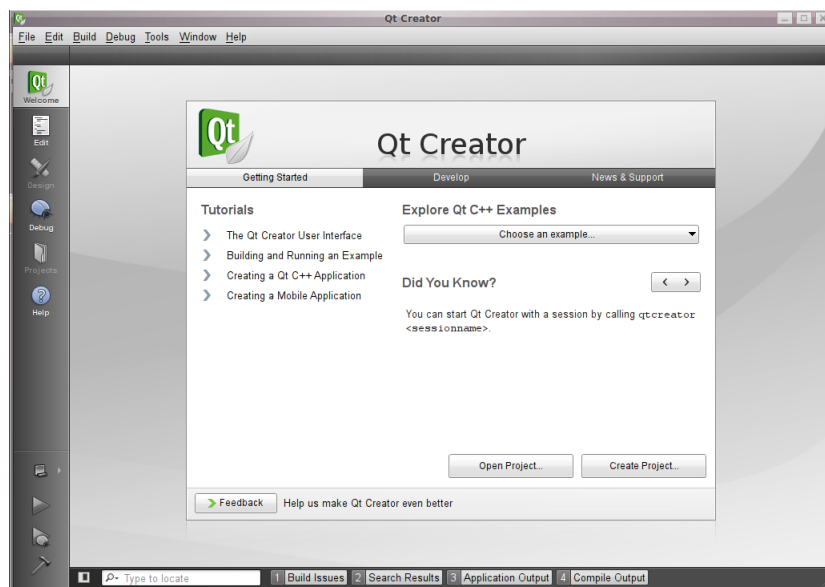


Figura 3.1: Finestra inicial de qtcreator.

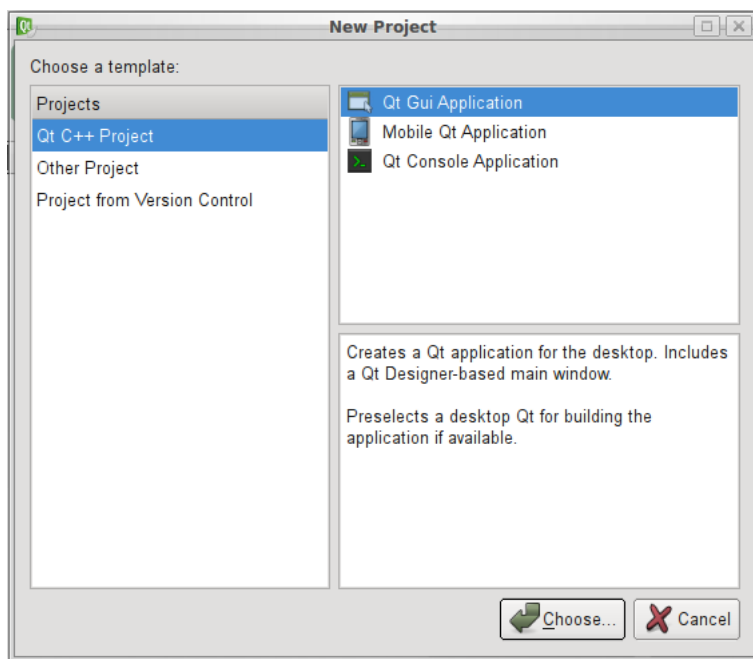


Figura 3.2: Creem un projecte d'aplicació Qt.



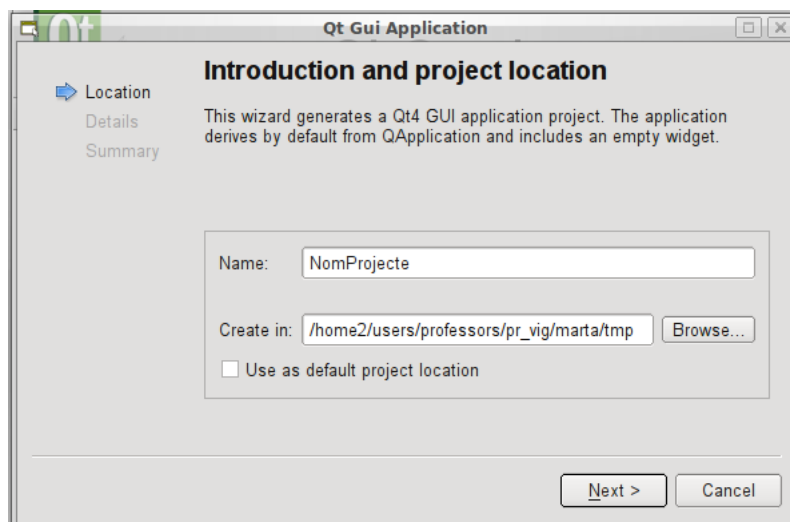


Figura 3.3: Dades del projecte.

