Compilazione di un programma con QT

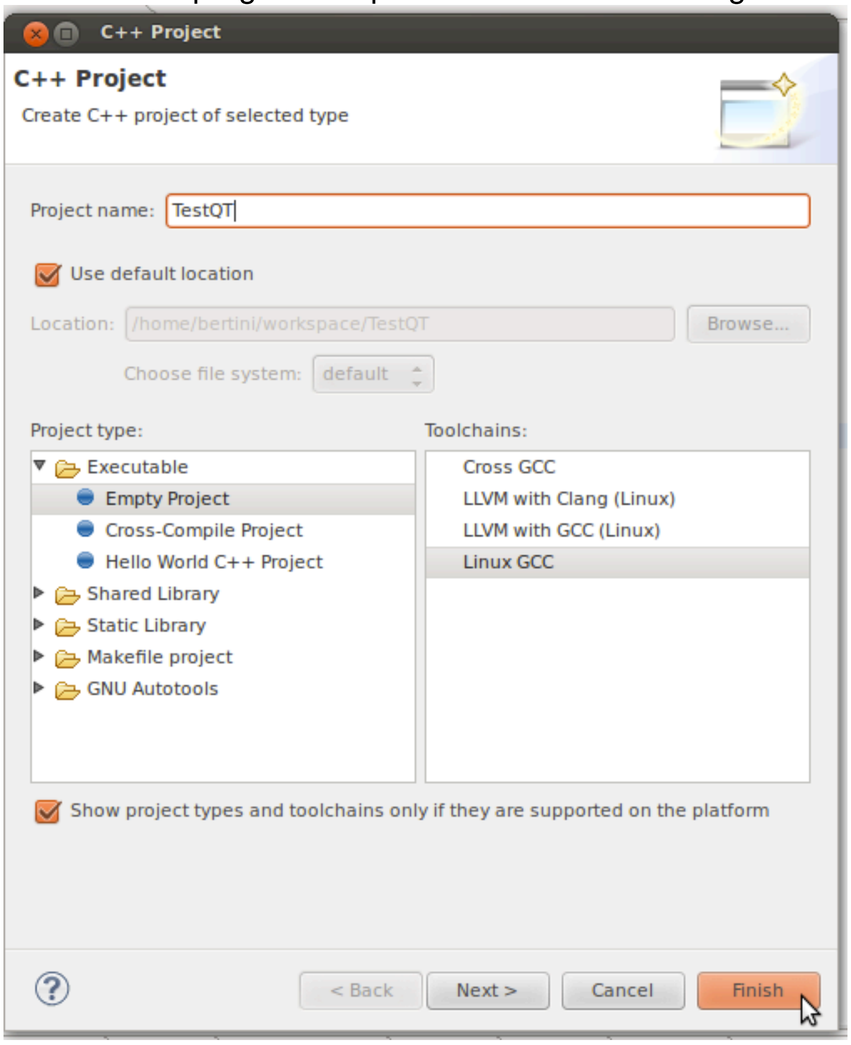
Nell'esempio seguente si mostra come compilare un programma con GUI basata sulle librerie QT. Nell'esempio è stato usato Ubuntu 12.04, Eclipse 3.7 e QT 4.8.
E' stato usato QT Designer per disegnare l'interfaccia e creare il codice C++ di base.

Questo tutorial si basa su:

- http://koehllab.genomecenter.ucdavis.edu/documentation/how-to/how-to-use-eclipse-with-qt
- http://sector.ynet.sk/qt4-tutorial/my-first-qt-gui-application.html

1. Installare le librerie QT comprensive dei file di sviluppo (e.g. libqt-dev), secondo le modalità del proprio sistema operativo. Eventualmente scaricare QT SDK dal sito web di Nokia: http://qt.nokia.com/downloads

