National Taipei University of Technology

Object-Oriented Programming (Fall, 2008)
Homework # 4

0. Introduction

This document mainly describes to write and build Qt programs under Windows XP operating system using Eclipse IDE. We choose the Eclipse as our develop environment. Eclipse is an open source community whose projects are focused on providing an extensible development platform and frameworks for building software. In order to get support for Qt development in Eclipse platform, the Qt Eclipse Integration must be installed to extend the functionality. The installation steps are listed bellow:

1. Install Qt(4.4.1)
2. Install Qt Eclipse Integration(1.4.1)
3. Compile Qt Test Runner library
4. Test Qt develop environment

Reference:

http://trolltech.com/developer/faqs/Qt/installation

http://dist.trolltech.com/pdf/eclipse-whitepaper.pdf

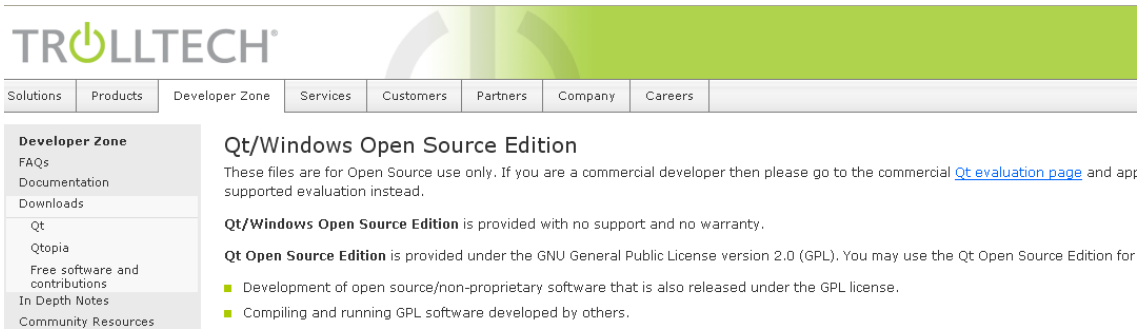
http://cppunit.sourceforge.net/cppunit-wiki/QtTestRunnerWithEclipse

1. Install Qt

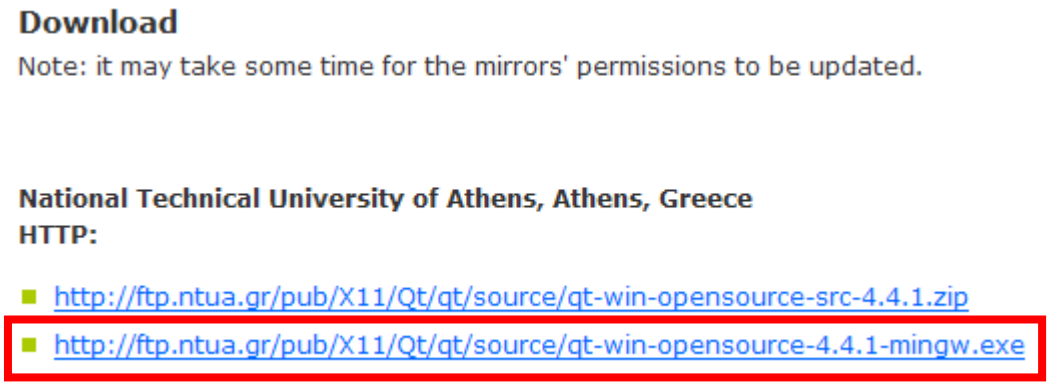
Qt is a cross-platform application development framework, widely used for the development of GUI programs, and also used for developing non-GUI programs such as console tools and servers. Qt is most notably used in KDE, the web browser Opera, Google Earth, Skype, Qtopia and OPIE. It is produced by the Norwegian company Trolltech. Trolltech insiders pronounce Qt as "cute".

1.1. Download Qt

Go to Trolltech website at http://trolltech.com/developer/downloads/qt/windows and you will see:



In the file list, locate and click the current version of Qt. This will have an .exe extension as shown here:

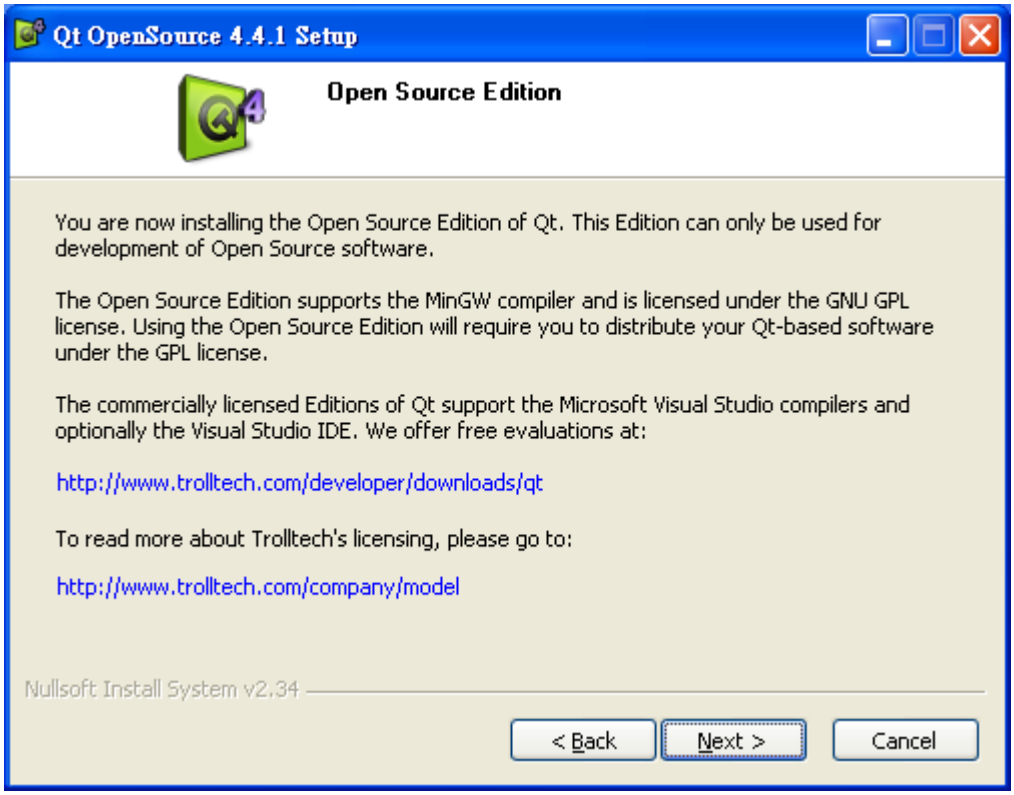


1.2. Install Qt

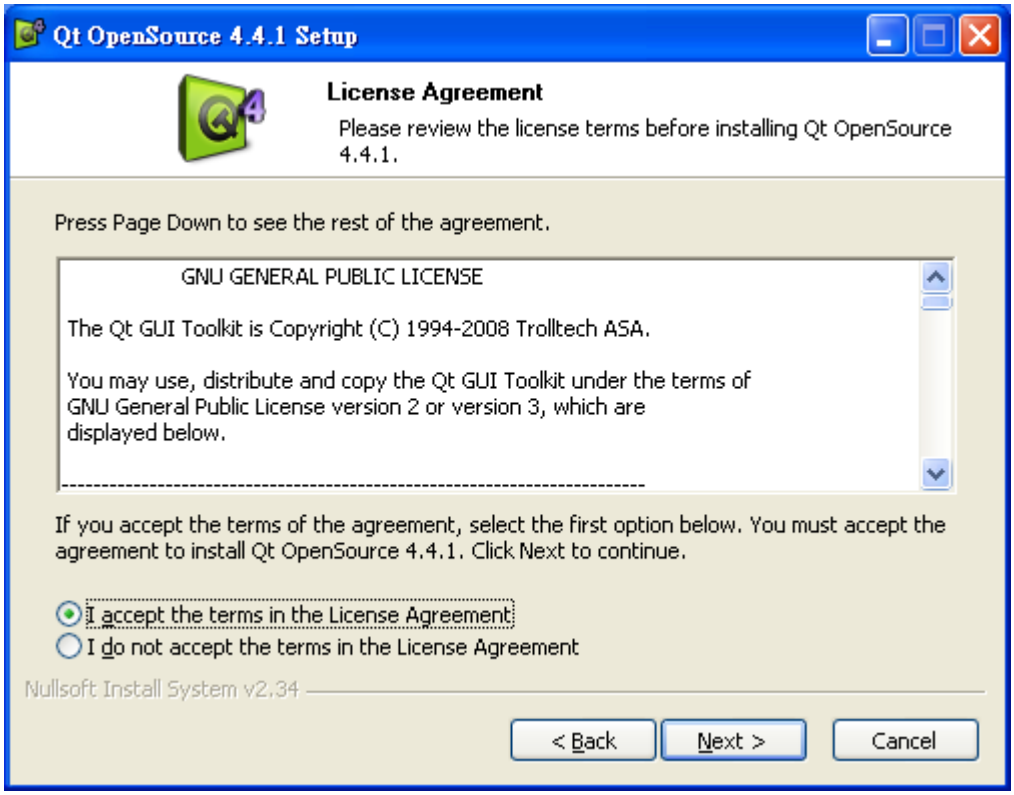
Using Windows Explorer, locate the file you downloaded, and double-click it:

Qt uses a regular "InstallShield" type installer. Click **Next** when the introductory screen, shown here, appears:

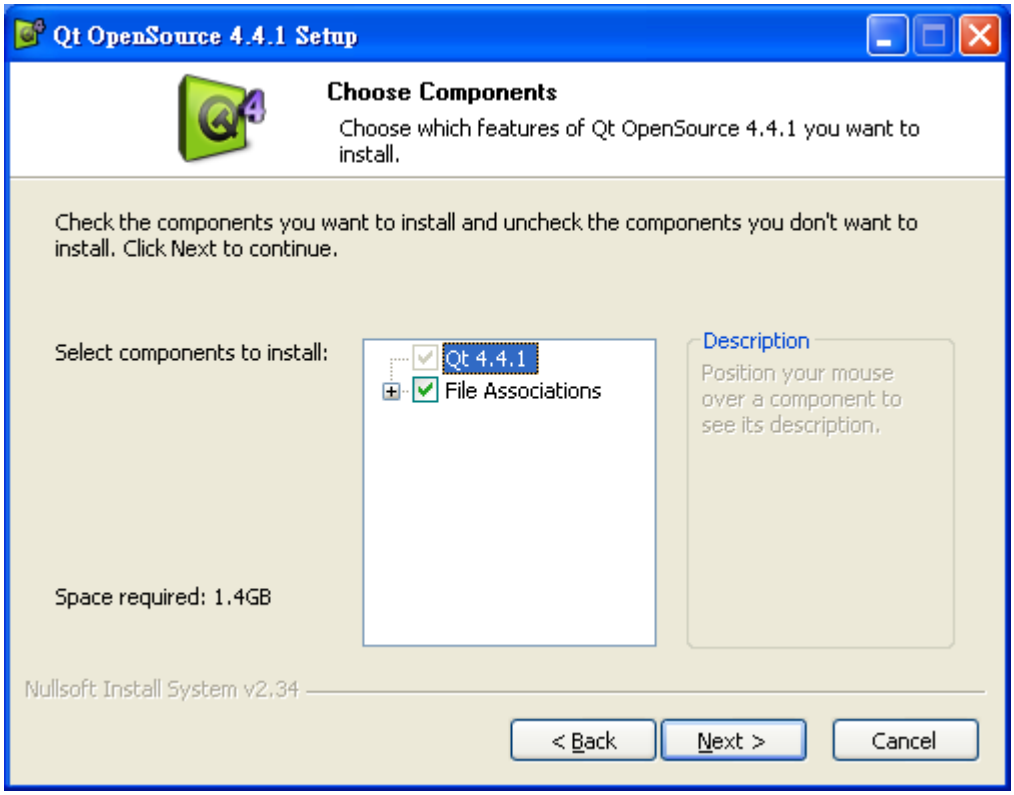
Click Next to install the Qt:



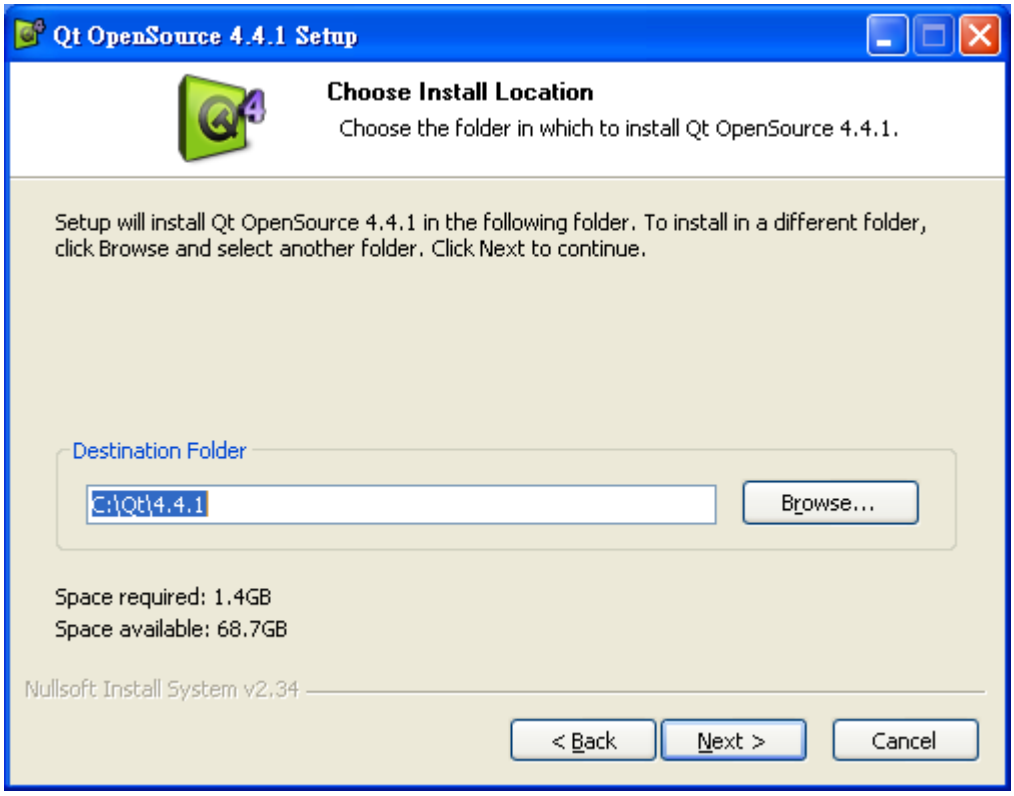
Select “I accept the terms in the License Agreement” and then click **Next** to installation:



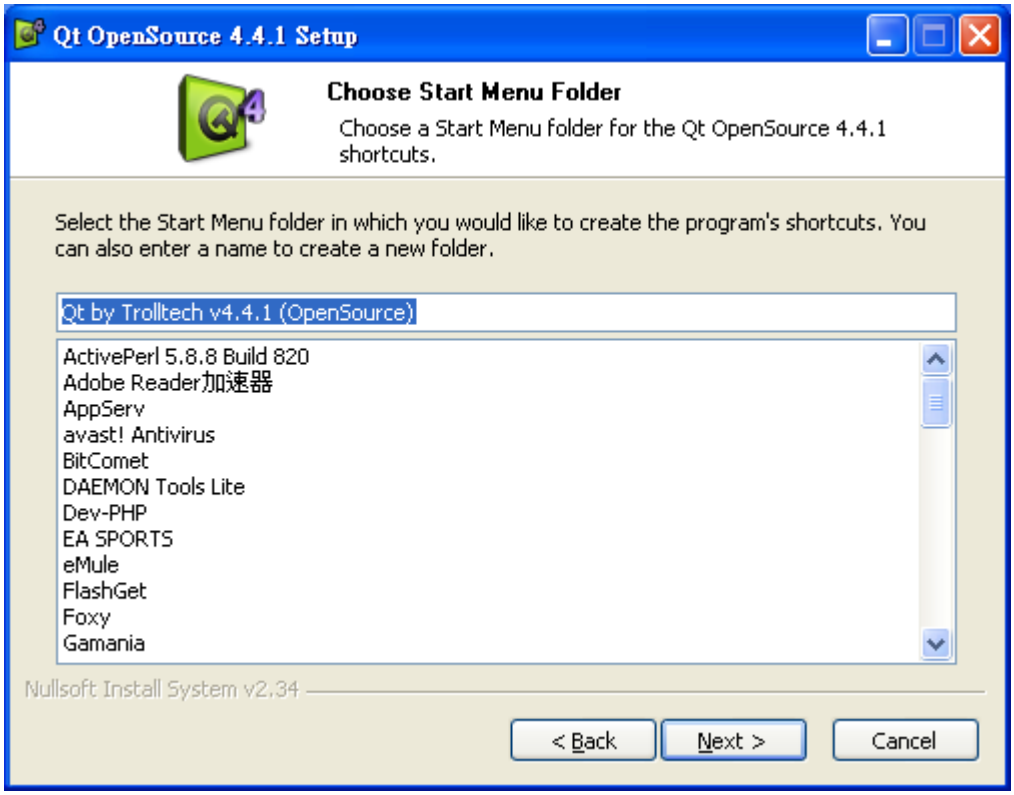
Click **Next**:



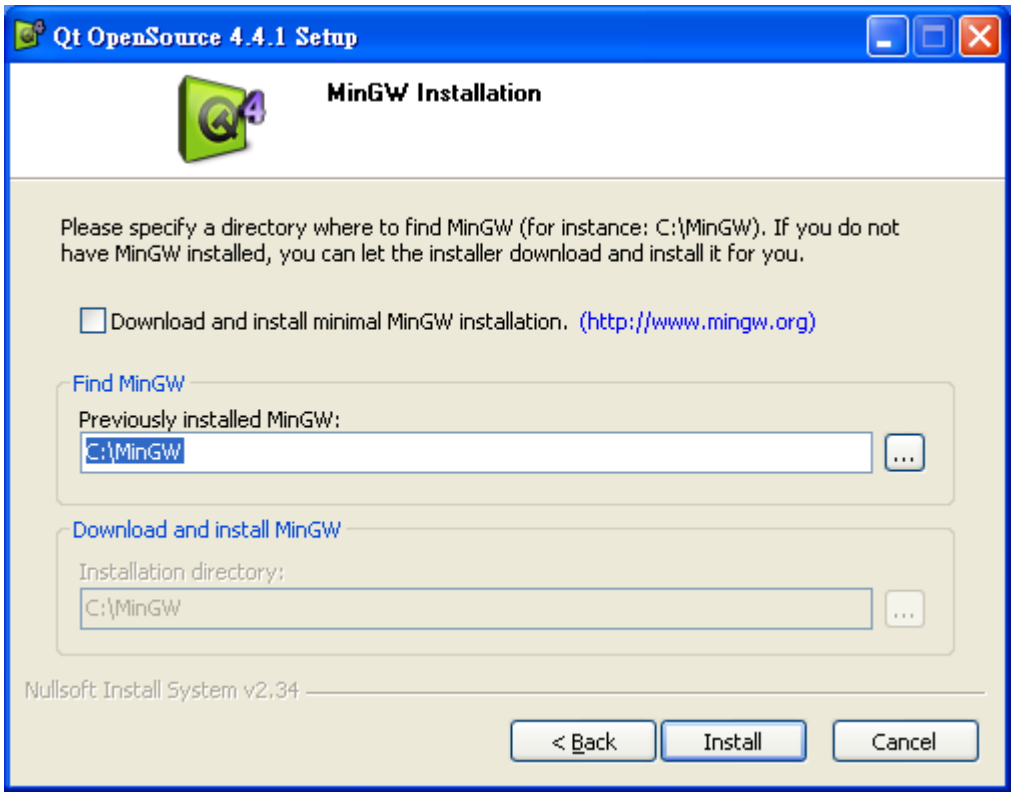
Select a folder where you would like to install Qt. Click **Next**:



Click **Next**:

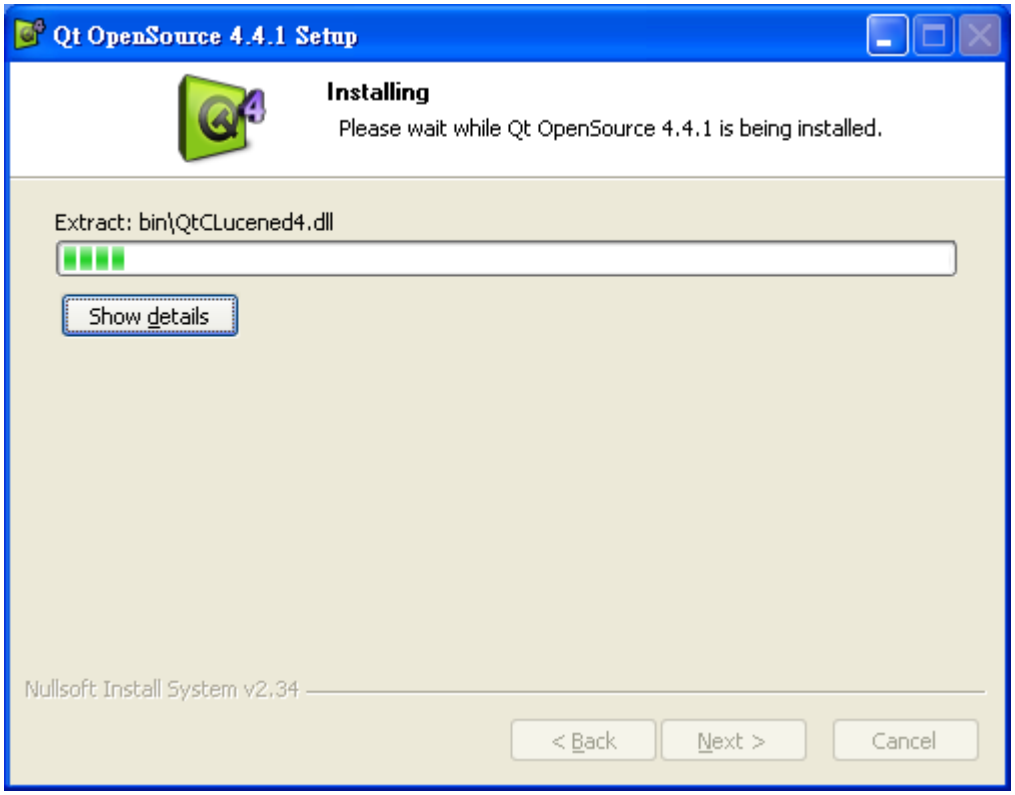


Click **Install** to begin installation:

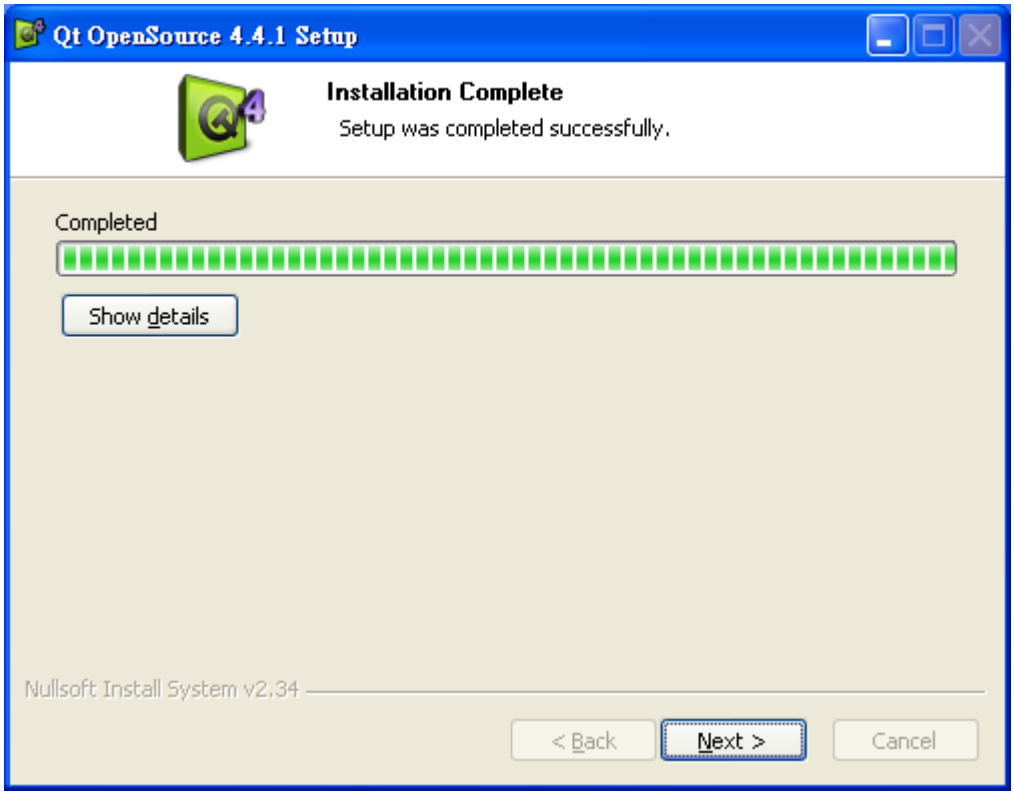


If you get a warring message, just click **Yes** to skip it:

Here's the dialog that is displayed while the files are installed :



Click **Next**:



Click **Finish** to finalize the installation:

2. Install Qt Eclipse Integration

The Qt Eclipse Integration allows programmers to create, build, debug and run Qt applications from within the Eclipse IDE. Integrations are available for Qt C++ on top of the Eclipse C/C++ Development Tooling (CDT) plug-in.

2.1. Download Qt Eclipse Integration

Visit the main Qt Eclipse Integration download site at:

http://trolltech.com/developer/downloads/qt/eclipse-integration-download

Locate and click the **Download** link as shown here:

Qt Eclipse Integration for C++

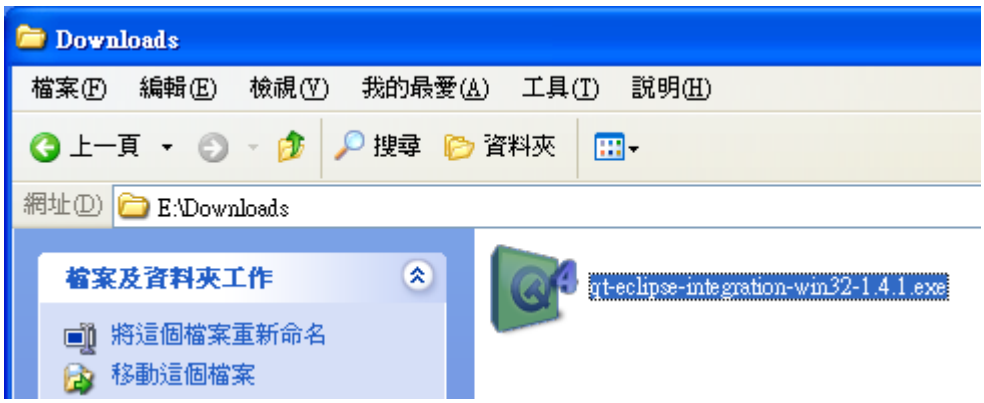
The Eclipse plugin can be used to create programs using any Qt version since 4.1.0.

Downloads

platform	specification	usage guidelines	built with version	download
Windows	Win32	All editions	Qt 4.4.1	Download
Linux (x86 32 bit)	gcc 3.3	All editions	Qt 4.4.1	Download
Linux (x86 32 bit)	gcc 4	All editions	Qt 4.4.1	Download
Linux (x86 64 bit)	gcc 4	All editions	Qt 4.4.1	Download

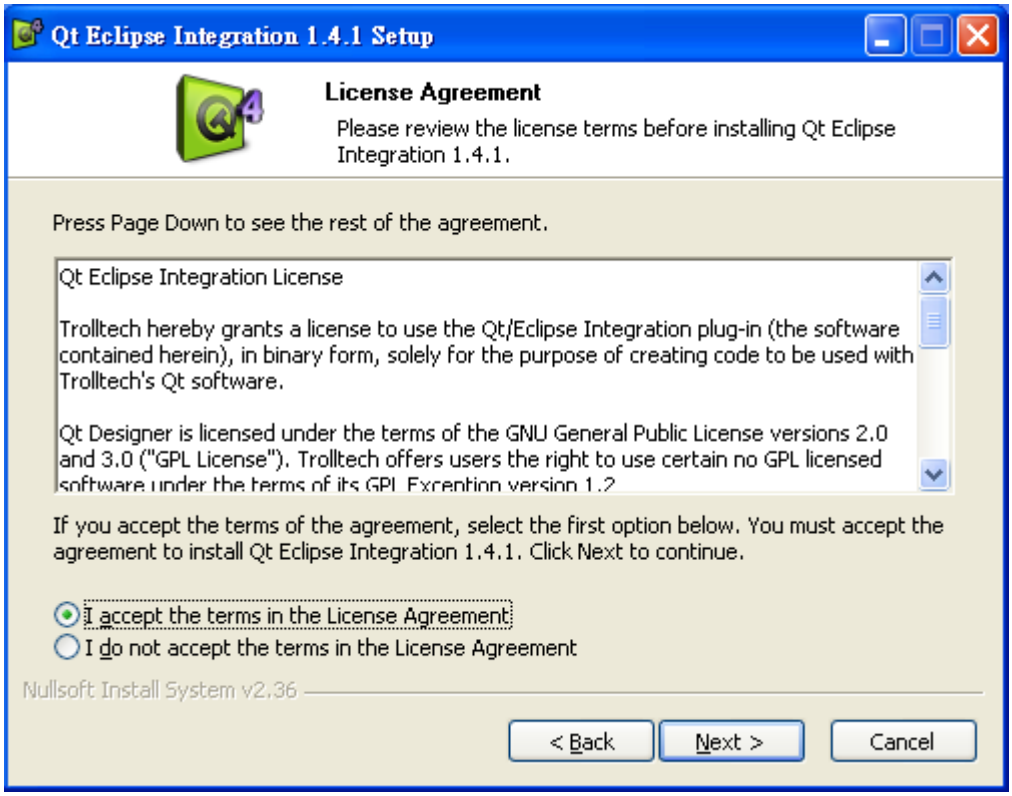
2.2. Install Qt Eclipse Integration

Using Windows Explorer, locate the file you downloaded, and double-click it:

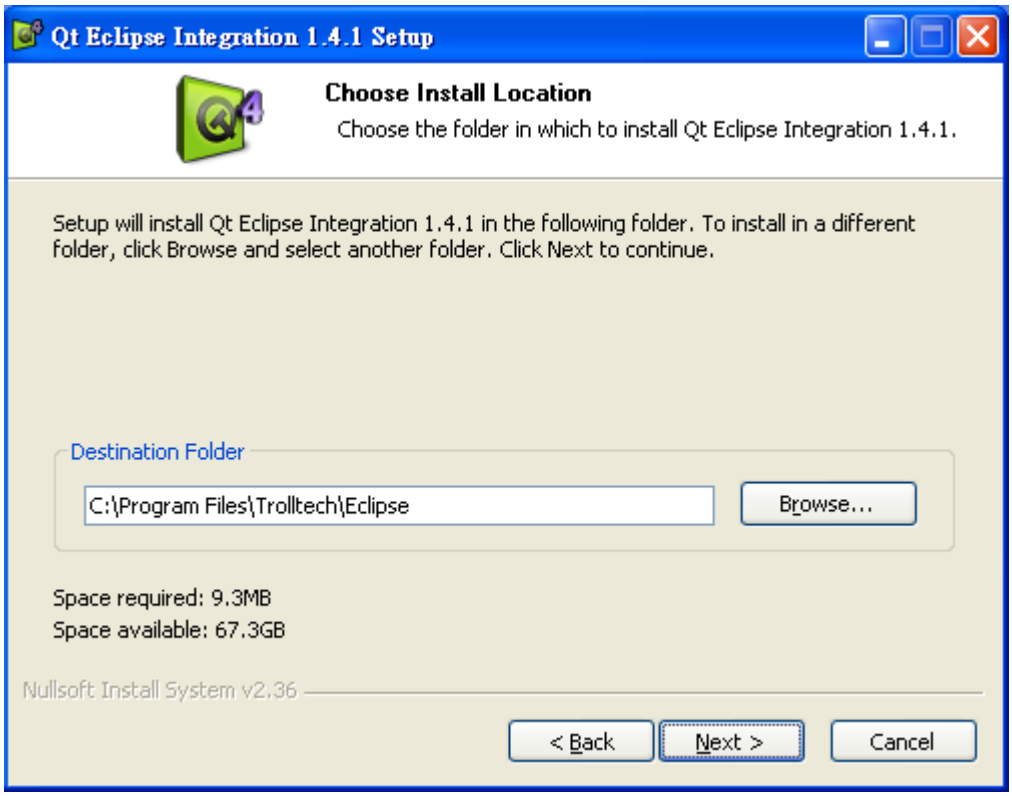


Qt Eclipse Integration uses a regular "InstallShield" type installer. Click **Next** when the introductory screen, shown here, appears:

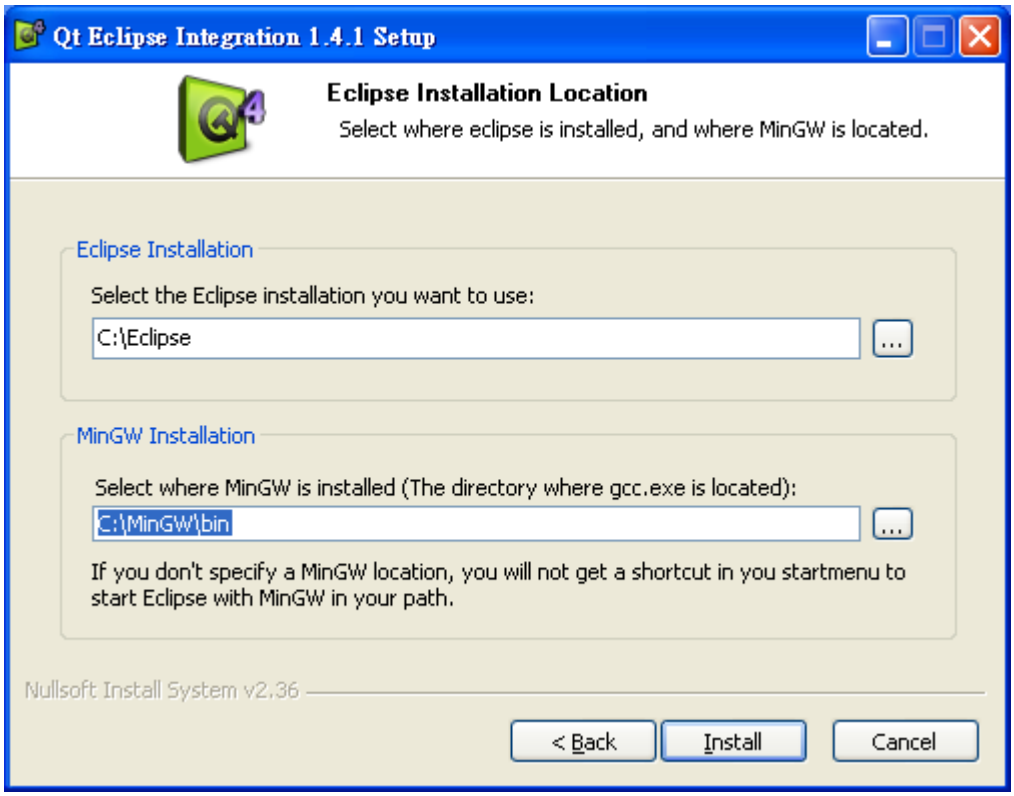
Select “I accept the terms in the License Agreement” and then click **Next** to installation:



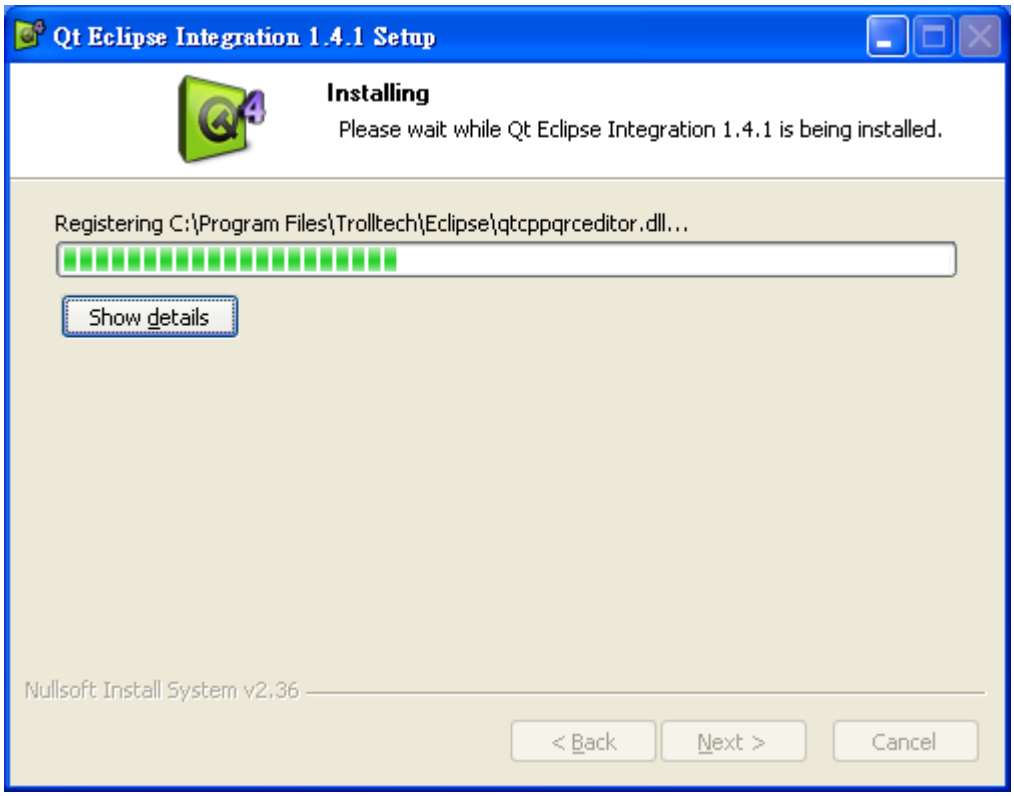
Select a folder where you would like to install Qt Eclipse Integration. Click **Next**:



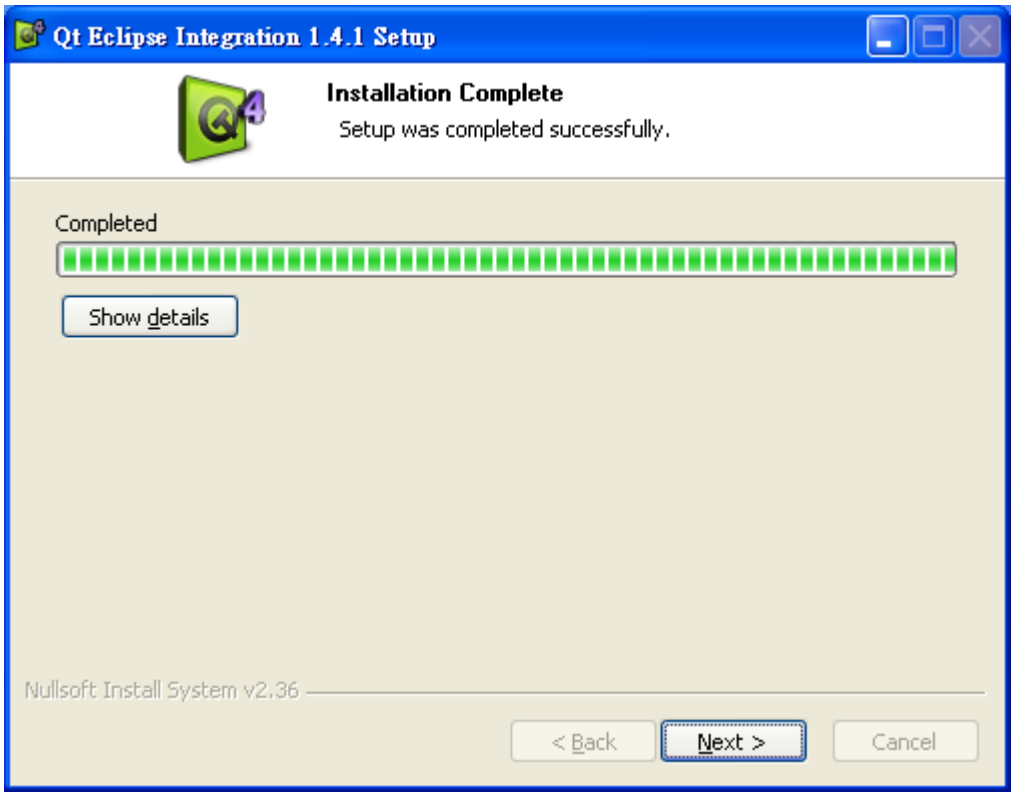
Click **Install** to begin installation:



Here's the dialog that is displayed while the files are installed :



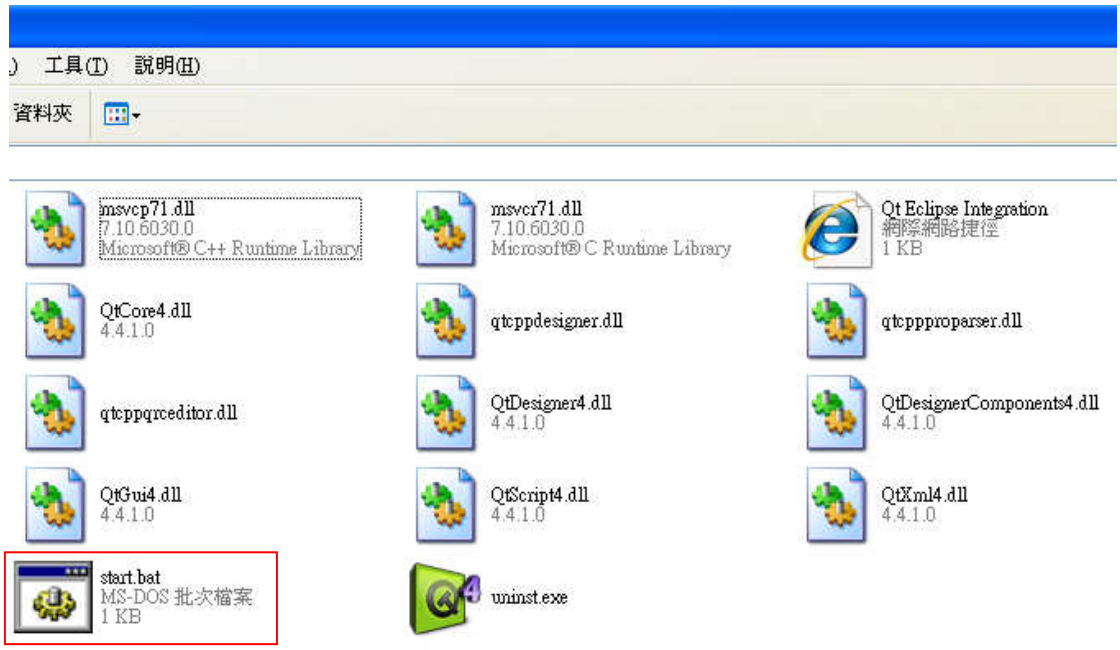
Click **Next**:



Click **Finish** to finalize the installation:

2.3. Configure the Qt Eclipse Integration

Go to the “C:\Program Files\Trolltech\Eclipse” folder, and look for the file name: start.bat