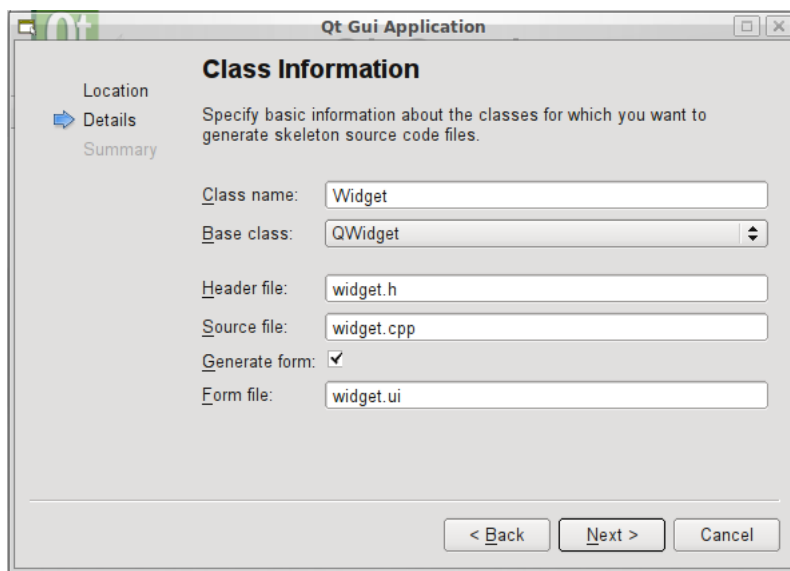


Figura 3.4: Informació de la classe principal.

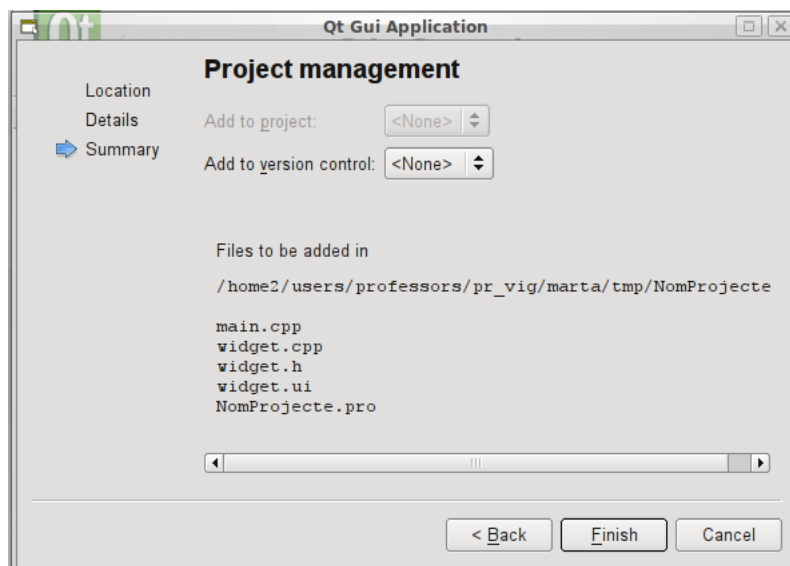


Figura 3.5: Fitxers generats automàticament.

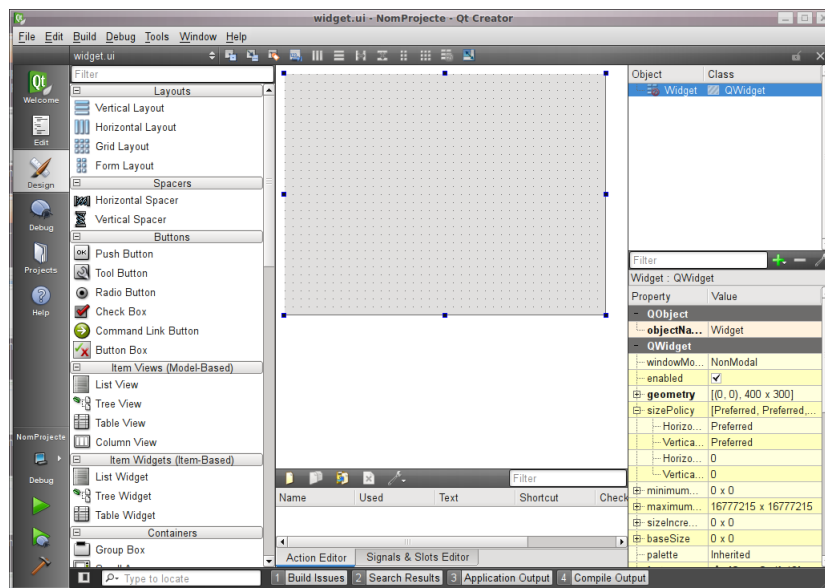


Figura 3.6: Vista amb el designer obert.

Fent doble clic sobre el nom del formulari (fitxer `.ui`) també es posa en marxa el designer en qualsevol moment si no estava ja obert, si estava obert d'abans, hi ha prou amb anar a la icona de diseny (a l'esquerra de tot de la finestra del `qtcreator`).

En tot moment `qtcreator` mostra a la seva esquerra i en columna les icones amb les eines bàsiques: l'edició, el diseny, les finestres de depuració, la gestió de projectes i l'ajuda.