2. Creare un progetto Eclipse C++ il cui Makefile è gestito da Eclipse:



C++ Project

Create C++ project of selected type

Project name: TestQT

[✓] Use default location

Location: /home/bertini/workspace/TestQT

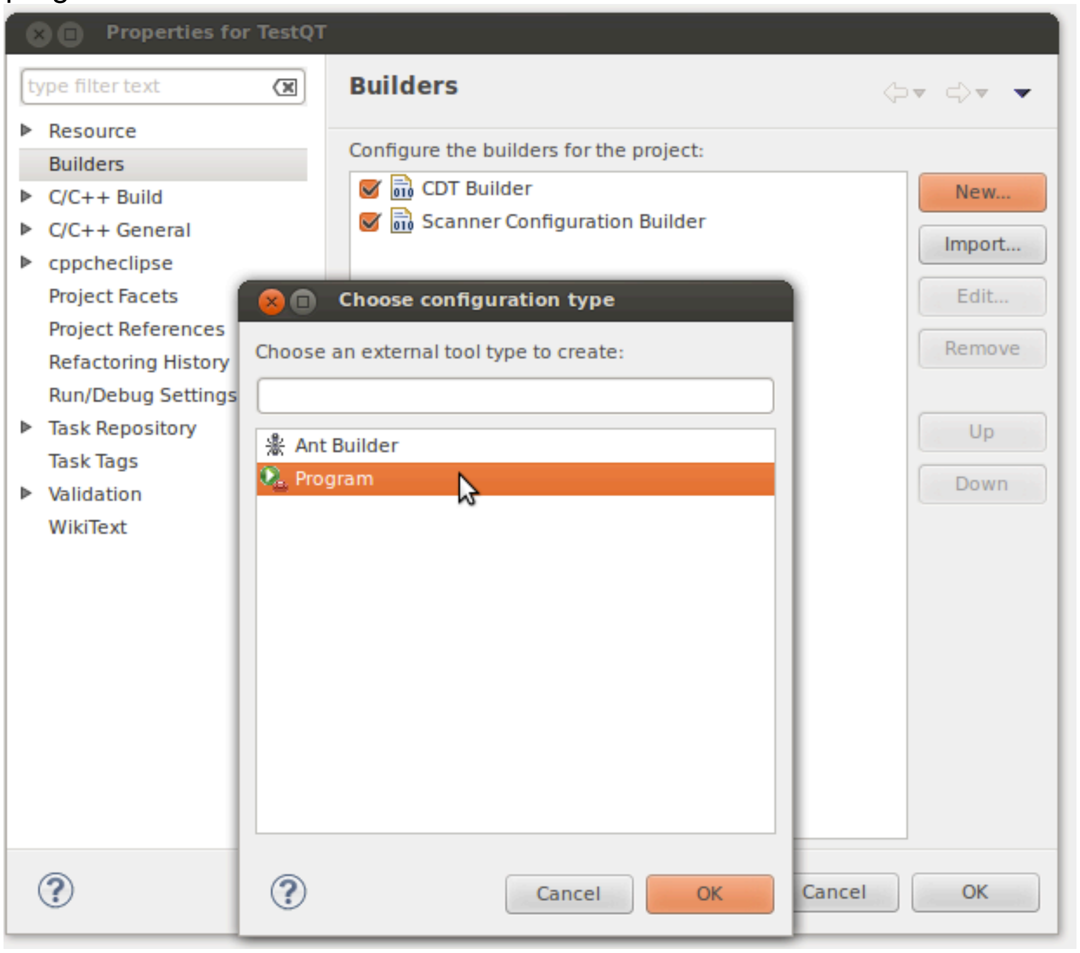
Choose file system: default

Project type: Executable > Empty Project

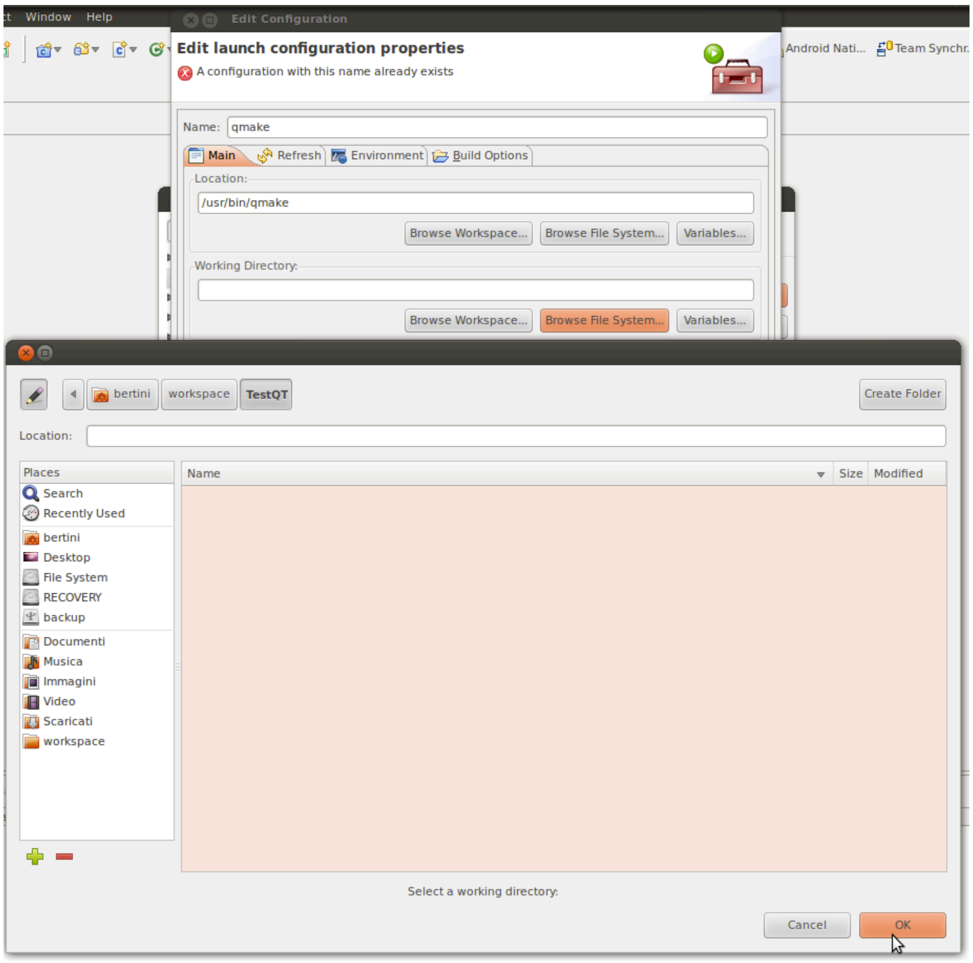
Toolchains: Linux GCC

[✓] Show project types and toolchains only if they are supported on the platform

3. Aprire le proprietà del progetto e nel tab Builders aggiungere un nuovo builder di tipo program:



4. Impostare il builder:
   1.Dare un nome al builder (es. qmake), nel campo “Location” scrivere il nome del programma qmake completo del path (eventualmente selezionandolo con il pulsante “Browse System File”).
   2. Nel campo “Working Directory” mettere la directory del progetto (es. usando il pulsante “Browse Workspace”)
   3. Nel tab “Build Options” impostare “During auto builds” e “During clean”, per far eseguire qmake in modo automatico nel caso il progetto abbia impostato l’autobuild o in caso se ne effettui una pulitura (i.e. cancellazione di file temporanei ed eseguibile).



Edit launch configuration properties

A configuration with this name already exists

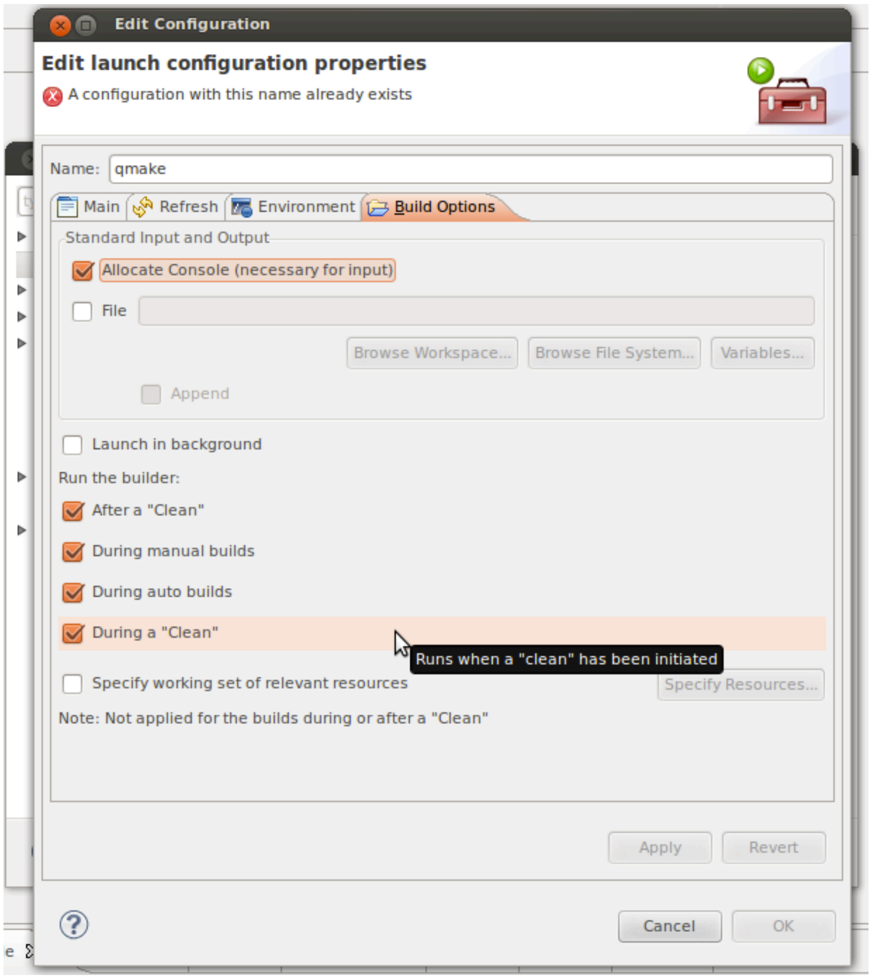
Name: qmake

Location:
/usr/bin/qmake

Working Directory:

Location:

Select a working directory:



Edit launch configuration properties

A configuration with this name already exists

Name: qmake

Standard Input and Output

- [x] Allocate Console (necessary for input)
- [ ] File
- [ ] Append

- [ ] Launch in background

Run the builder:

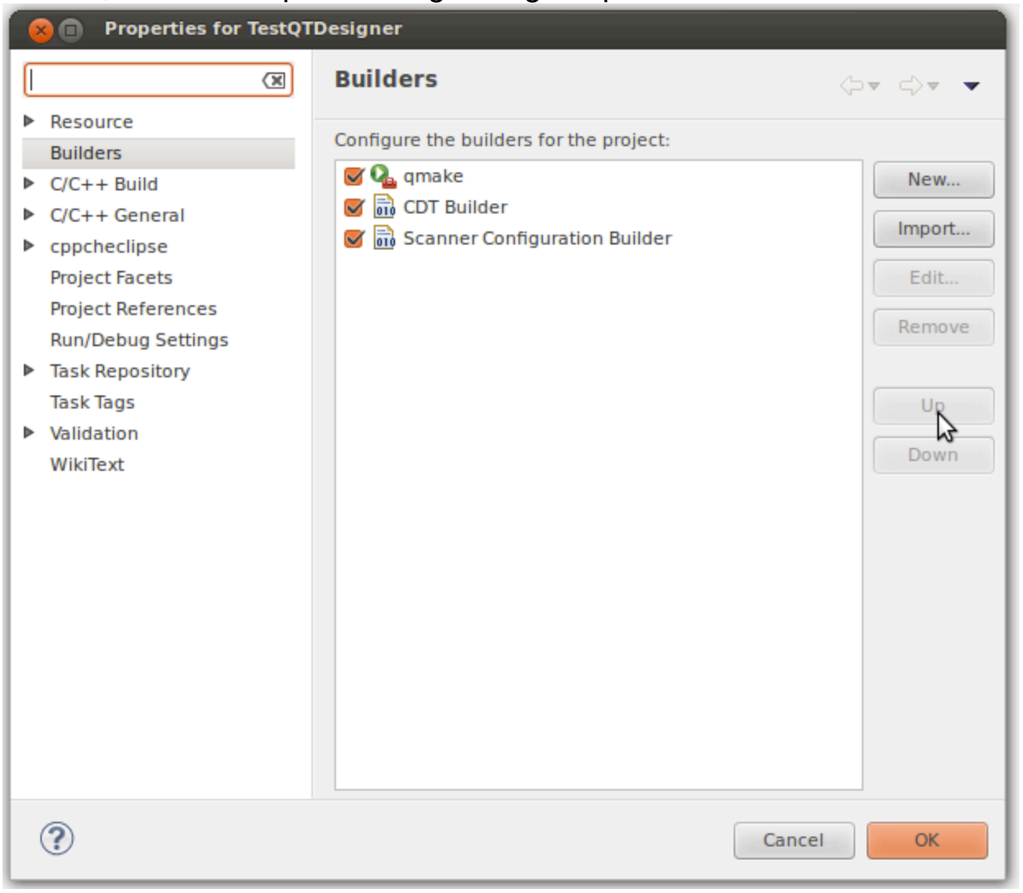
- [x] After a "Clean"
- [x] During manual builds
- [x] During auto builds
- [x] During a "Clean"

Runs when a "clean" has been initiated

- [ ] Specify working set of relevant resources

Note: Not applied for the builds during or after a "Clean"

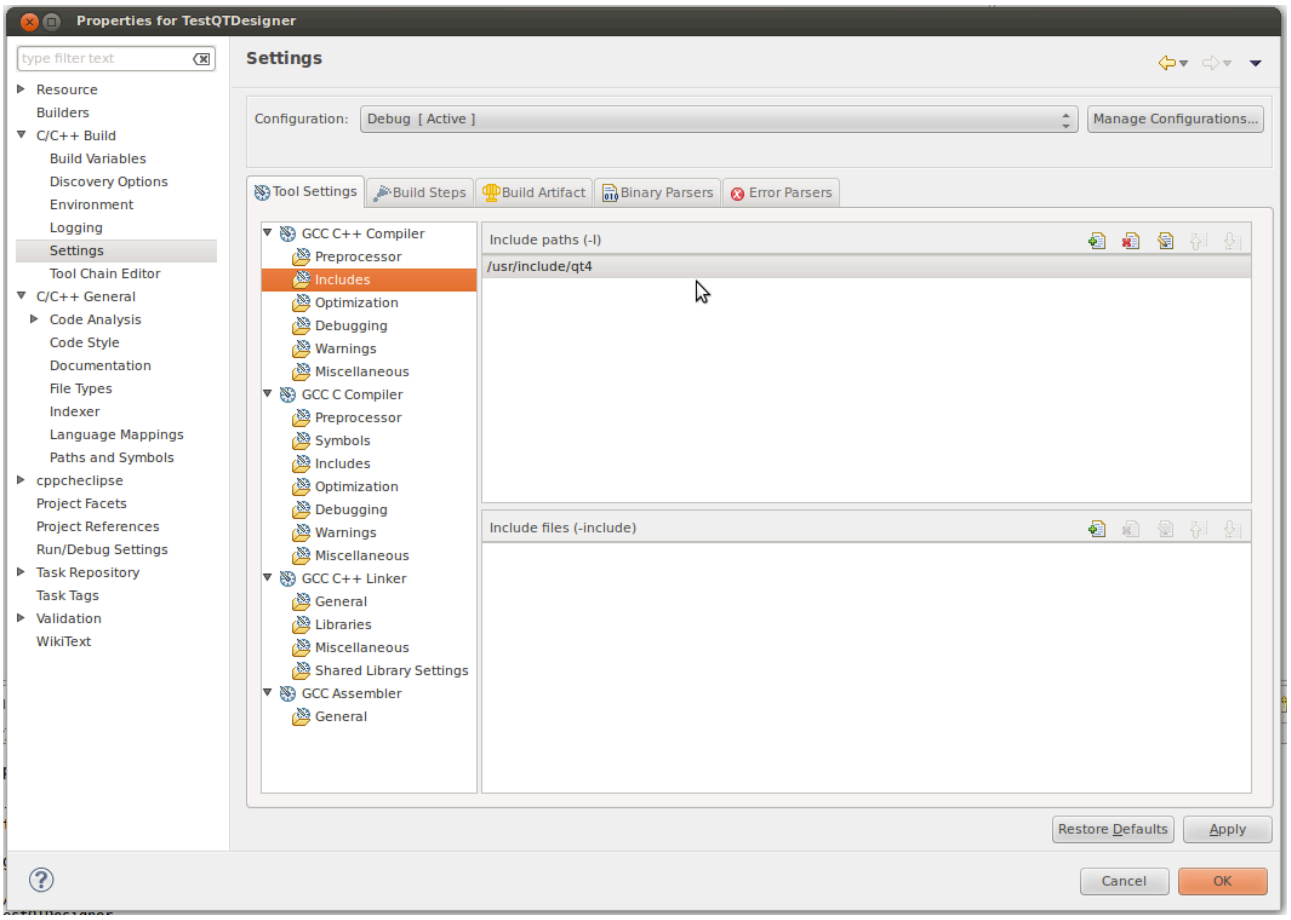
5. Usando il pulsante “Up” fare in modo che il builder appena creato preceda il CDT Builder, così che il qmake venga eseguito prima del make.