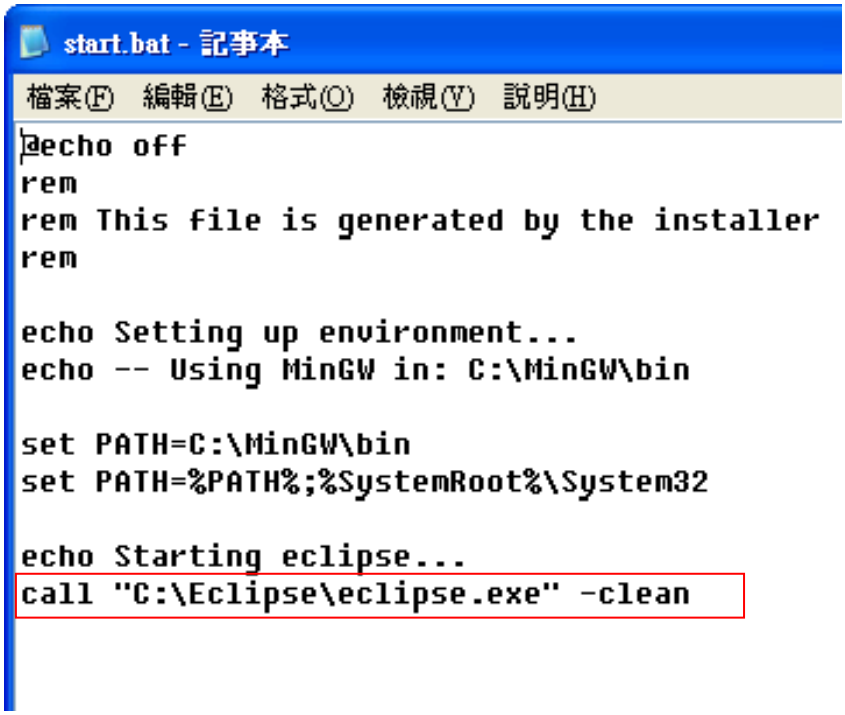


Open the file with notepad (or other editor), and modify the last line from

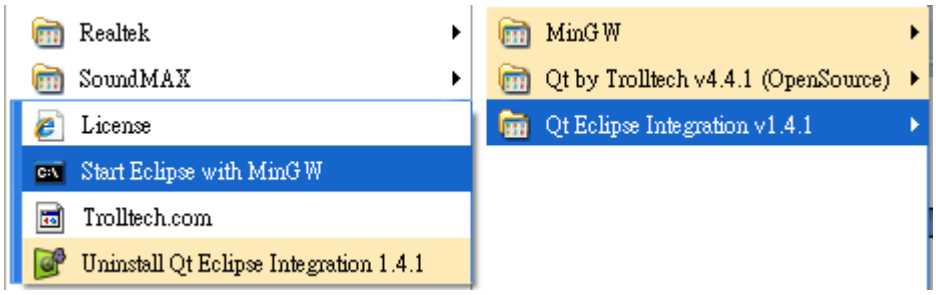
call "C:\Eclipse\eclipse.exe"

to

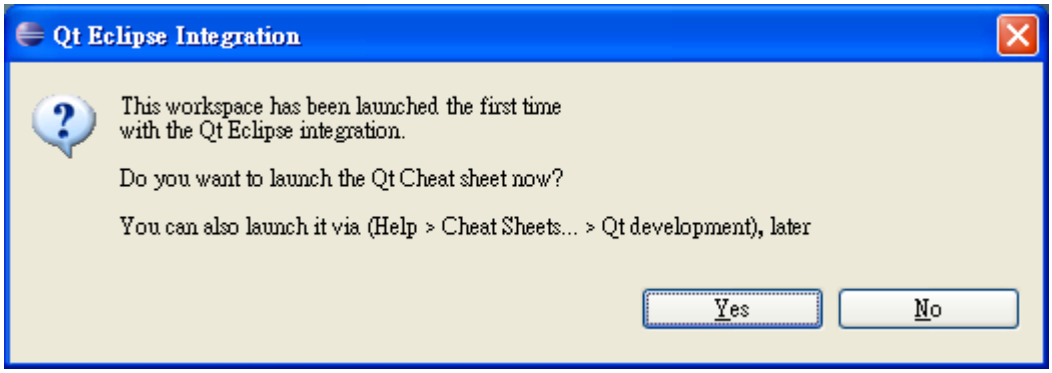
call "C:\Eclipse\eclipse.exe" –clean



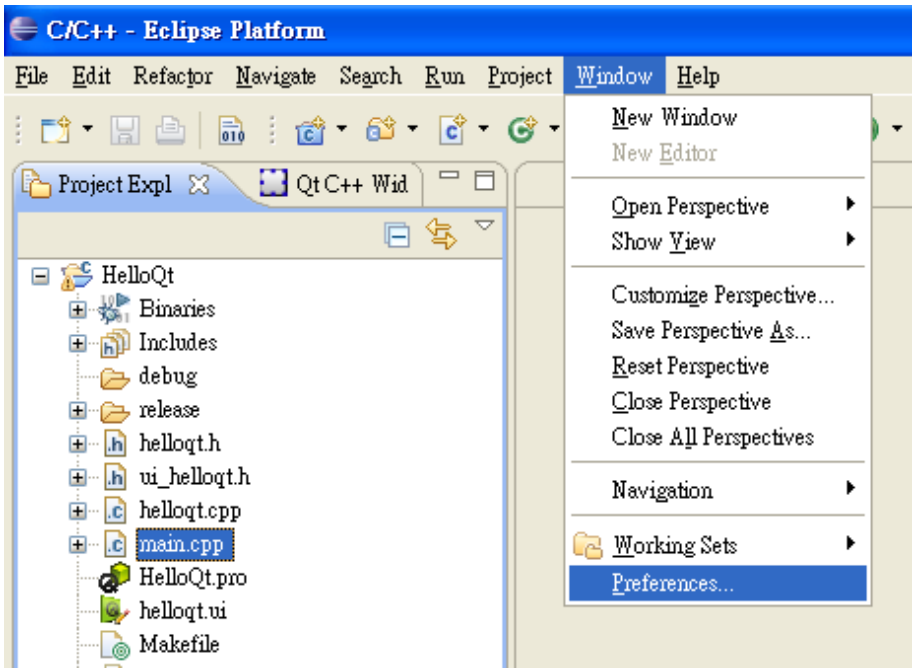
Launch eclipse by going to “All Programs/Qt Eclipse Integration v1.4.1/Start Eclipse with MinGW”.



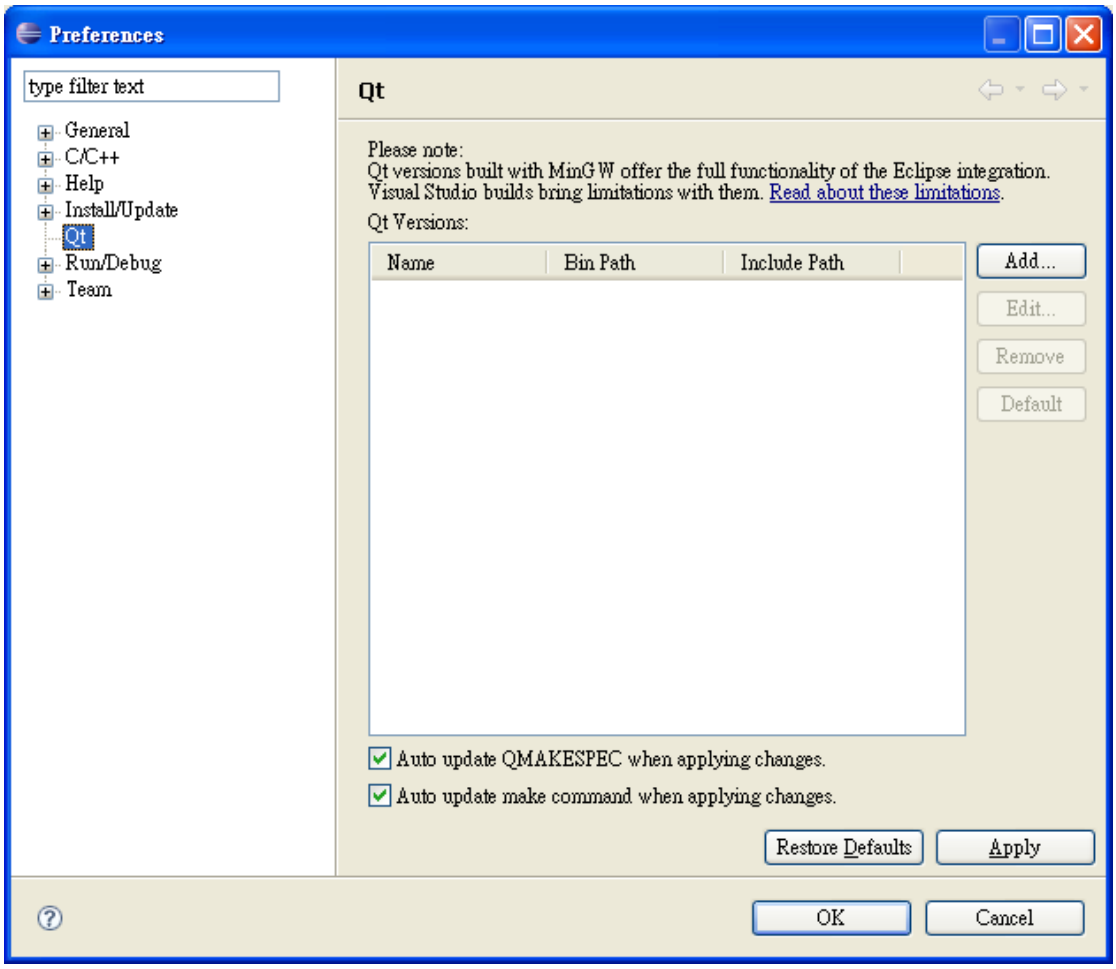
Next, you'll see a Qt Eclipse Integration setup dialog, click **no**:



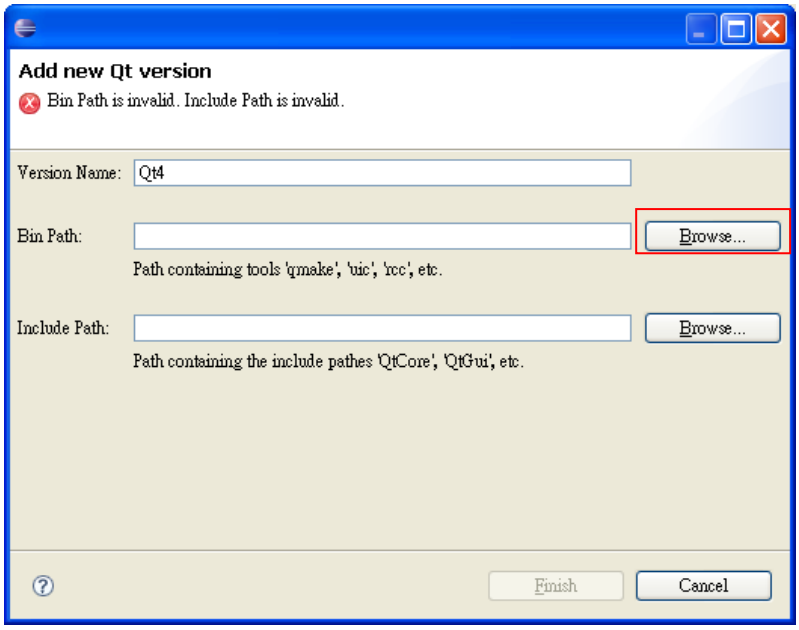
Open the preferences window by going to “Window/Preferences…”.



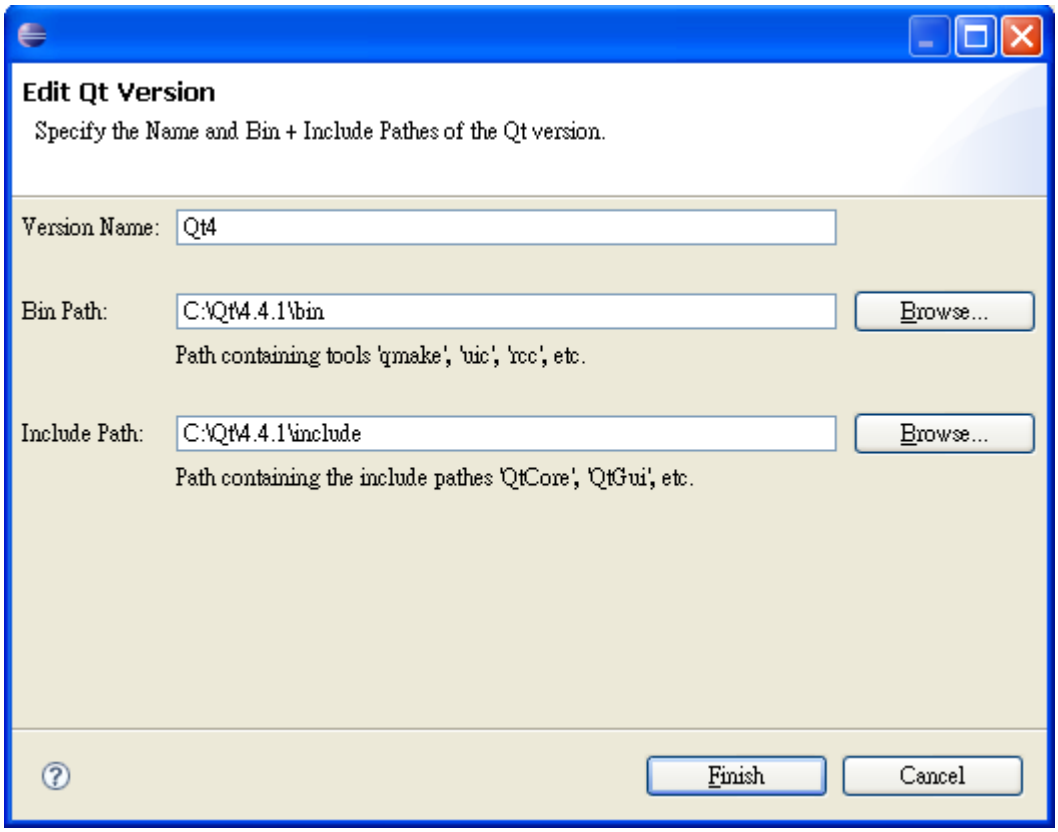
When the dialog box pops up, click on **Qt** and click “**Add…**” button.



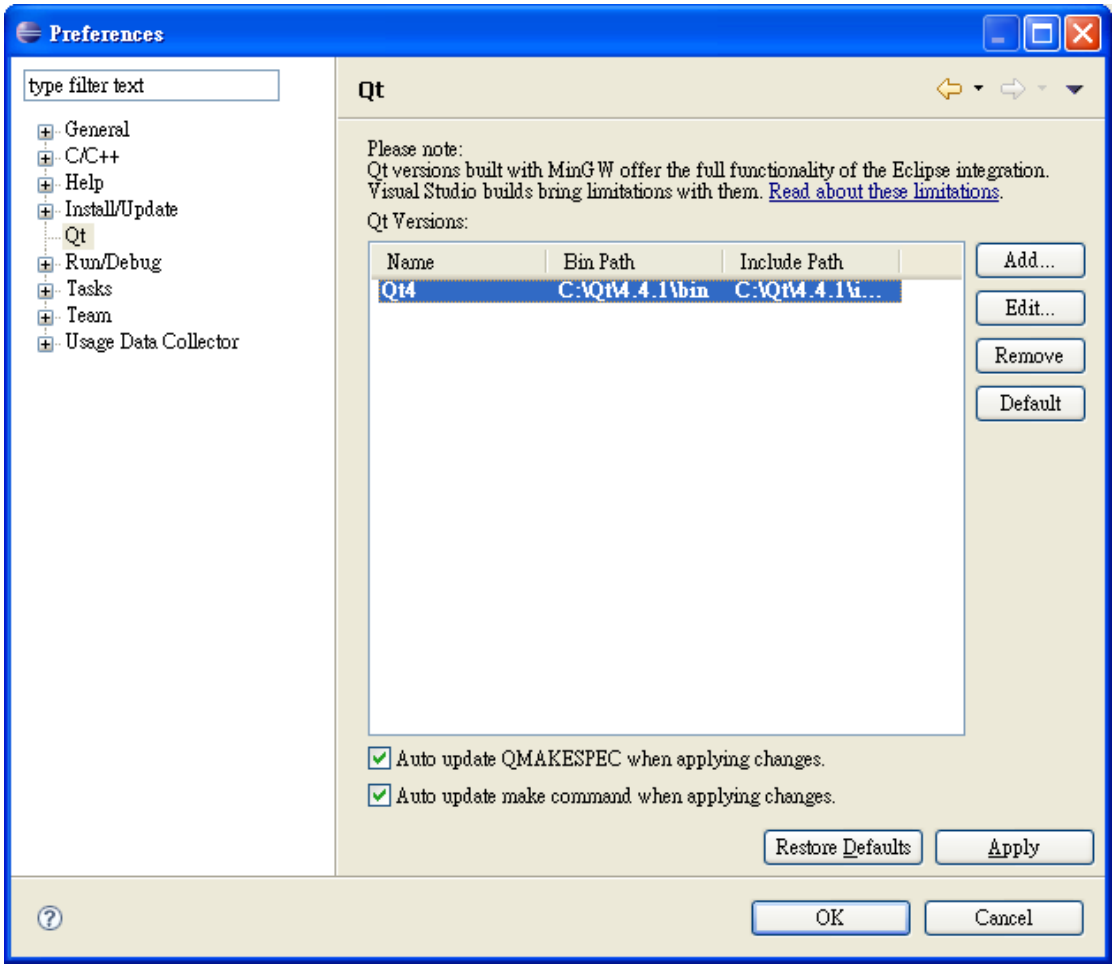
When the Qt setup dialog pops up, type “Qt4” for the **Version Name** and click “Browse…” button.