Seleccionant la icona d'edició ens mostra la pantalla d'edició, en que ens trobem tots els components del nostre projecte. Fent doble clic sobre el nom del projecte (fitxer `.pro`) se'ns obre el fitxer per a poder visualitzar-lo i editar-lo (figura 3.7).

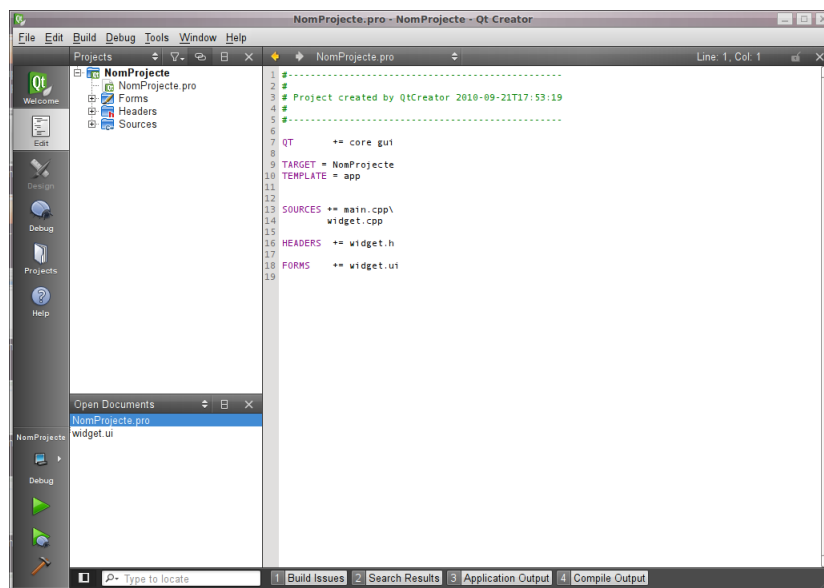


Figura 3.7: Vista del fitxer `.pro` en edició.

Finalment, podem compilar la nostra aplicació prement `<Ctrl>B` i executar-lo amb `<Ctrl>R`. Si volem veure la sortida de la compilació hem d'obrir "Compile Output" dels passos mostrats a la part d'abaix del `qtcreator` (veure figura 3.8).

### 3.3 Crear plugins en QtCreator

Amb *QtCreator* també és possible crear *custom widgets* de forma simple. A més es genera també de forma automàtica el codi necessari d'un *plugin* que permet treballar amb el *custom widget* des de *QtDesigner*.

Si en l'ús del `qtcreator` volem generar *custom widgets*, hem d'engegar el `qtcreator` després d'haver inicialitzat, en la mateixa consola on es crida al

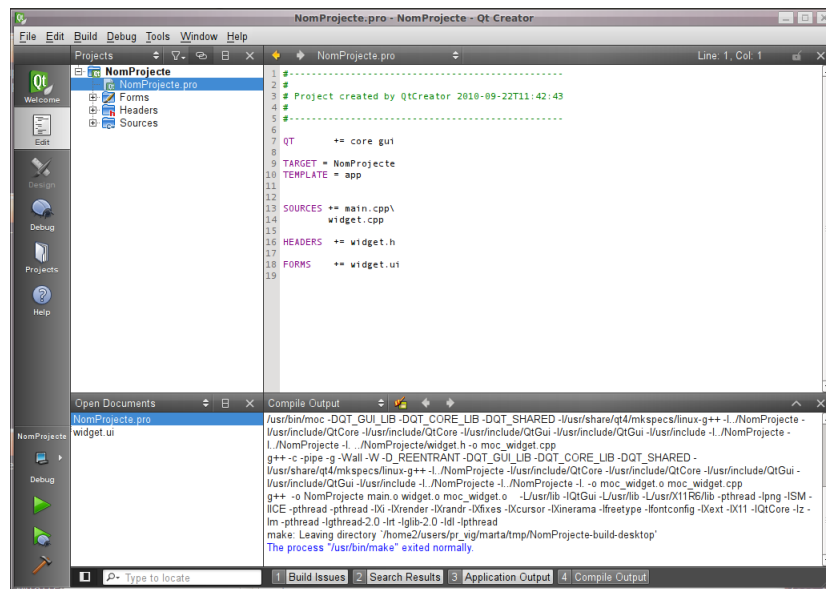


Figura 3.8: Resultat de la compilació.

programa, la variable d'entorn `QT_PLUGIN_PATH` posant-li el path on el designer anirà a buscar els *plugins* (per exemple: `$HOME/.designer/plugins`).