Configure the builders for the project:

- ✓ qmake
- ✓ CDT Builder
- ✓ Scanner Configuration Builder

6. Impostare il compilatore C++ ed il linker così che possano trovare dove sono gli header e le librerie:

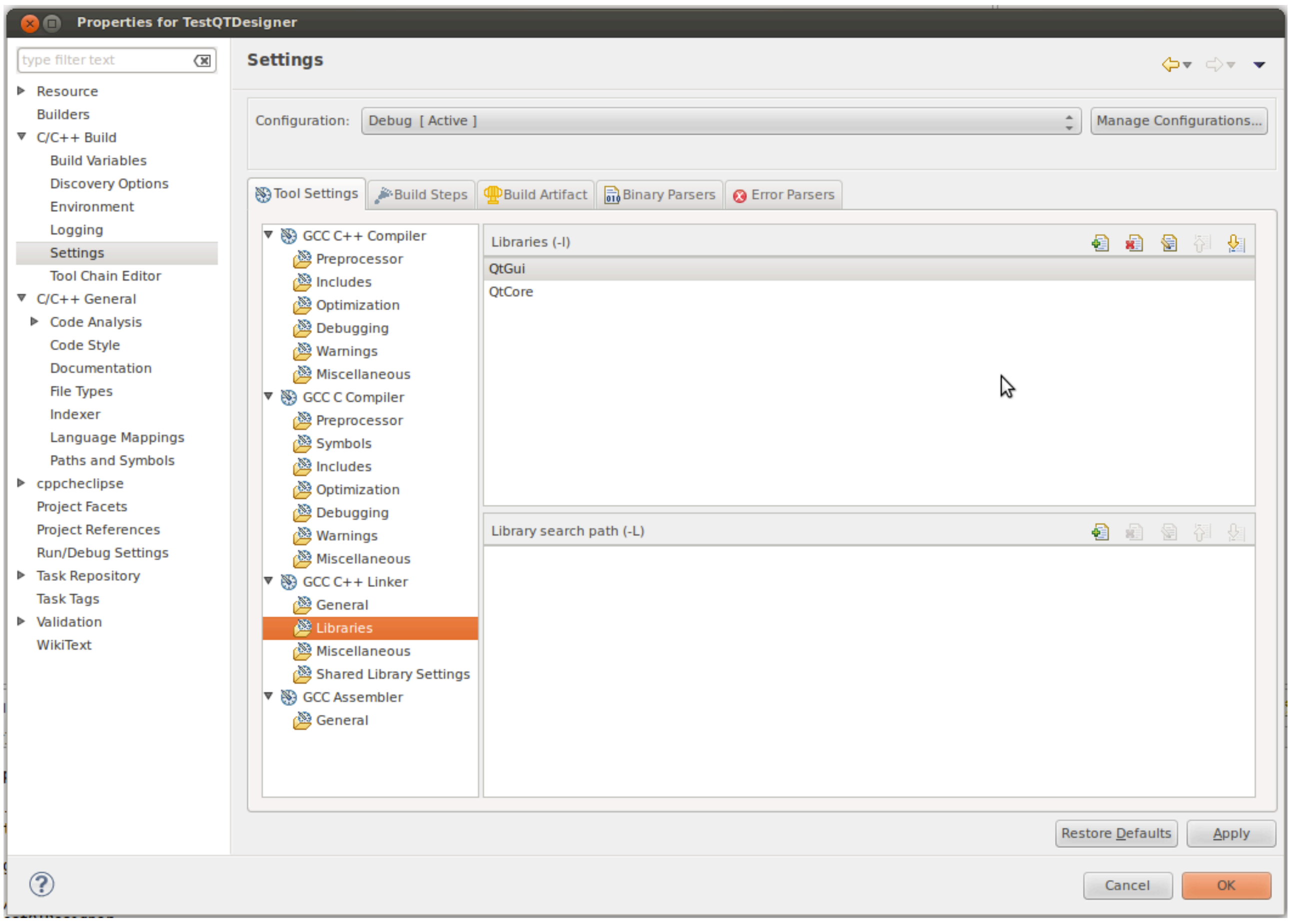


Settings

Configuration: Debug [Active]

Include paths (-I)
/usr/include/qt4

Include files (-include)



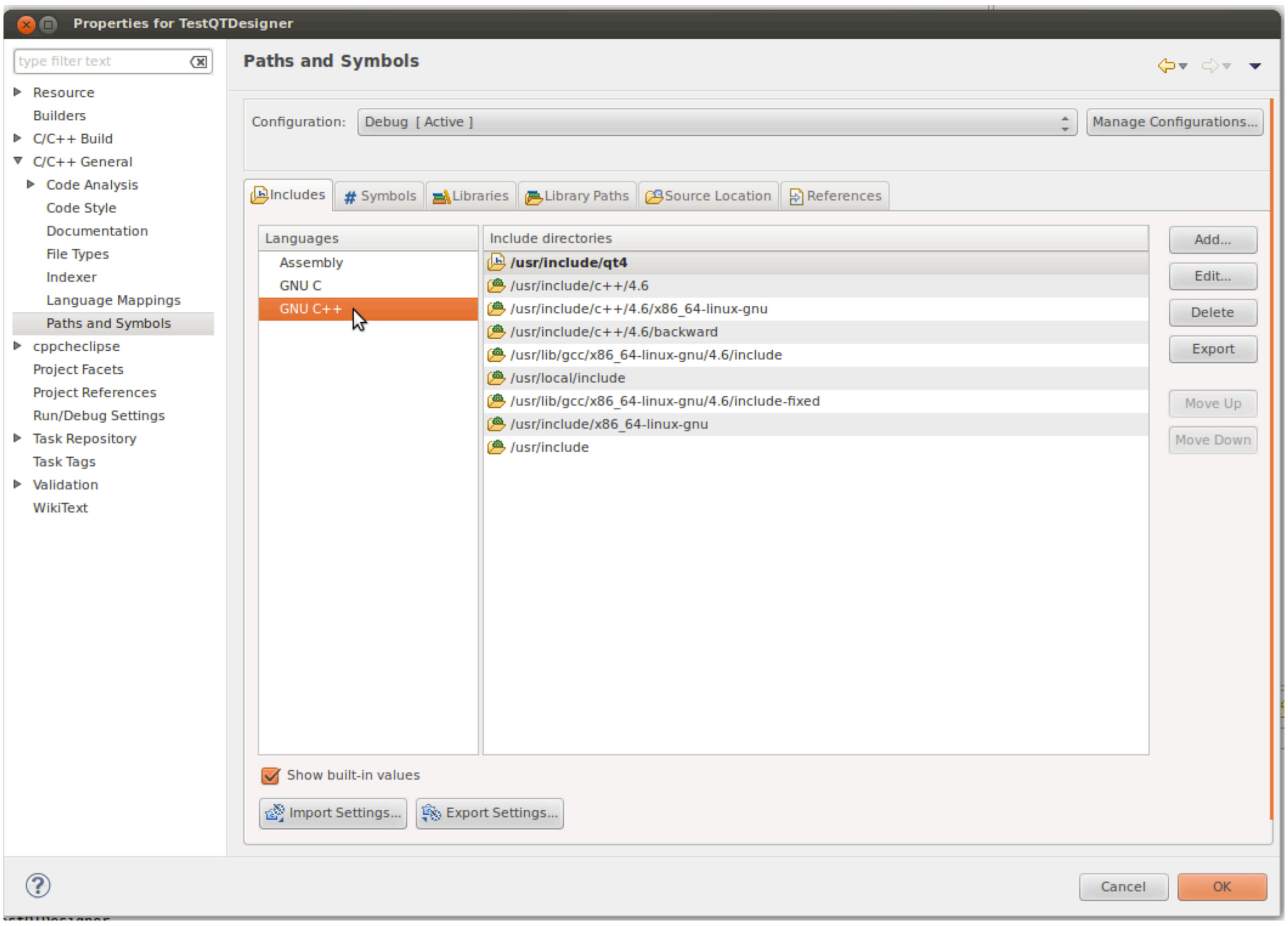
Settings

Configuration: Debug [Active]