Browse for a Location of a Qt installation base folder (the default location is C:\Qt\4.4.1\bin) on your system, click **Finish**.



Click **OK**.

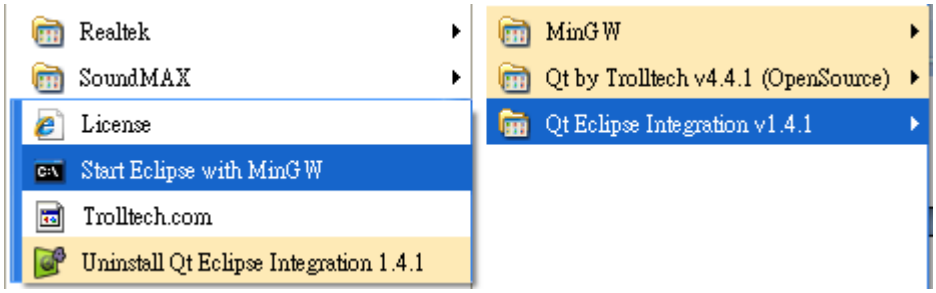

Preferences
type filter text
General
C/C++
Help
Install/Update
Qt
Run/Debug
Tasks
Team
Usage Data Collector
Qt
Please note:
Qt versions built with MinGW offer the full functionality of the Eclipse integration.
Visual Studio builds bring limitations with them. Read about these limitations.
Qt Versions:
Name
Bin Path
Include Path
Qt4
C:\Qt\4.4.1\bin
C:\Qt\4.4.1\i...
Add...
Edit...
Remove
Default
Auto update QMAKESPEC when applying changes.
Auto update make command when applying changes.
Restore Defaults
Apply
OK
Cancel

3. Compile Qt Test Runner library

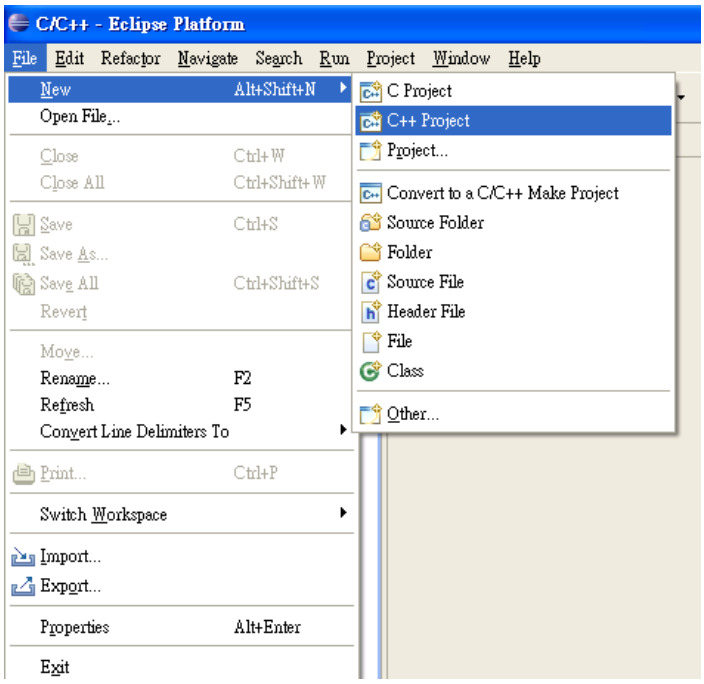
Download the Qt Test Runner source code (QtTestRunner_Qt4.zip) from the following link.

http://cppunit.sourceforge.net/cppunit-wiki/QtTestRunnerWithEclipse?action=AttachFile&do=get&target=QtTestRunner_Qt4.zip

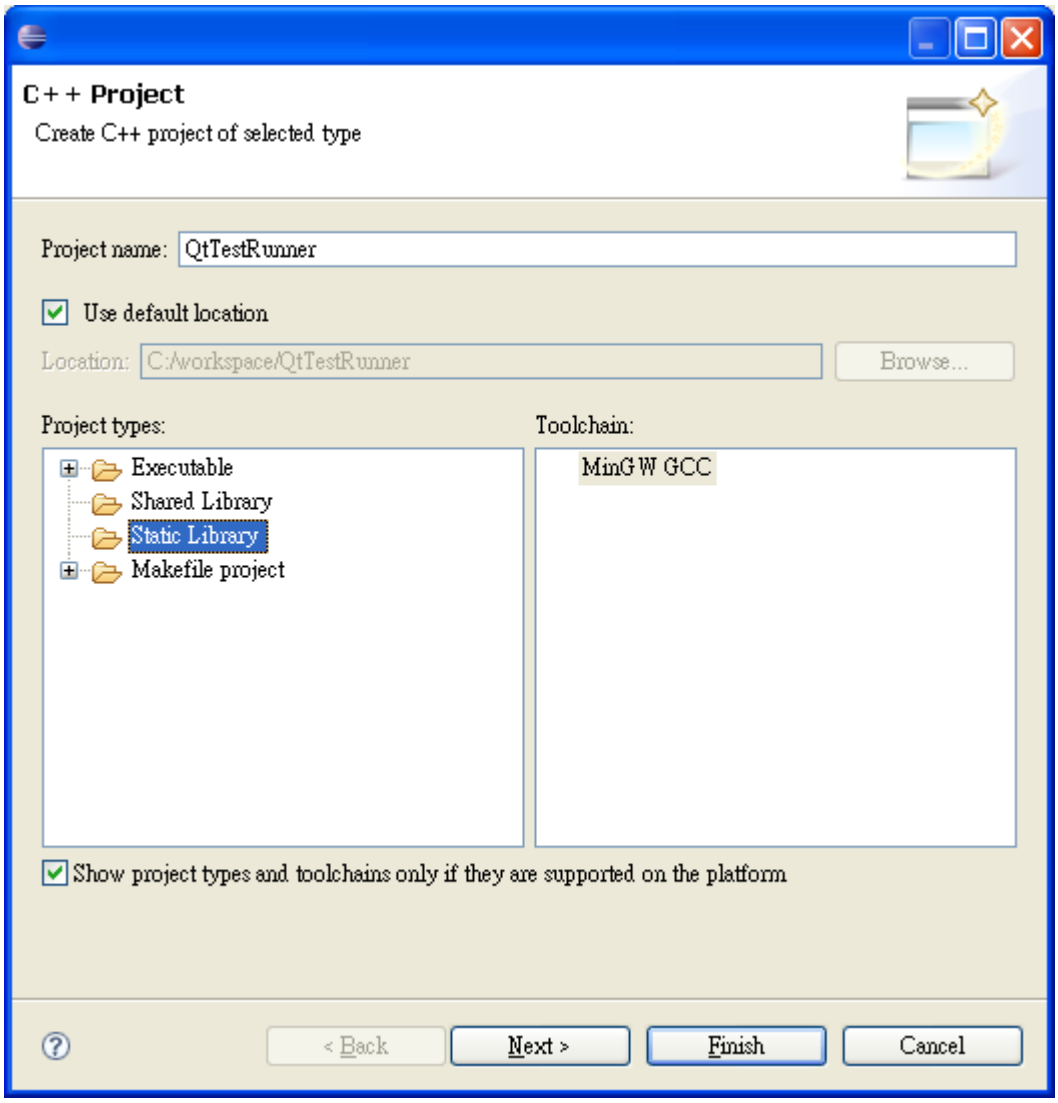
Launch eclipse by going to “All Programs/Qt Eclipse Integration v1.4.1/Start Eclipse with MinGW”.



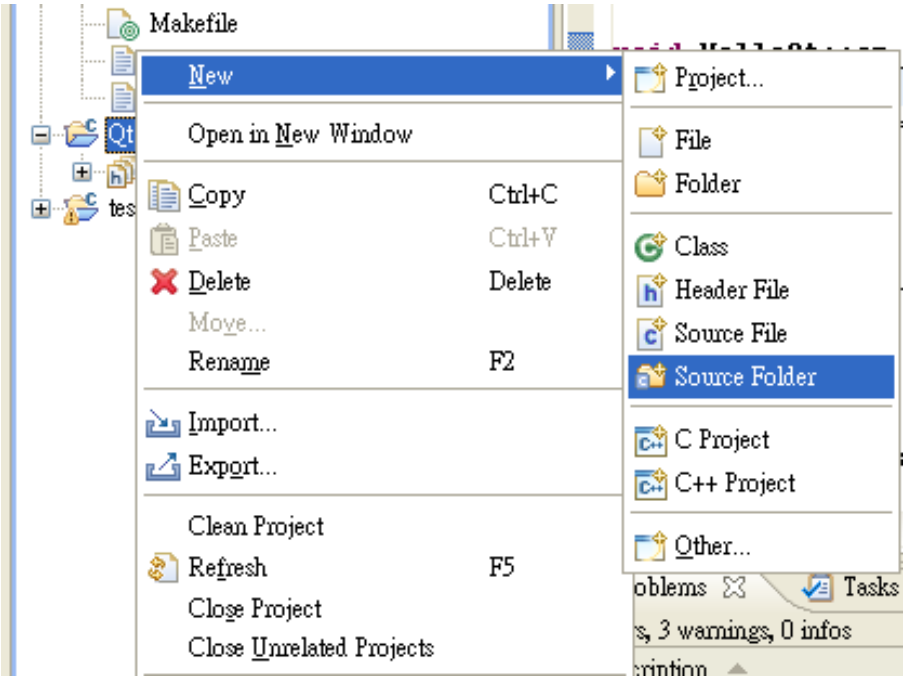
Create a new C++ project by going to “File / New / C++ Project”.



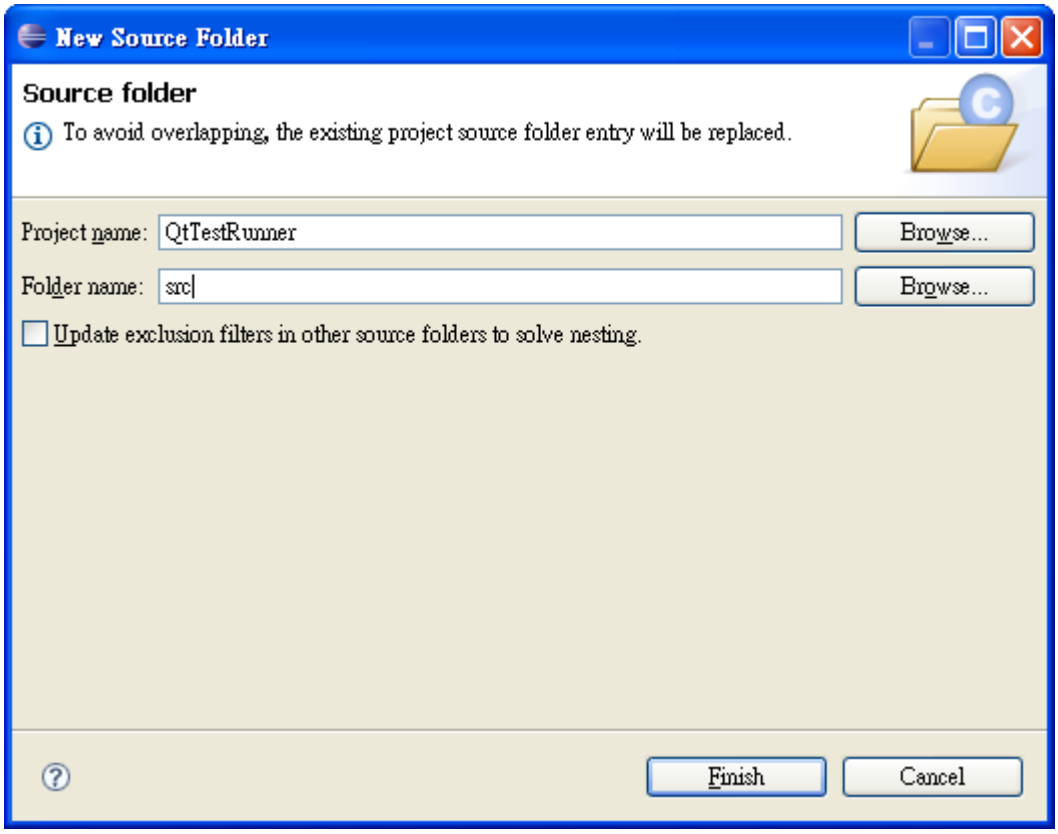
When the dialog box pops up, click on **MinGW GCC** and select **Static Library**. Enter **QtTestRunner** for the name of the project. Click **Finish** to create c++ project.



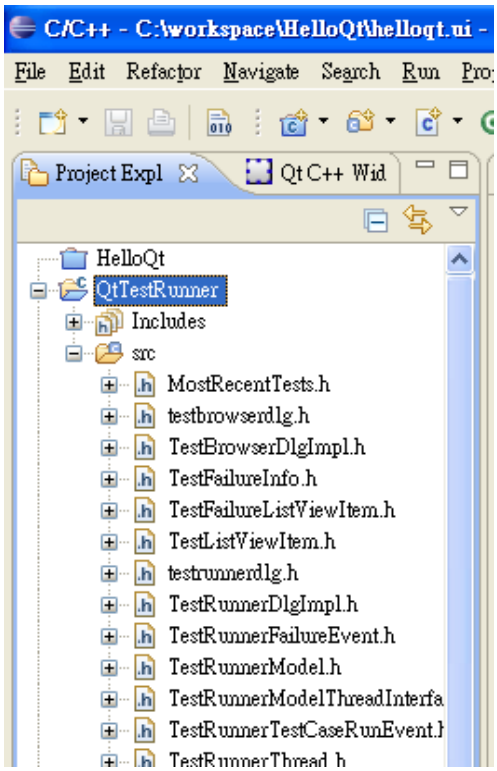
You can see the QtTestRunner project you just created. Right-Click on the project and create a new source folder by going to “New/Source Folder”.