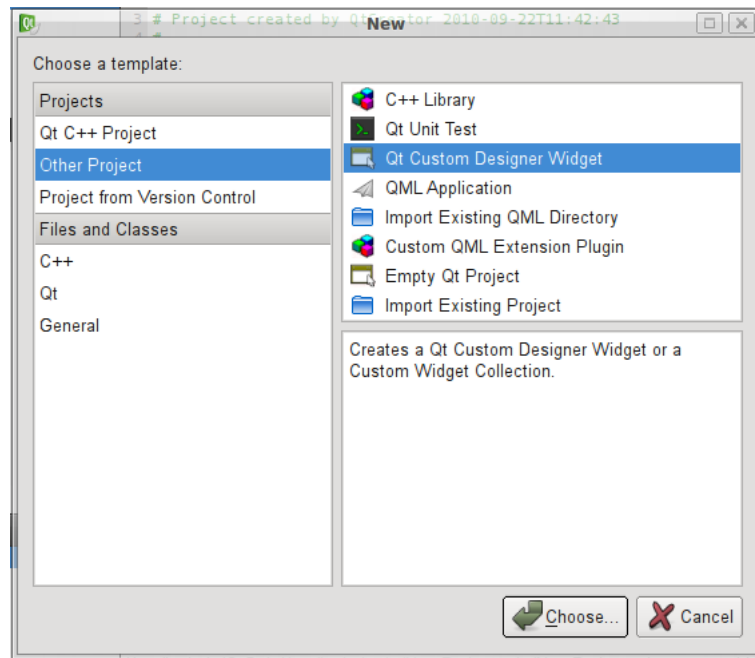
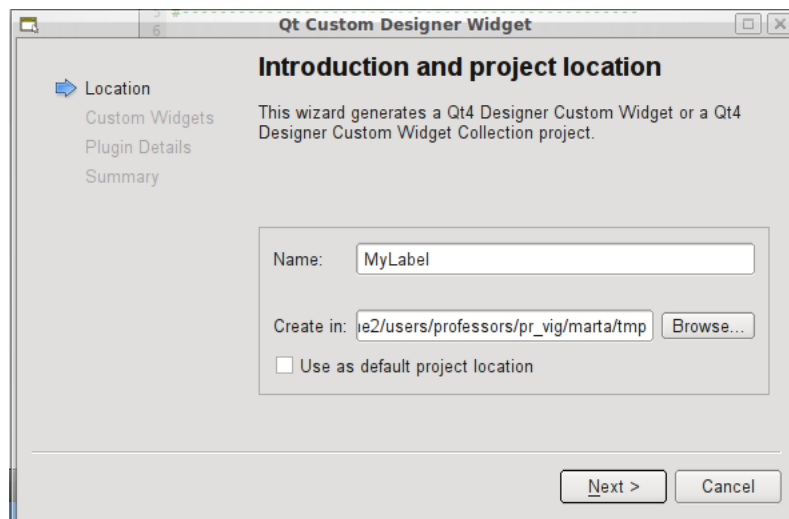
Per a crear un *custom widget* hem d'iniciar un nou projecte i seleccionar el tipus “Qt Custom Designer Widget” dins de “Other Project” (figura 3.9).

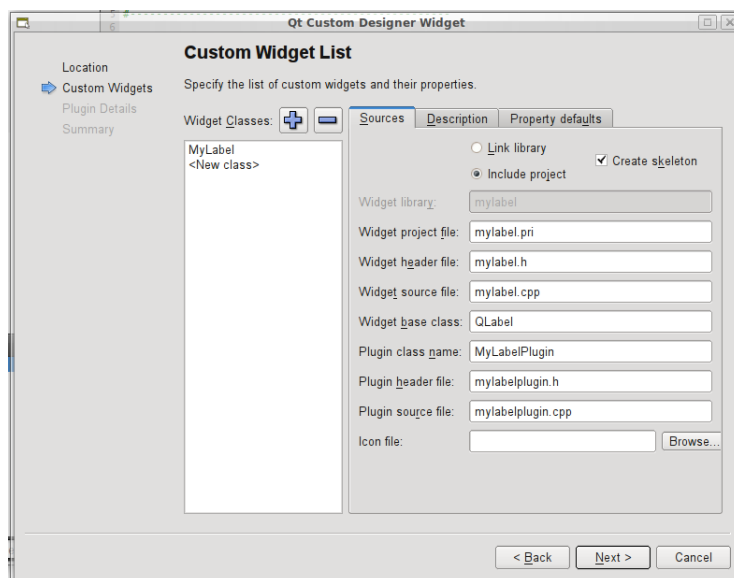
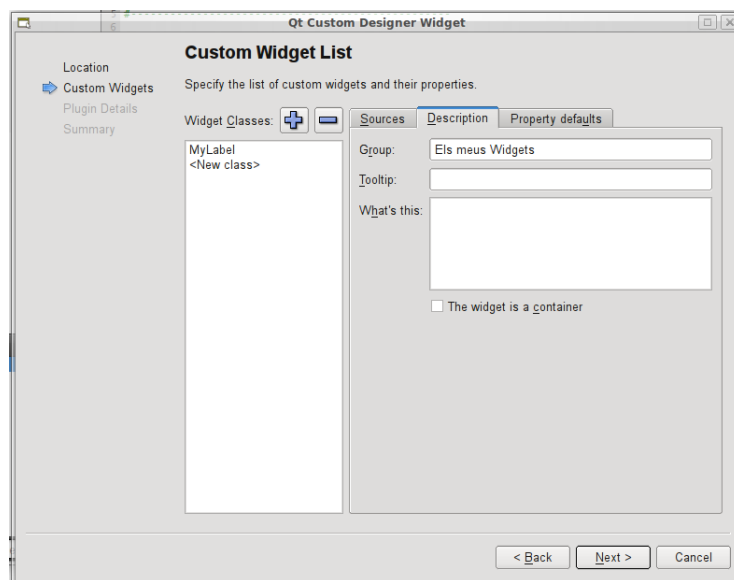
En la següent pantalla hem d'indicar el nom del projecte i la seva ubicació (figura 3.10). A continuació hem de posar nom a la classe que contindrà el *custom widget* i especificar quina és la seva classe base (figura 3.11). *QtCreator* genera els fitxers necessaris tant pel widget nou com pel *plugin*. A la pestanya **Description** (figura 3.12) podem especificar en quin grup del *QtDesigner* volem que el *plugin* aparegui i a la següent pantalla quin és el nom del *plugin* (figura 3.13). El llistat amb tots els fitxers creats es pot veure a la pantalla següent (figura 3.14).