Libraries (-l)
QtGui
QtCore

Library search path (-L)

Aggiungere il path degli header alla voce “Paths and Symbols” per far vedere al CODAN (l’analizzatore statico di codice C++ di Eclipse) i vari simboli (es. classi e strutture) usati in QT:


Properties for TestQTDesigner

Paths and Symbols

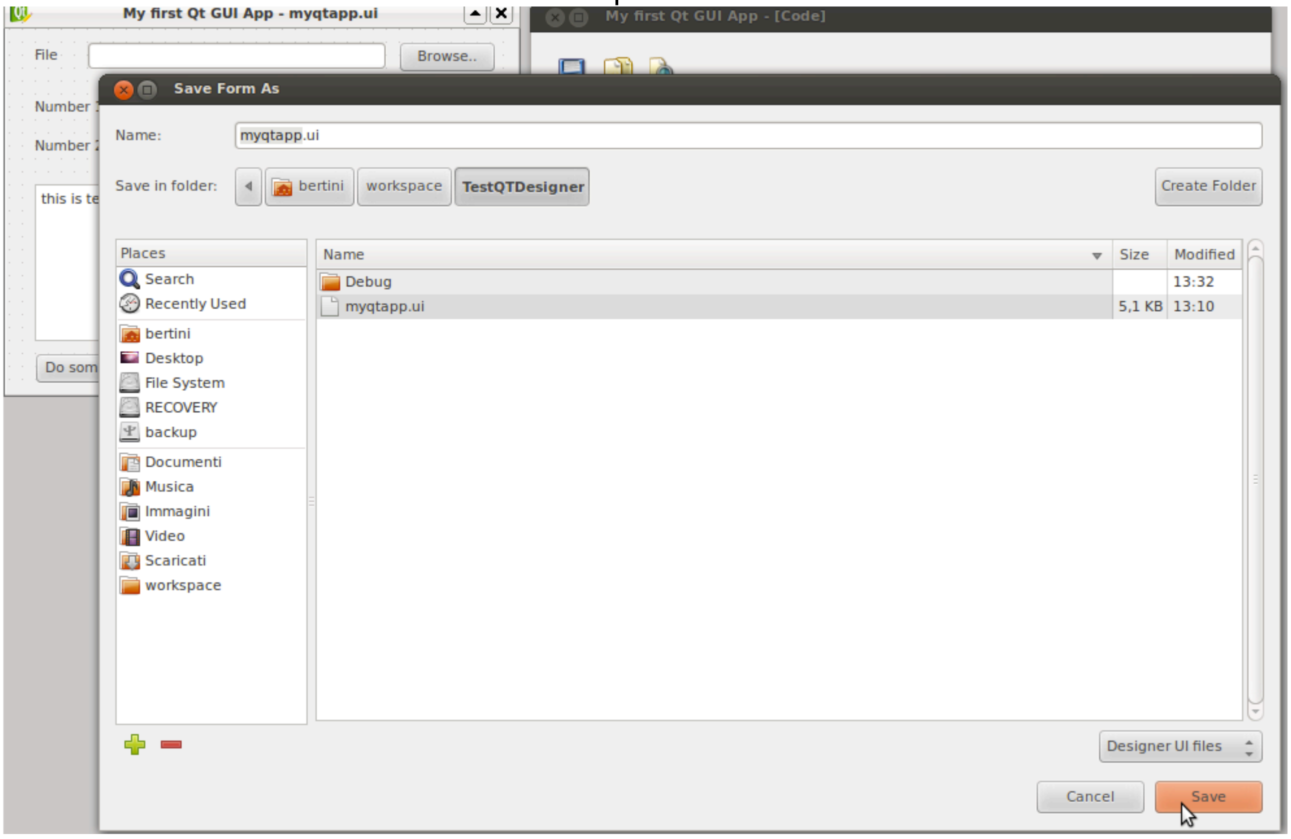
Configuration: Debug [Active]

Include directories (GNU C++):

```
/usr/include/qt4
/usr/include/c++/4.6
/usr/include/c++/4.6/x86_64-linux-gnu
/usr/include/c++/4.6/backward
/usr/lib/gcc/x86_64-linux-gnu/4.6/include
/usr/local/include
/usr/lib/gcc/x86_64-linux-gnu/4.6/include-fixed
/usr/include/x86_64-linux-gnu
/usr/include
```

Show built-in values ✓

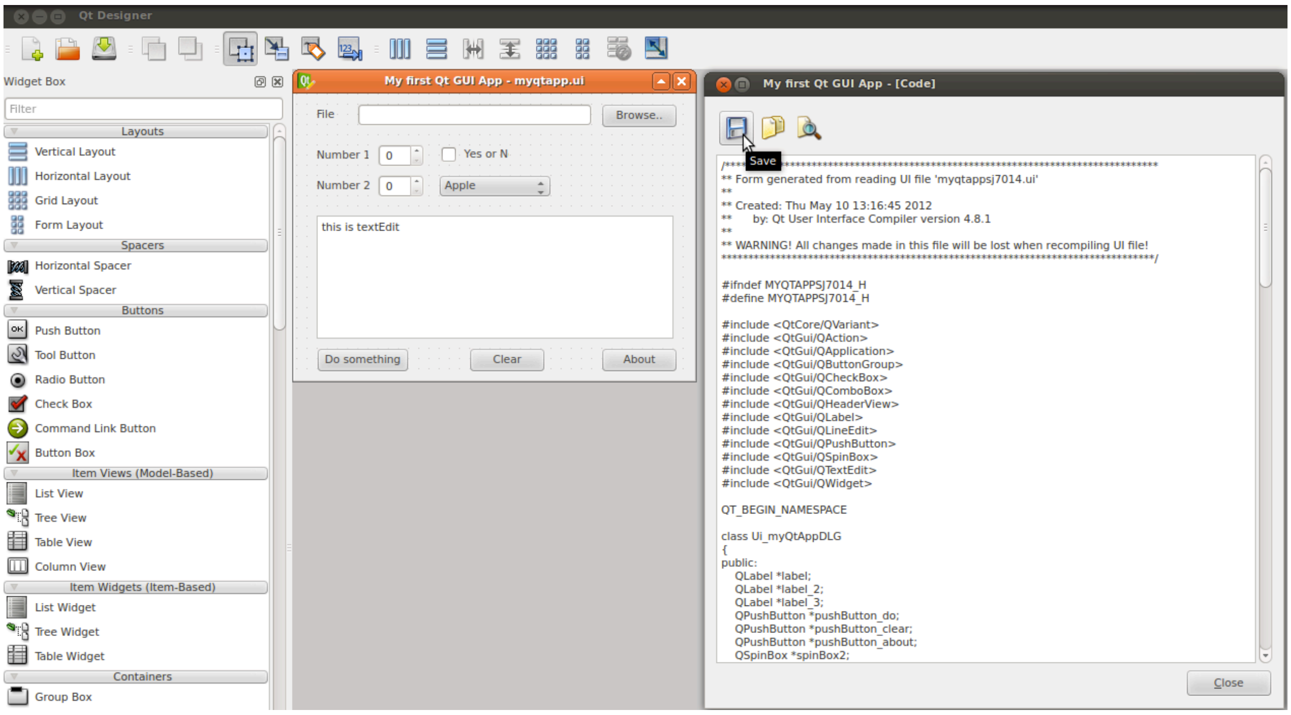
7. Aggiungere il codice del programma scheletro creato con QT Designer:
    1. il file .ui contenente l'interfaccia
    2. il file .h contenente il codice corrispondente