Enter **src** for the name of the Folder name. Click **Finish**.



Copy all of *.cpp from QtTestRunner_Qt4.zip to the **src** folder.



Comment the following text from the moc_*.cpp

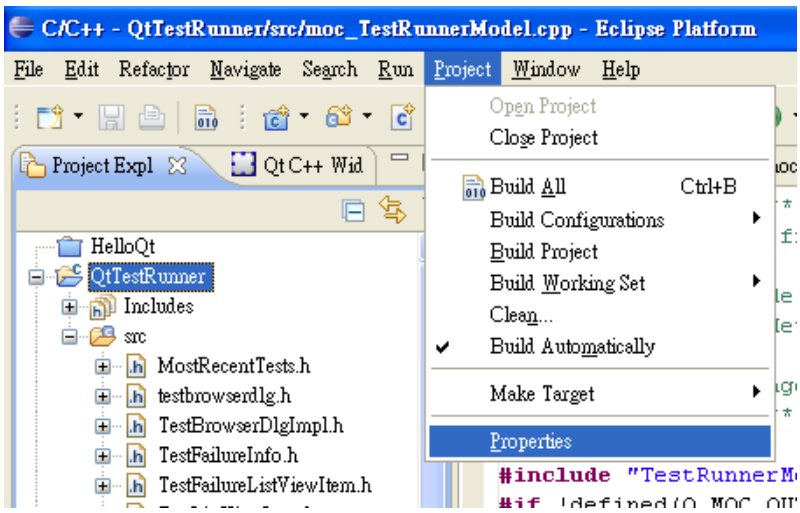
```
#elif Q_MOC_OUTPUT_REVISION != 58
#error "This file was generated using the moc from 4.0.1. It"
#error "cannot be used with the include files from this version of Qt."

#error "(The moc has changed too much.)"
```

```
*moc_MostRecentTests   moc_testbrowserdlg.c   moc_testrunnerdlg.cp   moc_TestRunnerModel.

/*****************************************************************************
** Meta object code from reading C++ file 'MostRecentTests.h'
**
** Created: qui 1. dez 16:23:46 2005
**      by: The Qt Meta Object Compiler version 58 (Qt 4.0.1)
**
** WARNING! All changes made in this file will be lost!
*****************************************************************************/

#include "MostRecentTests.h"
#if !defined(Q_MOC_OUTPUT_REVISION)
#error "The header file 'MostRecentTests.h' doesn't include <QObject>."
//#elif Q_MOC_OUTPUT_REVISION != 58
//#error "This file was generated using the moc from 4.0.1. It"
//#error "cannot be used with the include files from this version of Qt."
//#error "(The moc has changed too much.)"
#endif

static const uint qt_meta_data_MostRecentTests[] = {

 // content:
       1,       // revision
       0,       // classname
       0,    0, // classinfo
       3,   10, // methods
       0,    0, // properties
       0,    0, // enums/sets

 // signals: signature, parameters, type, tag, flags
```

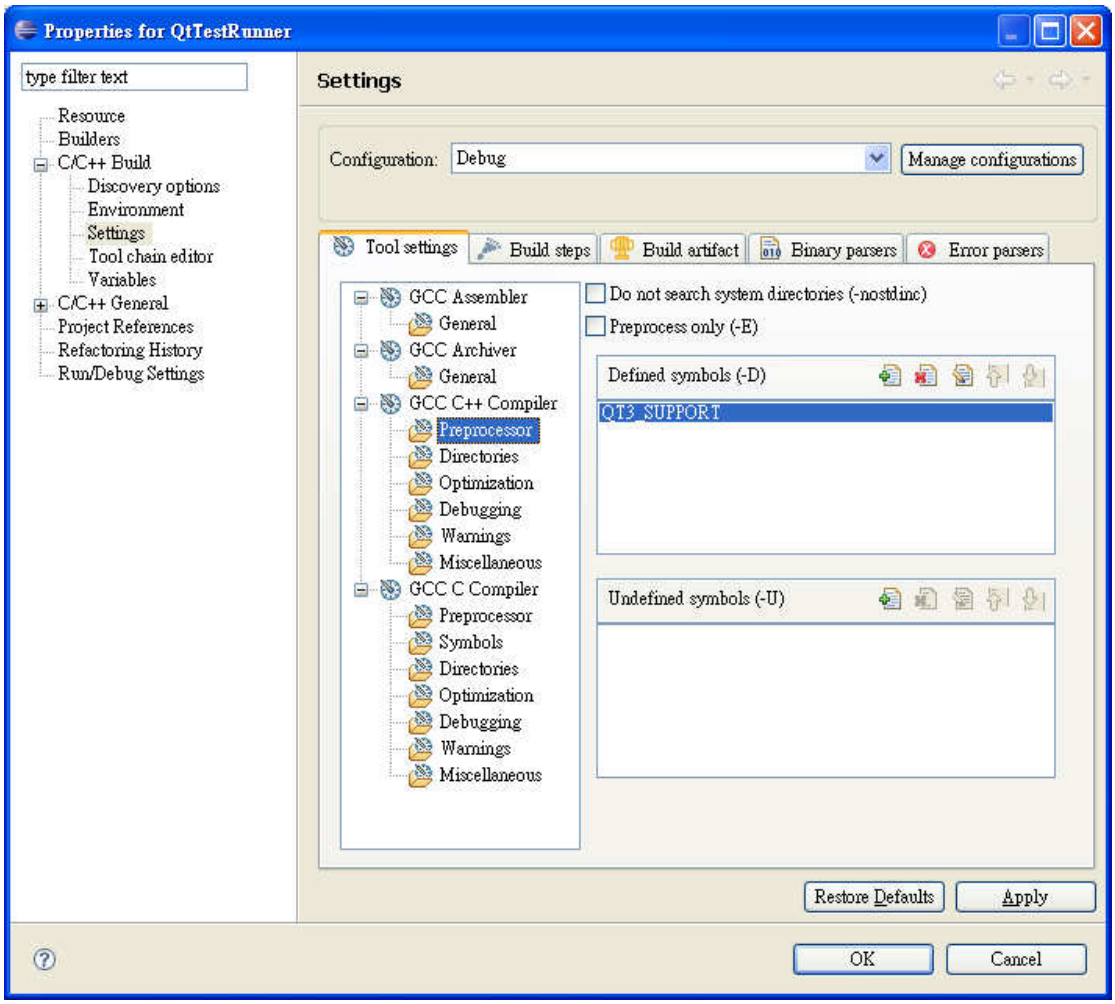
Add a new line “#include <string>” to the c:\cppunit\include\cppunit\TestResult.h, as shown below:

```
UltraEdit-32 - [C:\cppunit\include\cppunit\TestResult.h]
檔案(F) 編輯(E) 搜尋(S) 專案(P) 檢視(V) 格式(T) 區塊(L) 巨集(M) 進階(A) 視
TestResult.h

過濾:        刷新(R)
開啓檔案
開啓檔案
  C:\cppunit\include\cppunit\T

 1 #ifndef CPPUNIT_TESTRESULT_H
 2 #define CPPUNIT_TESTRESULT_H
 3
 4 #include <cppunit/Portability.h>
 5 #include <string>
 6
 7 #if CPPUNIT_NEED_DLL_DECL
 8 #pragma warning( push )
 9 #pragma warning( disable: 4251 )
10 #endif
11
12 #include <cppunit/SynchronizedObj
13 #include <cppunit/portability/Cpp
14
15 CPPUNIT_NS_BEGIN
16
```

Click on the project and modify the project properties by going to “Project/Properties”

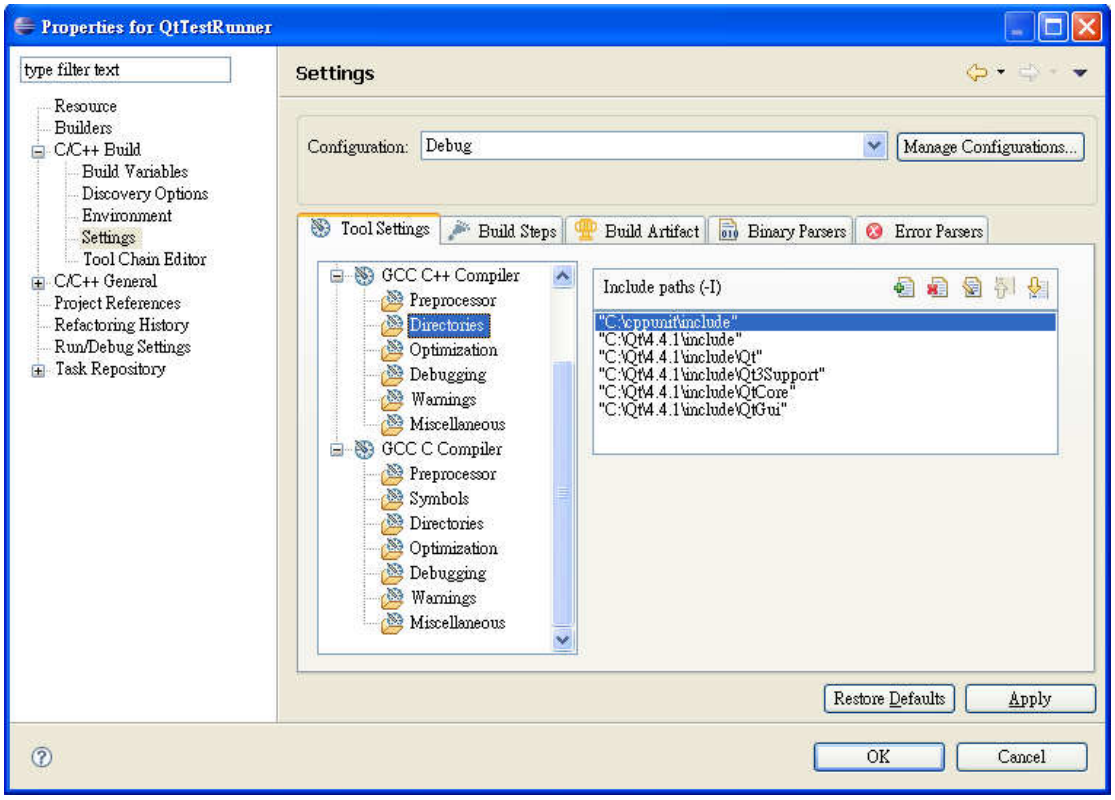


Define the variable QT3_SUPPORT

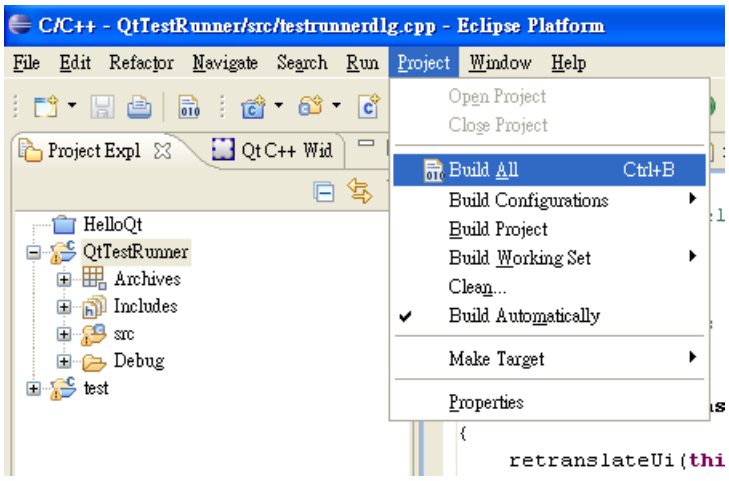


Add the following directories to the include path: ($CppUnit and $Qt refer to your top level installation directories of CppUnit and Qt)

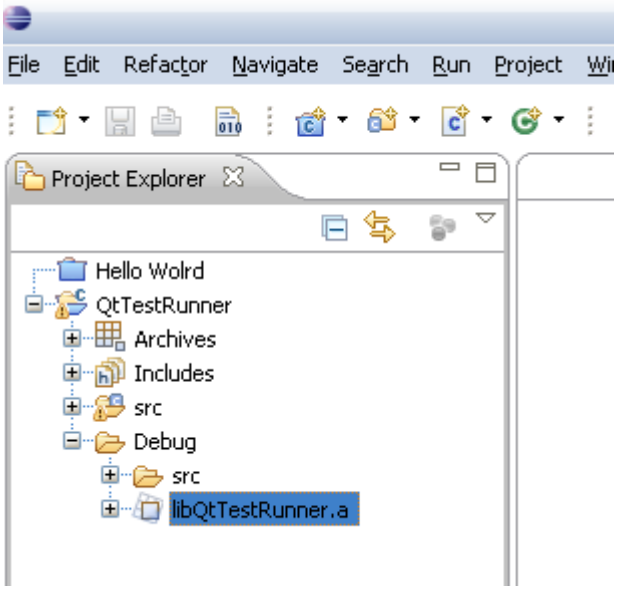
- $CppUnit/include
- $Qt/include
- $Qt/include/Qt3Support
- $Qt/include/QtCore
- $Qt/include/QtGui



Build the project (Ctrl + B)



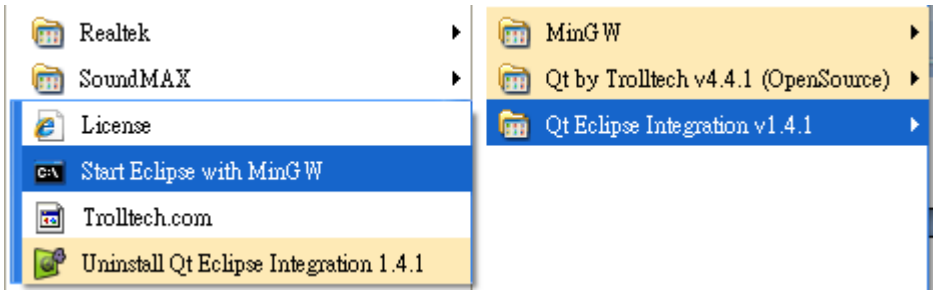
Copy the libQtTestRunner.a file from “C:\workspace\QtTestRunner\Debug\” to “c:\cppunit\lib\”


File Edit Refactor Navigate Search Run Project
Project Explorer
Hello Wolrd
QtTestRunner
Archives
Includes
src
Debug
src
libQtTestRunner.a