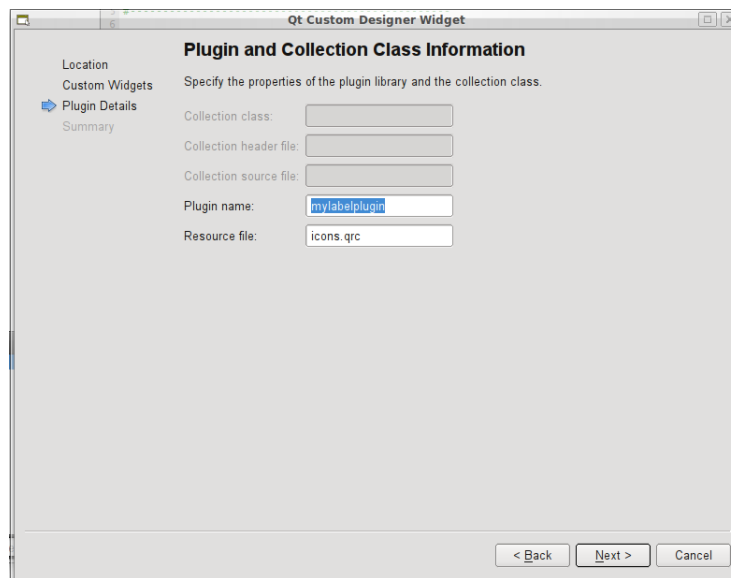
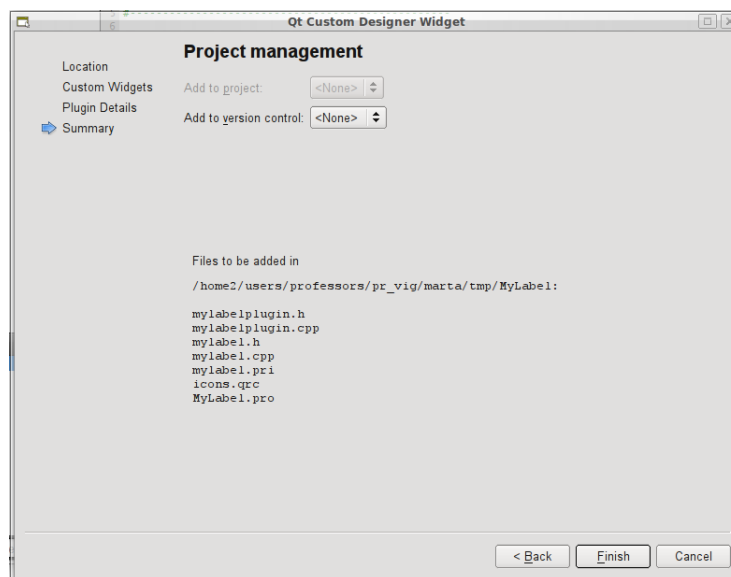
Un cop tenim el projecte creat ens apareix a l'esquerra de la pantalla l'arbre amb tots els fitxers que *QtCreator* ha generat. Per una banda tenim el `.h` i el `.cpp` del *custom widget* pròpiament dit i per altra tenim el `.h` i el `.cpp` del *plugin* (figura 3.15). Aquests darrers no caldrà tocar-los mai!