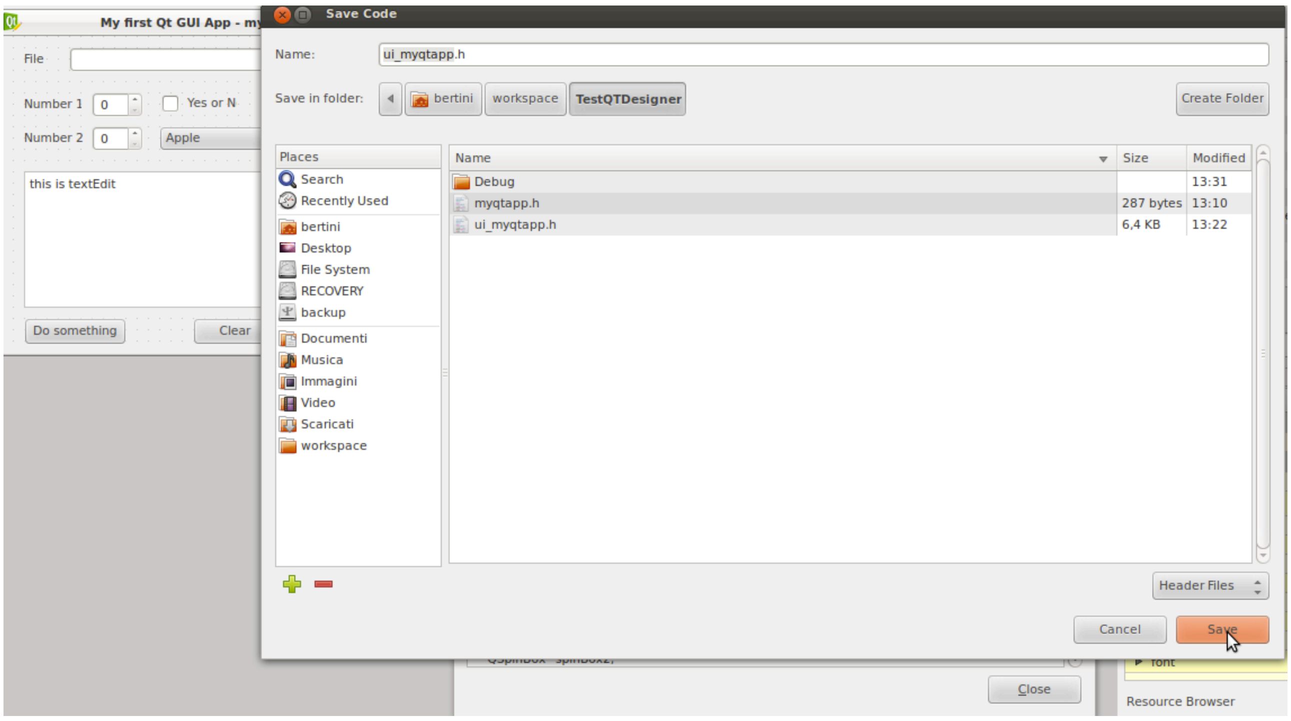
Name: myqtapp.ui

Save in folder: bertini / workspace / **TestQTDesigner**

Name	Size	Modified
Debug		13:32
myqtapp.ui	5,1 KB	13:10



```
/****************************************************************************
** Form generated from reading UI file 'myqtappsj7014.ui'
**
** Created: Thu May 10 13:16:45 2012
**      by: Qt User Interface Compiler version 4.8.1
**
** WARNING! All changes made in this file will be lost when recompiling UI file!
****************************************************************************/

#ifndef MYQTAPPSJ7014_H
#define MYQTAPPSJ7014_H

#include <QtCore/QVariant>
#include <QtGui/QAction>
#include <QtGui/QApplication>
#include <QtGui/QButtonGroup>
#include <QtGui/QCheckBox>
#include <QtGui/QComboBox>
#include <QtGui/QHeaderView>
#include <QtGui/QLabel>
#include <QtGui/QLineEdit>
#include <QtGui/QPushButton>
#include <QtGui/QSpinBox>
#include <QtGui/QTextEdit>
#include <QtGui/QWidget>

QT_BEGIN_NAMESPACE

class Ui_myQtAppDLG
{
public:
    QLabel *label;
    QLabel *label_2;
    QLabel *label_3;
    QPushButton *pushButton_do;
    QPushButton *pushButton_clear;
    QPushButton *pushButton_about;
    QSpinBox *spinBox2;
```