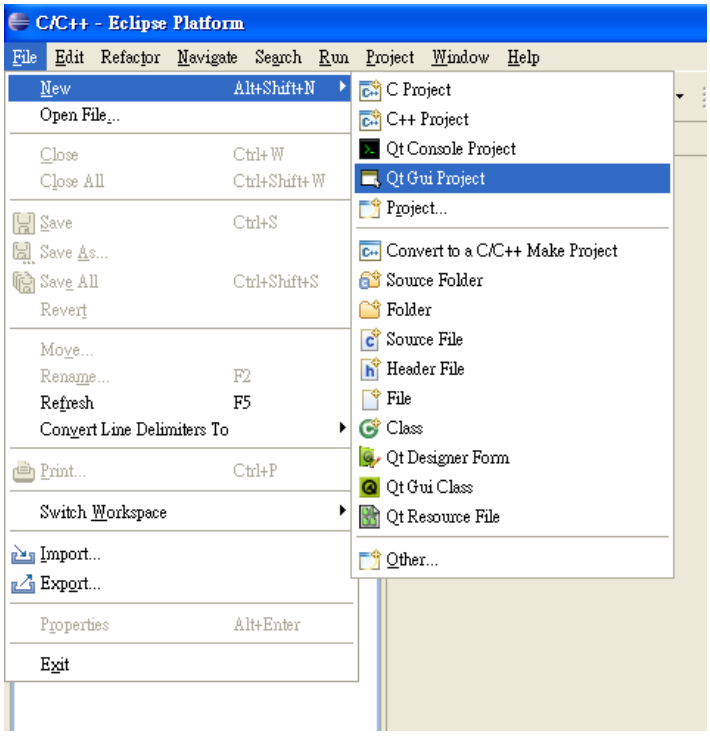

lib
檔案(F) 編輯(E) 檢視(V) 我的最愛(A) 工具(T) 說明(H)
上一頁
搜尋
資料夾
網址(D)
C:\cppunit\lib
檔案及資料夾工作
建立新的資料夾
將這個資料夾發佈到網站
共用這個資料夾
libQTTestRunner.a
A 檔案
10,804 KB

4. Test Qt develop environment

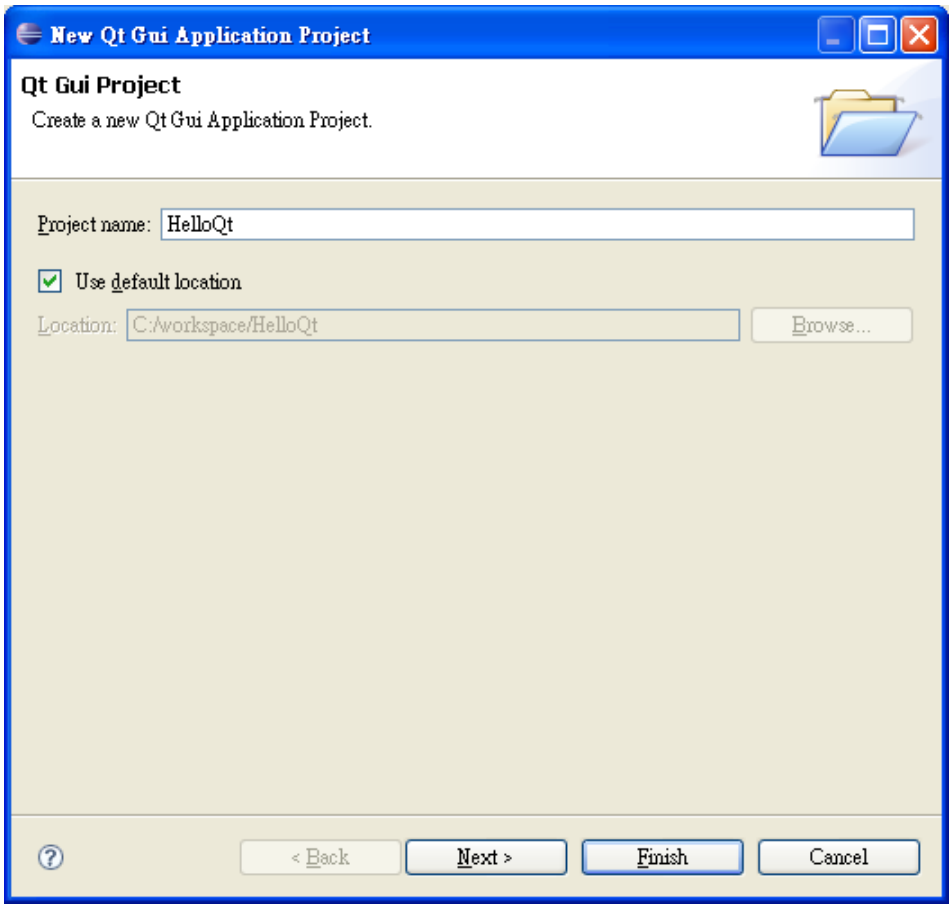
Launch eclipse by going to “All Programs/Qt Eclipse Integration v1.4.1/Start Eclipse with MinGW”.



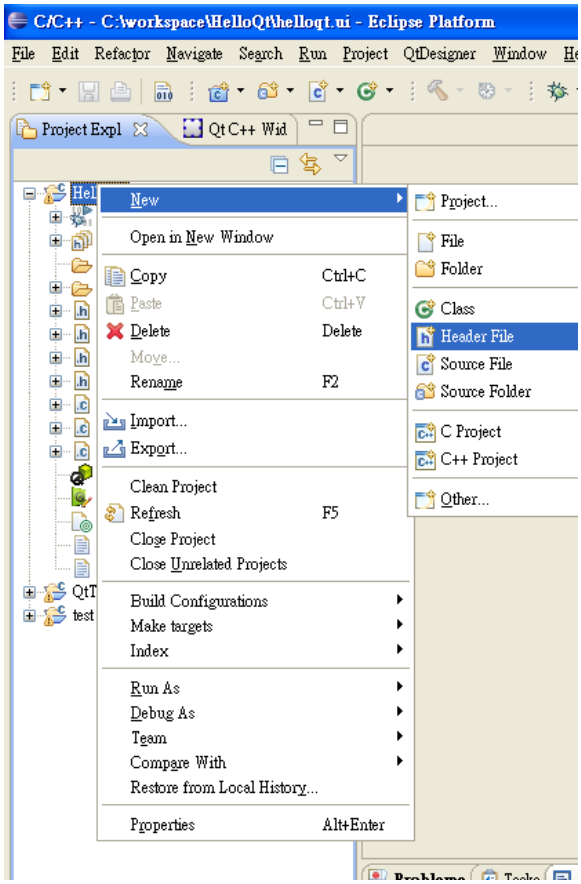
Create a new C++ project by going to “File / New / Qt Gui Project”.



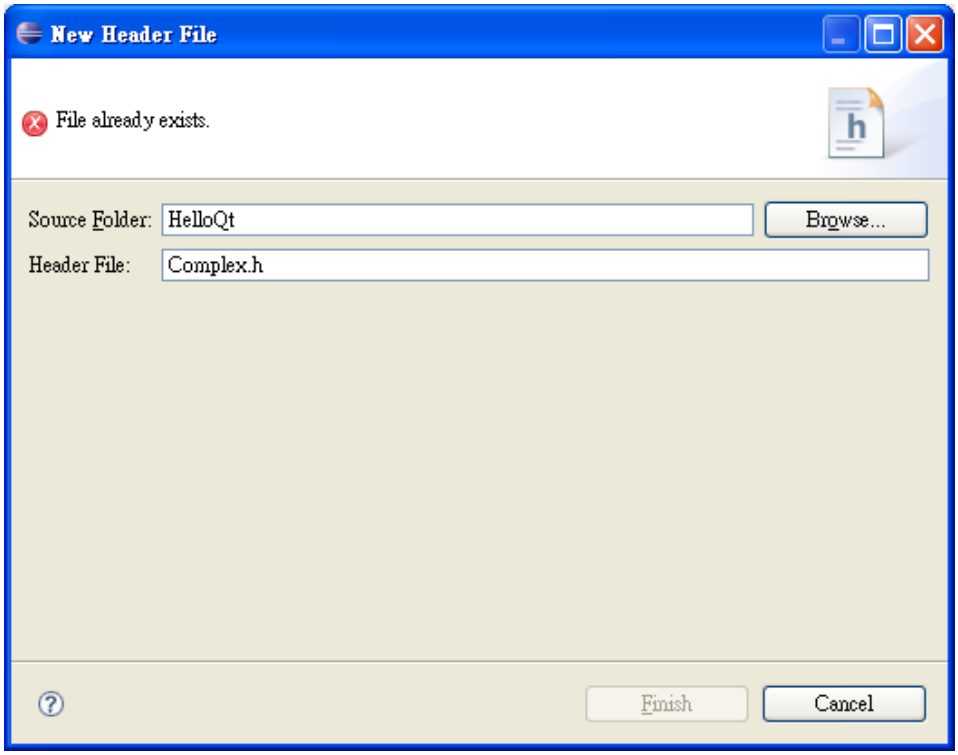
Enter **HelloQt** for the name of the project. Click **Finish** to create Qt project.



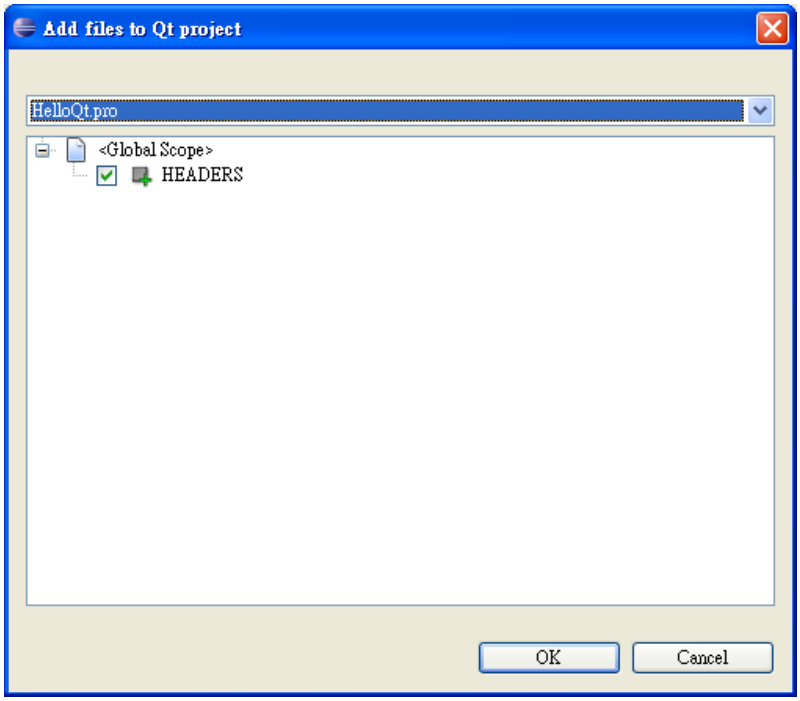
Right-Click on the project and create a new source file by going to “New/Header File”.



Enter “Complex.h” for the file name and click **Finish**.



Click **OK.**

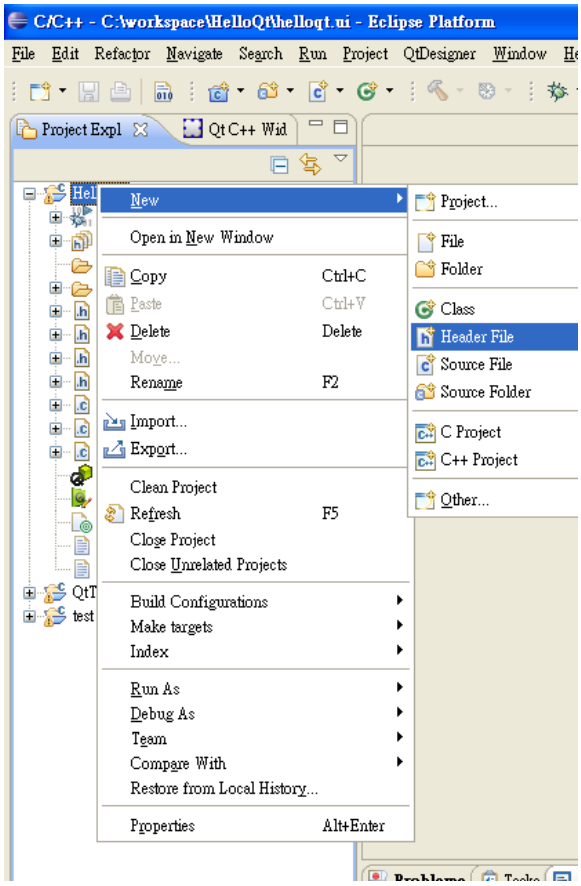


Once the file is created, copy and pastes the following code into **Complex.h.**
Press **Ctrl + S** to save the file:

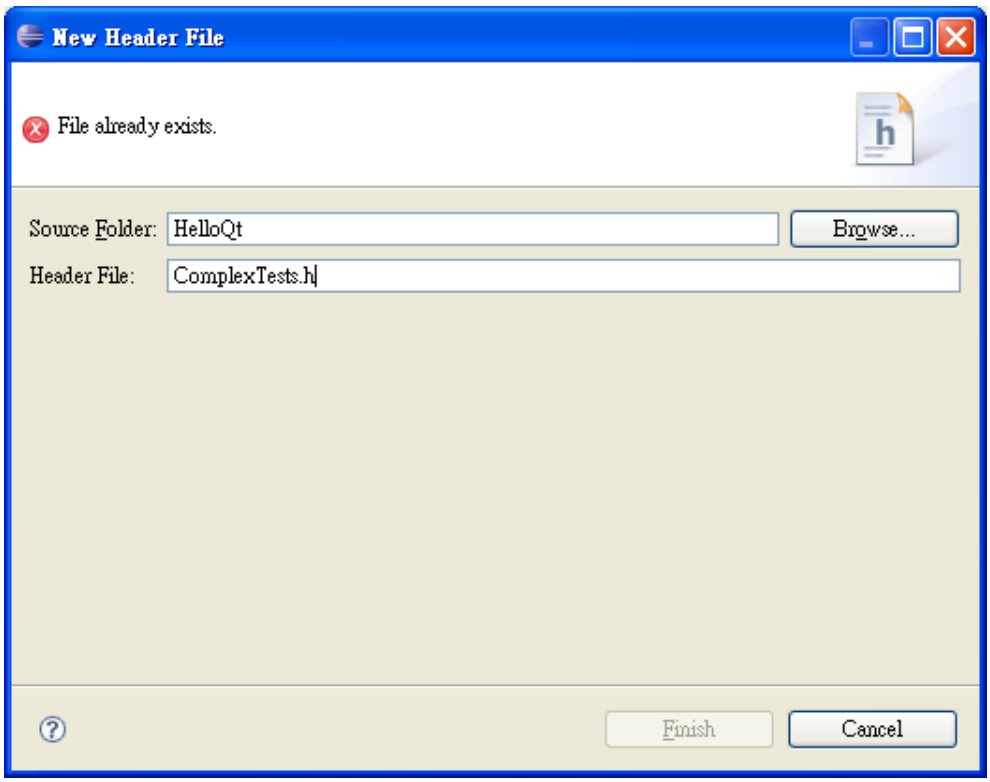
```
#ifndef COMPLEX_H_
#define COMPLEX_H_

class Complex {

  friend bool operator ==(const Complex& a, const Complex& b);
  friend const Complex operator +(const Complex& a, const Complex& b);

  double real, imaginary;

public:

  Complex(double r, double i = 0) : real(r), imaginary(i) {}

};

bool operator ==(const Complex &a, const Complex &b)
{
  return (a.real == b.real) && (a.imaginary == b.imaginary);
```

```
}

const Complex operator +(const Complex &a, const Complex &b)
{
  return Complex(a.real + b.real, a.imaginary + b.imaginary);
}

#endif /*COMPLEX_H_*/
```

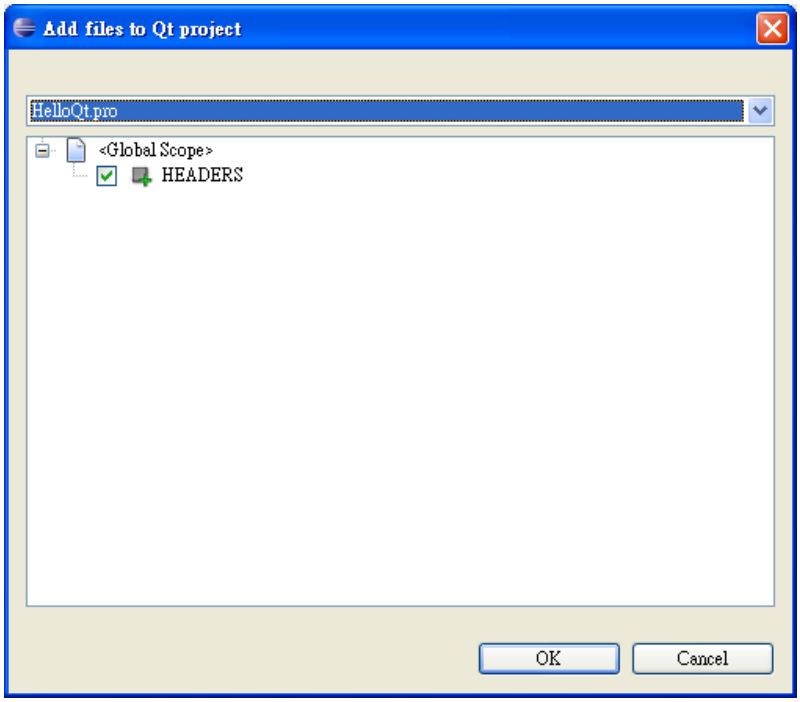
Right-Click on the project and create a new source file by going to “New/Header File”.