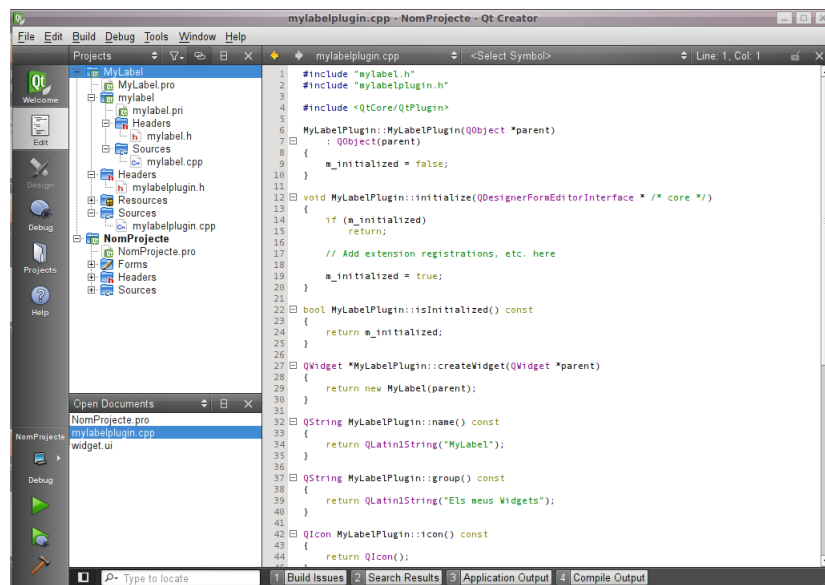
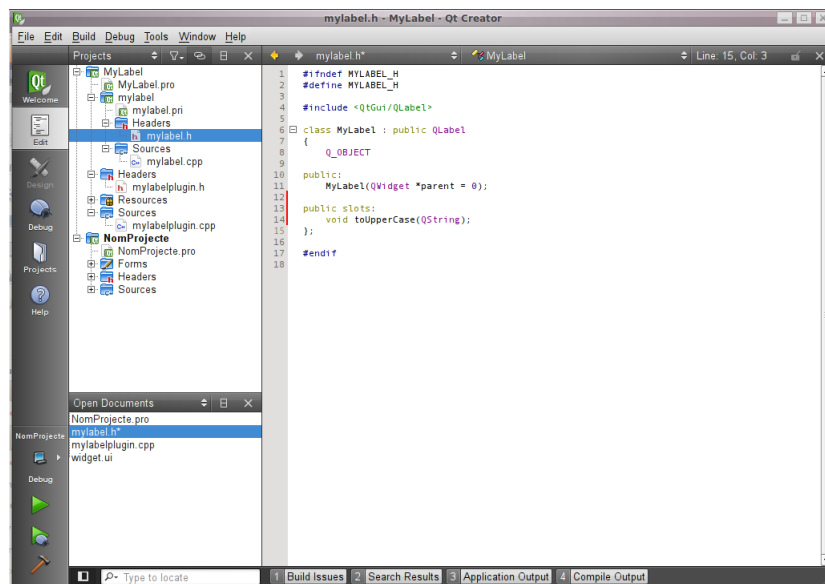
Fent doble clic sobre el fitxer `.h` del *custom widget* accedim al codi corresponent on podem incorporar les capçaleres dels signals i slots del nou *widget* (figura 3.16). Fent doble clic sobre el fitxer `.cpp` accedim al codi on només caldrà implementar els slots (figura 3.17).

Quan tinguem tot el necessari implementat en el *custom widget* hem de compilar el projecte i també instal·lar el *plugin* per tal que el *QtDesigner* el recongui i el carregui. *QtCreator* genera automàticament el fitxer `.pro` de

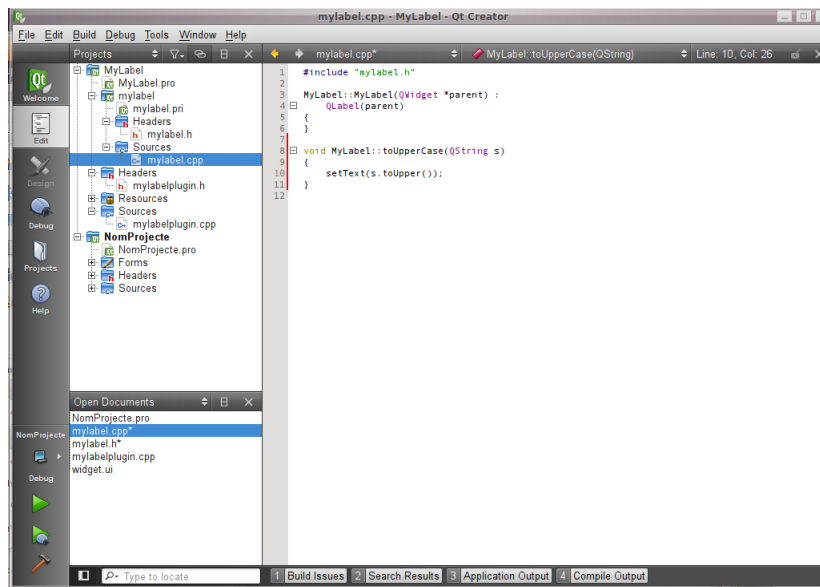
Figura 3.9: Construint un *custom widget*.Figura 3.10: Dades del projecte pel *custom widget*.

Figura 3.11: Dades del propi *custom widget*.Figura 3.12: Grup del *custom widget* per al *QtDesigner*.

Figura 3.13: Dades per al *plugin*.Figura 3.14: Fitxers generats automàticament per al *custom widget*.

Figura 3.15: Tots els fitxers afegits al *QtCreator*.Figura 3.16: Editant fitxer .h del *custom widget*.



Figura 3.17: Editant fitxer .cpp del *custom widget*.

definició de projecte que permet a **qmake** construir un fitxer **Makefile**. Tant **qmake** com **make** els executa *QtCreator* quan premem <Ctrl>B però no així el **make install** necessari per a instal·lar el *plugin*. Podem afegir aquest nou pas en la pestanya “Projects” com es veu a la figura 3.18.