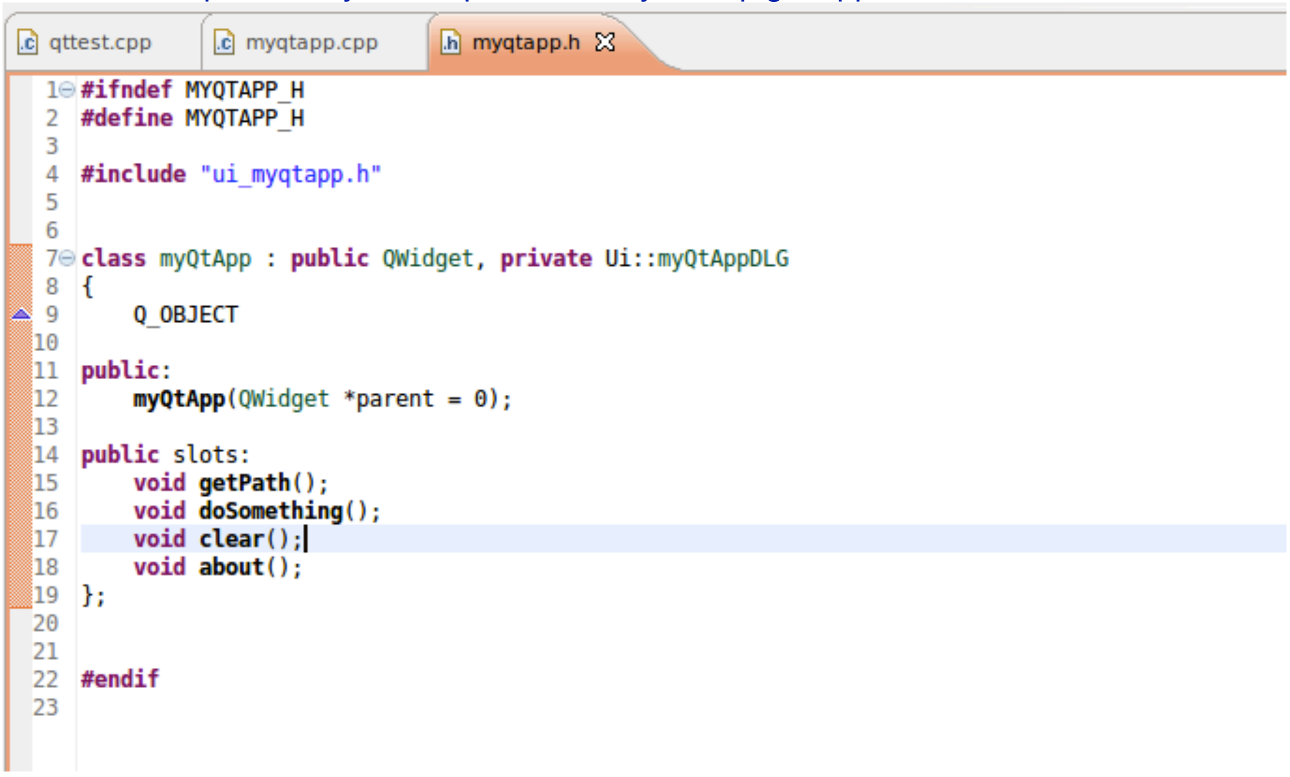
Name: ui_myqtapp.h

Save in folder: bertini / workspace / TestQTDesigner

Name	Size	Modified
Debug		13:31
myqtapp.h	287 bytes	13:10
ui_myqtapp.h	6,4 KB	13:22

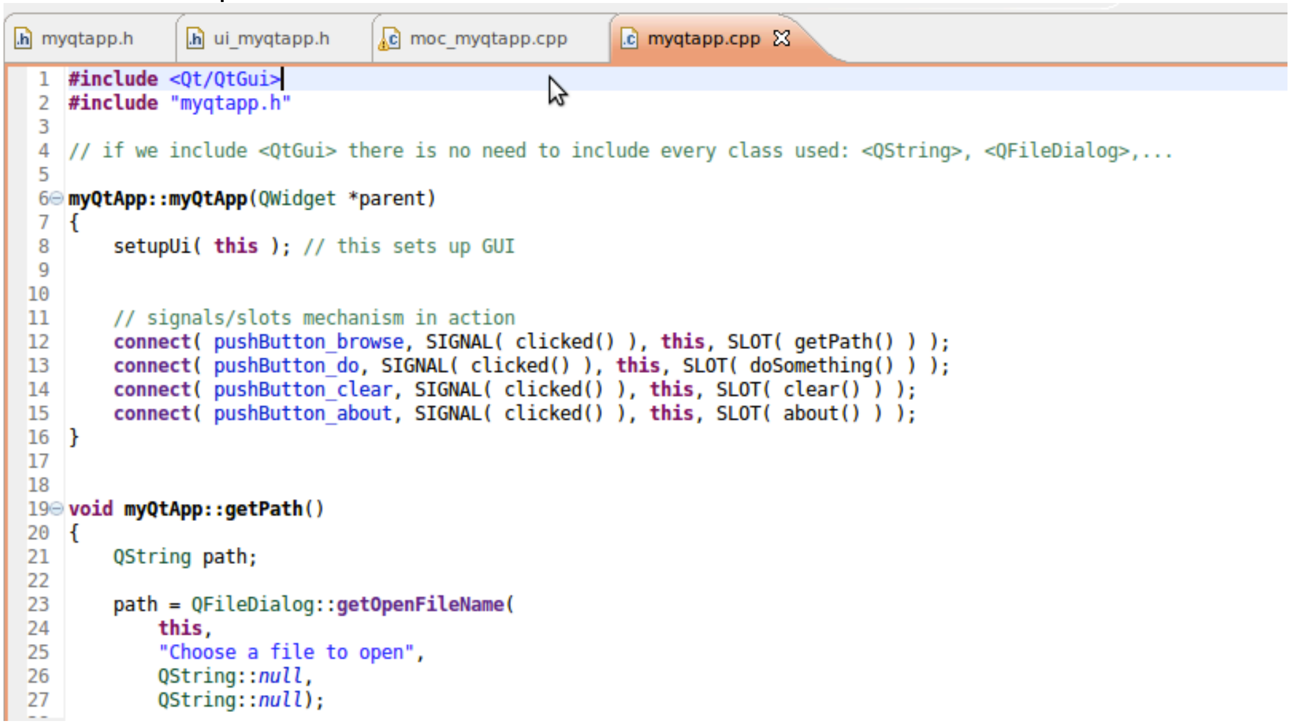
Header Files

8. Scrivere il programma secondo uno dei metodi indicati nel manuale di QT Designer o nel tutorial: http://sector.ynet.sk/qt4-tutorial/my-first-qt-gui-application.html



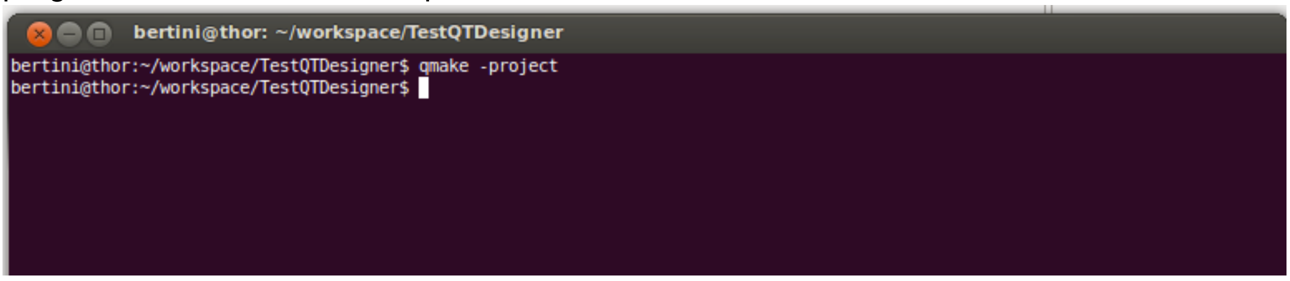
```
#ifndef MYQTAPP_H
#define MYQTAPP_H

#include "ui_myqtapp.h"


class myQtApp : public QWidget, private Ui::myQtAppDLG
{
	Q_OBJECT

public:
	myQtApp(QWidget *parent = 0);

public slots:
	void getPath();
	void doSomething();
	void clear();
	void about();
};


#endif
```

Stare attenti ai path dei file da includere:



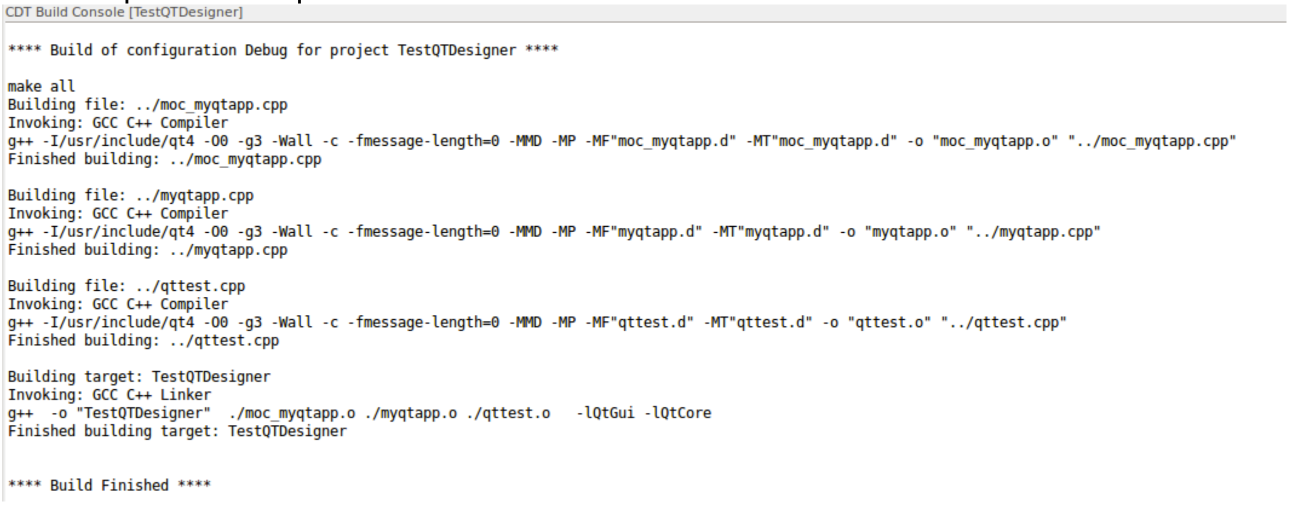
```
#include <Qt/QtGui>
#include "myqtapp.h"

// if we include <QtGui> there is no need to include every class used: <QString>, <QFileDialog>,...

myQtApp::myQtApp(QWidget *parent)
{
    setupUi( this ); // this sets up GUI


    // signals/slots mechanism in action
    connect( pushButton_browse, SIGNAL( clicked() ), this, SLOT( getPath() ) );
    connect( pushButton_do, SIGNAL( clicked() ), this, SLOT( doSomething() ) );
    connect( pushButton_clear, SIGNAL( clicked() ), this, SLOT( clear() ) );
    connect( pushButton_about, SIGNAL( clicked() ), this, SLOT( about() ) );
}


void myQtApp::getPath()
{
    QString path;

    path = QFileDialog::getOpenFileName(
        this,
        "Choose a file to open",
        QString::null,
        QString::null);
```

9. Aprire una shell ed eseguire il comando “qmake -project” nella cartella dove si trova il progetto. Verrà creato un file .pro



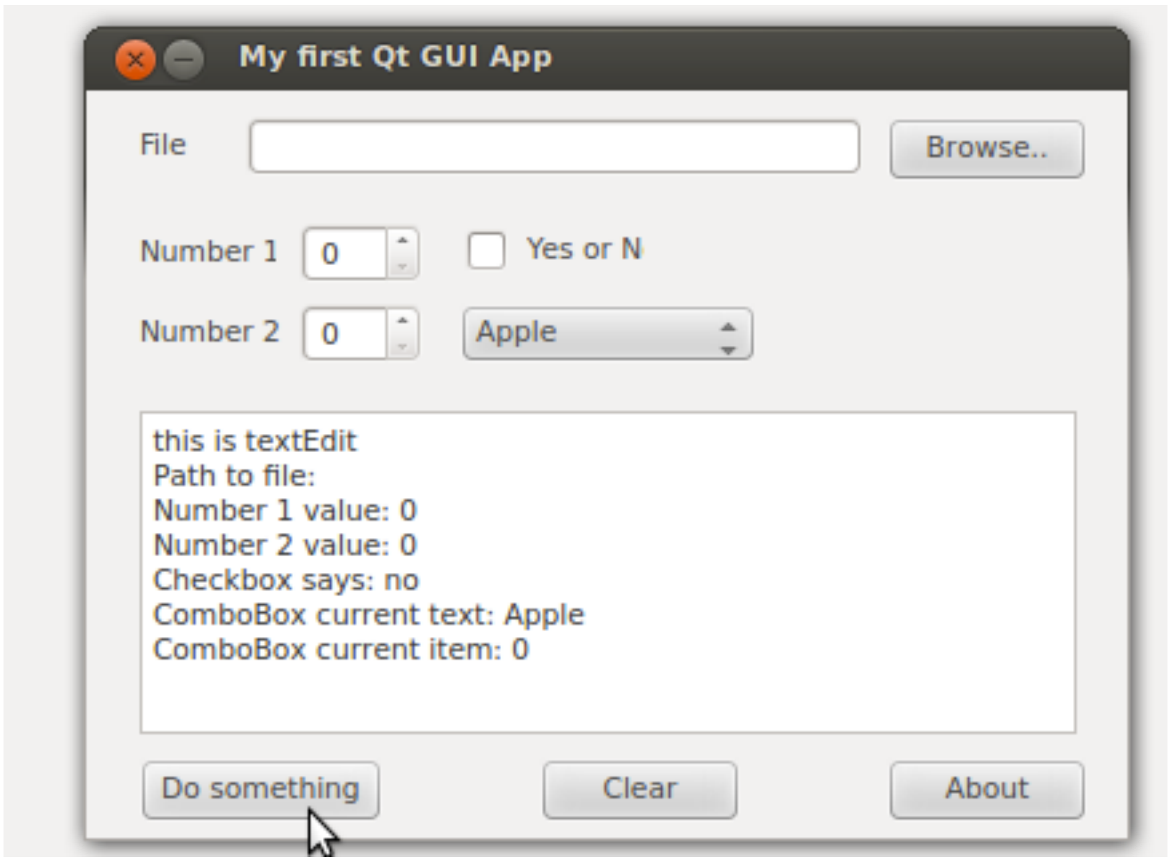
```
bertini@thor:~/workspace/TestQTDesigner$ qmake -project
bertini@thor:~/workspace/TestQTDesigner$ 
```

10. Compilare in Eclipse:



CDT Build Console [TestQTDesigner]

```
**** Build of configuration Debug for project TestQTDesigner ****

make all
Building file: ../moc_myqtapp.cpp
Invoking: GCC C++ Compiler
g++ -I/usr/include/qt4 -O0 -g3 -Wall -c -fmessage-length=0 -MMD -MP -MF"moc_myqtapp.d" -MT"moc_myqtapp.d" -o "moc_myqtapp.o" "../moc_myqtapp.cpp"
Finished building: ../moc_myqtapp.cpp

Building file: ../myqtapp.cpp
Invoking: GCC C++ Compiler
g++ -I/usr/include/qt4 -O0 -g3 -Wall -c -fmessage-length=0 -MMD -MP -MF"myqtapp.d" -MT"myqtapp.d" -o "myqtapp.o" "../myqtapp.cpp"
Finished building: ../myqtapp.cpp

Building file: ../qttest.cpp
Invoking: GCC C++ Compiler
g++ -I/usr/include/qt4 -O0 -g3 -Wall -c -fmessage-length=0 -MMD -MP -MF"qttest.d" -MT"qttest.d" -o "qttest.o" "../qttest.cpp"
Finished building: ../qttest.cpp

Building target: TestQTDesigner
Invoking: GCC C++ Linker
g++  -o "TestQTDesigner"  ./moc_myqtapp.o ./myqtapp.o ./qttest.o   -lQtGui -lQtCore
Finished building target: TestQTDesigner


**** Build Finished ****
```



```
this is textEdit
Path to file:
Number 1 value: 0
Number 2 value: 0
Checkbox says: no
ComboBox current text: Apple
ComboBox current item: 0
```