En aquesta pantalla cal seleccionar primer el projecte que volem modificar de les pestanyes que hi ha a la part de dalt. A continuació hem d’afegir una nova etapa de construcció al bloc “Build Steps” o “Etapas de construcción”; el nou pas serà de tipus **make** i l’únic argument que cal afegir és “install”. Aquest nou pas de construcció s’afegeix automàticament després del **make** que compila el projecte del *custom widget*.

Per tal que el *QtDesigner* trobi i carregui correctament el *plugin* calen dues operacions molt importants que no ens podem descuidar:

1. Modificar el fitxer **.pro** del projecte del *custom widget* i indicar en quin directori s’instal·larà el *plugin* a la variable **target.path** (figura 3.19).
2. El directori que hem indicat abans ha de coincidir, excepte pel **/designer** del final, amb el que val la variable **QT\_PLUGIN\_PATH** que haurem definit a la MATEIXA CONSOLA on hem iniciat el *QtCreator* (figura 3.20).

Un cop fet això (només cal fer-ho una sola vegada) ja podem compilar i instal·lar el *plugin* amb <Ctrl>B. Si tot ha anat bé es crearà un fitxer **.so** que es copiarà al directori que hem especificat a **target.path**. Per a carregar el *plugin* anirem al *QtDesigner* dins del *QtCreator* i al menú “Tools/Form Editor/About

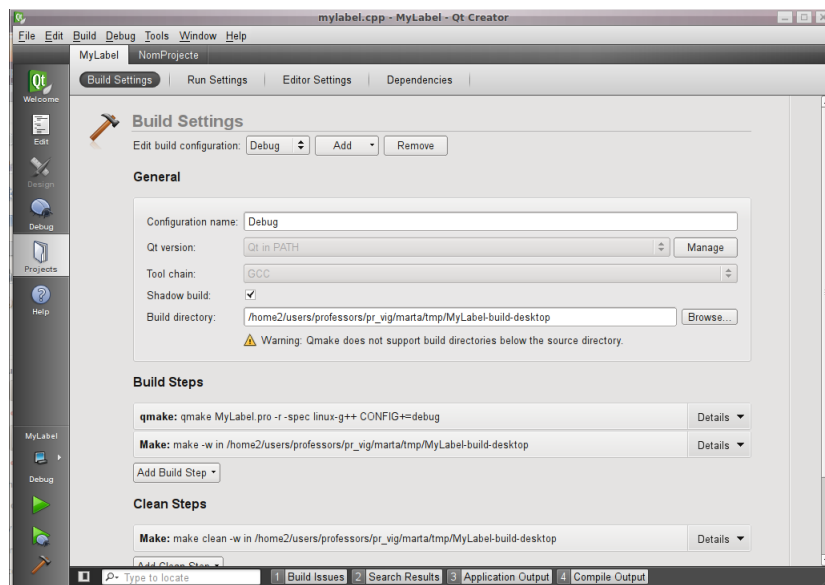
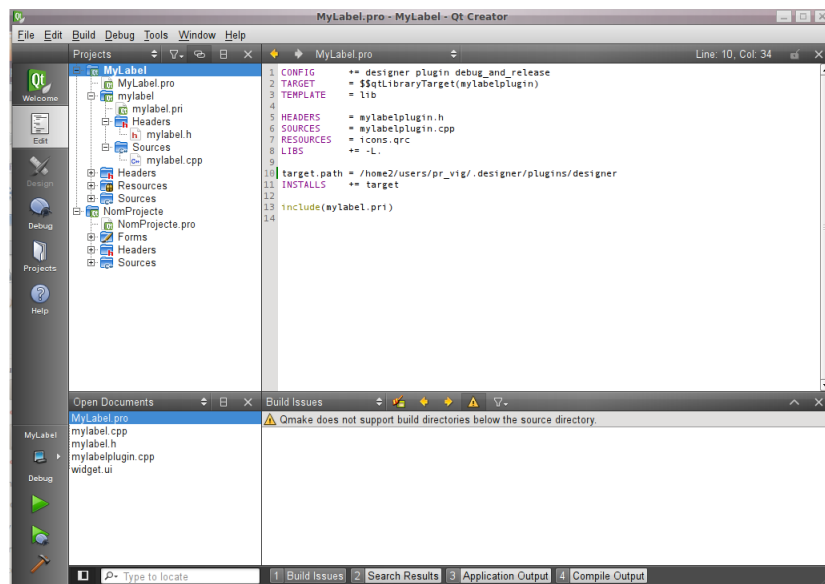


Figura 3.18: Editant passos de compilació en el projecte.

Figura 3.19: Edició del .pro del *custom widget*.

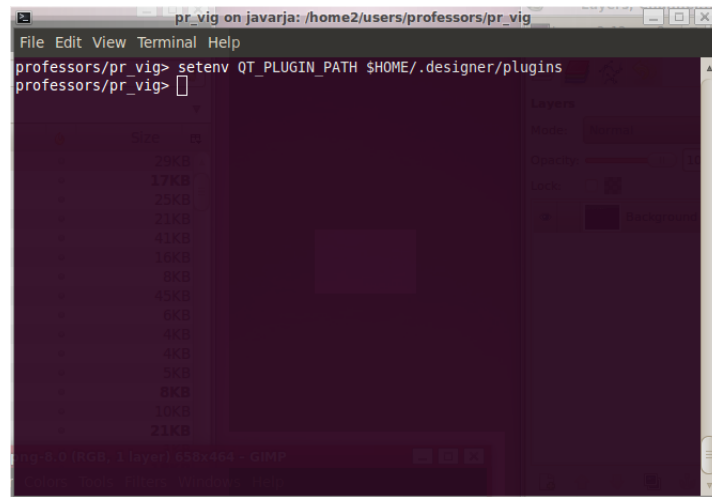


Figura 3.20: Consola des de la que cridarem al *QtCreator*.

Qt Designer Plugins” trobarem el botó ”Refresh” que recarrega els *plugins* nous (figura 3.21).

Quan feu una aplicació amb un diàleg on incorporeu un *plugin* cal necessàriament modificar la configuració de l’aplicació per tal que reconegui el *plugin*:

1. Amb el botó dret del ratolí sobre el nom del projecte seleccioneu “Add Existing Files” i afegiu el *.h* i el *.cpp* del *custom widget*.
2. Al fitxer *.pro* del projecte cal afegir la sentència: `INCLUDEPATH += directori on es troba el .h del custom widget` (veure figura 3.22).

En aquest punt ja es pot compilar el projecte que usa el *plugin* i provar la seva execució.

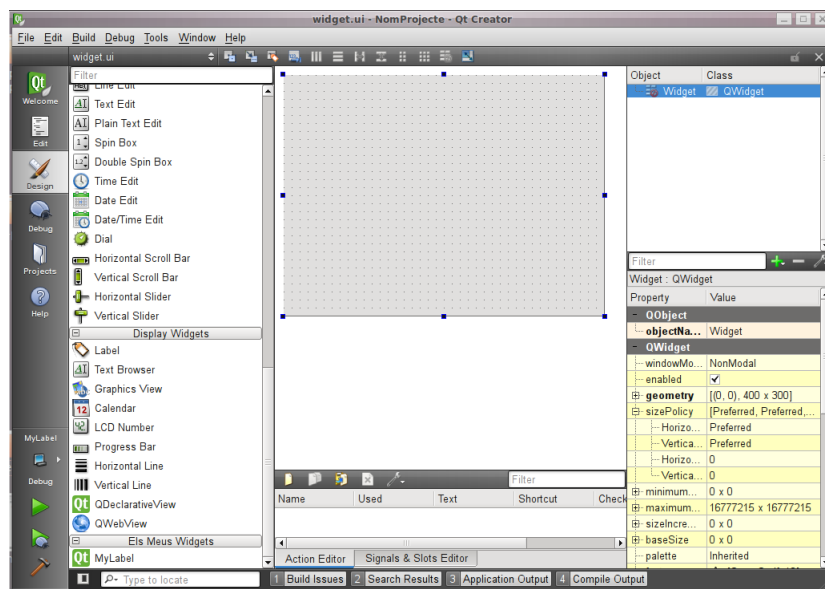
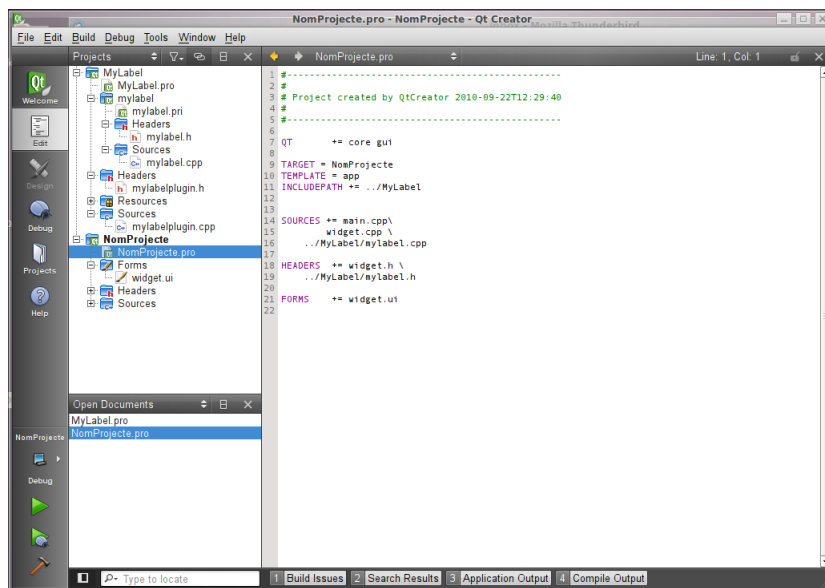
Figura 3.21: Designer amb el *plugin* instal·lat.

Figura 3.22: Afegint INCLUDEPATH al fitxer .pro del projecte.