Enter “ComplexTests.h” for the file name and click **Finish**.



Click **OK.**



Once the file is created, copy and pastes the following code into **ComplexTests.h.**

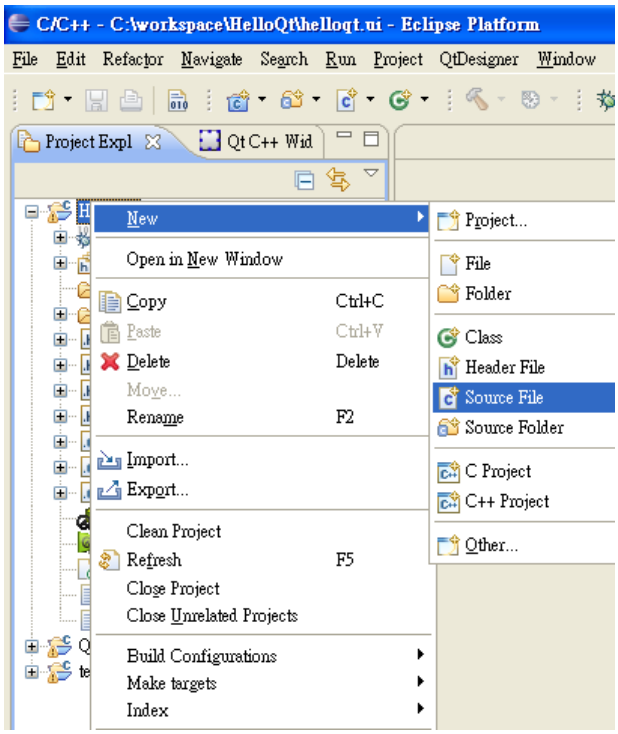
Press **Ctrl + S** to save the file:

```
#ifndef COMPLEXTESTS_H_
```

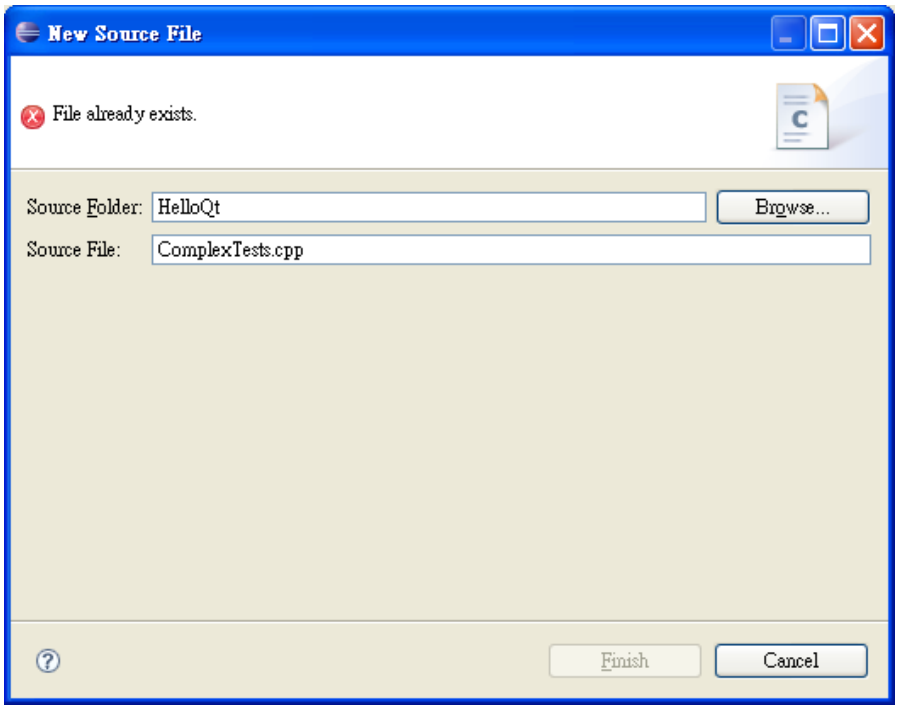
```
#define COMPLEXTESTS_H_

#include "Complex.h"
#include <cppunit/extensions/HelperMacros.h>

class ComplexTests : public CppUnit::TestFixture  {

private:

    Complex *m_10_1, *m_1_1, *m_11_2;

    CPPUNIT_TEST_SUITE(ComplexTests);
    CPPUNIT_TEST(testEquality);
    CPPUNIT_TEST(testAddition);
    CPPUNIT_TEST_SUITE_END();

public:

    void setUp() {
        m_10_1 = new Complex(10, 1);
        m_1_1 = new Complex(1, 1);
        m_11_2 = new Complex(11, 2);
    }

    void tearDown() {
        delete m_10_1;
        delete m_1_1;
        delete m_11_2;
    }

    void testEquality() {
        CPPUNIT_ASSERT( *m_10_1 == *m_10_1 );
        CPPUNIT_ASSERT( !(*m_10_1 == *m_11_2) );
    }

    void testAddition() {
        CPPUNIT_ASSERT( *m_10_1 + *m_1_1 == *m_11_2 );
    }
```

```
};

#endif  // COMPLEXTESTS_H_
```

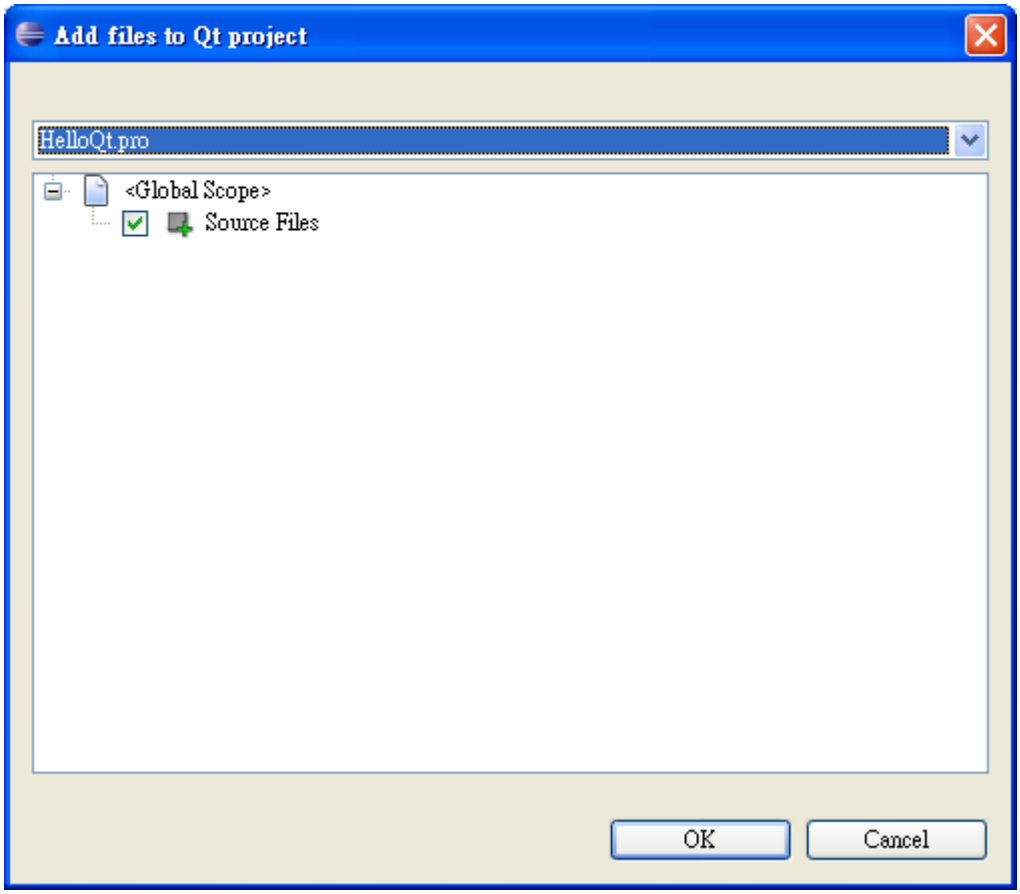
Right-Click on the project and create a new source file by going to “New/Source File”.



Enter **ComplexTests.cpp** as the name of the file. Click **Finish** to continue.



Click **OK.**

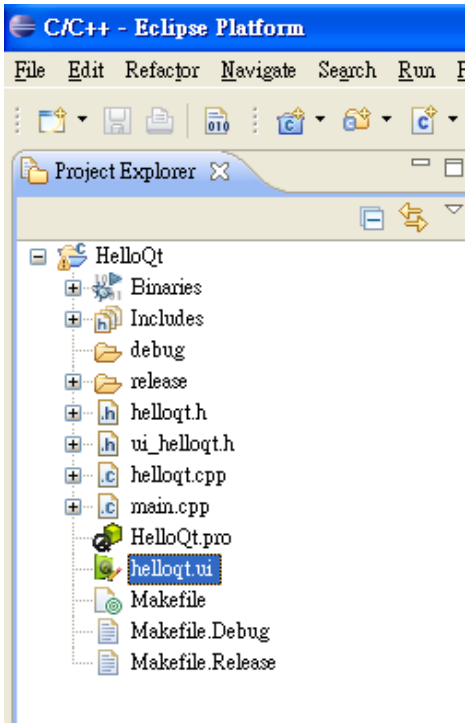


Once the file is created, copy and pastes the following code into **ComplexTests.cpp.**

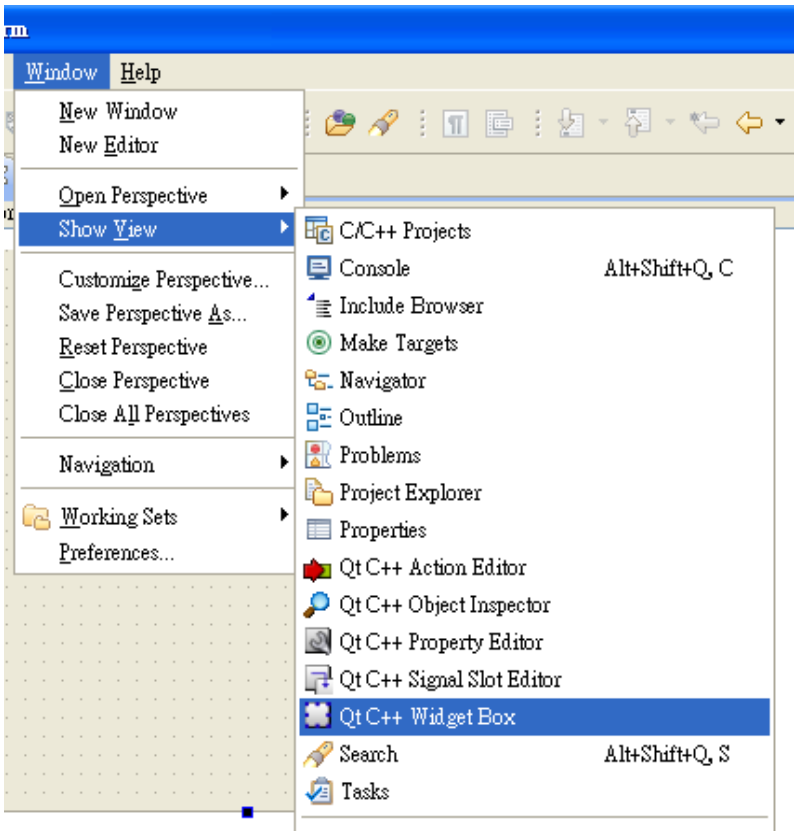
Press **Ctrl + S** to save the file:

```
#include "ComplexTests.h"


CPPUNIT_TEST_SUITE_REGISTRATION(ComplexTests);
```

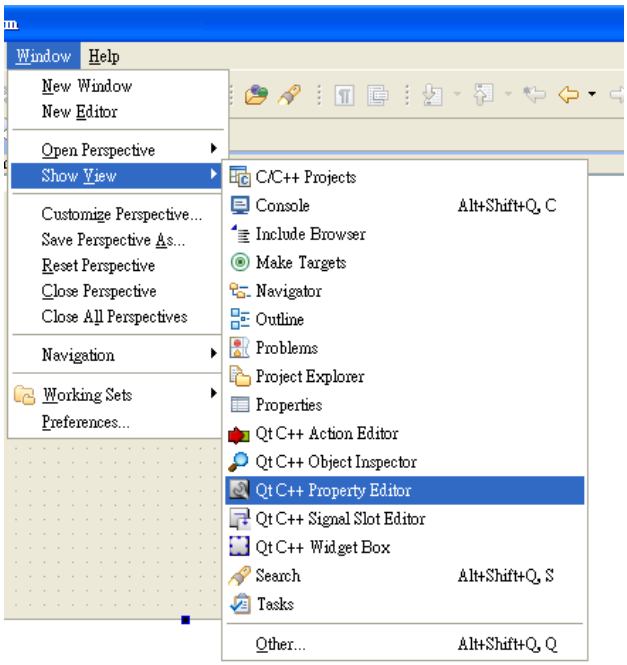
Double click “helloqt.ui” in Eclipse’s Project Explorer for designing GUI layout.



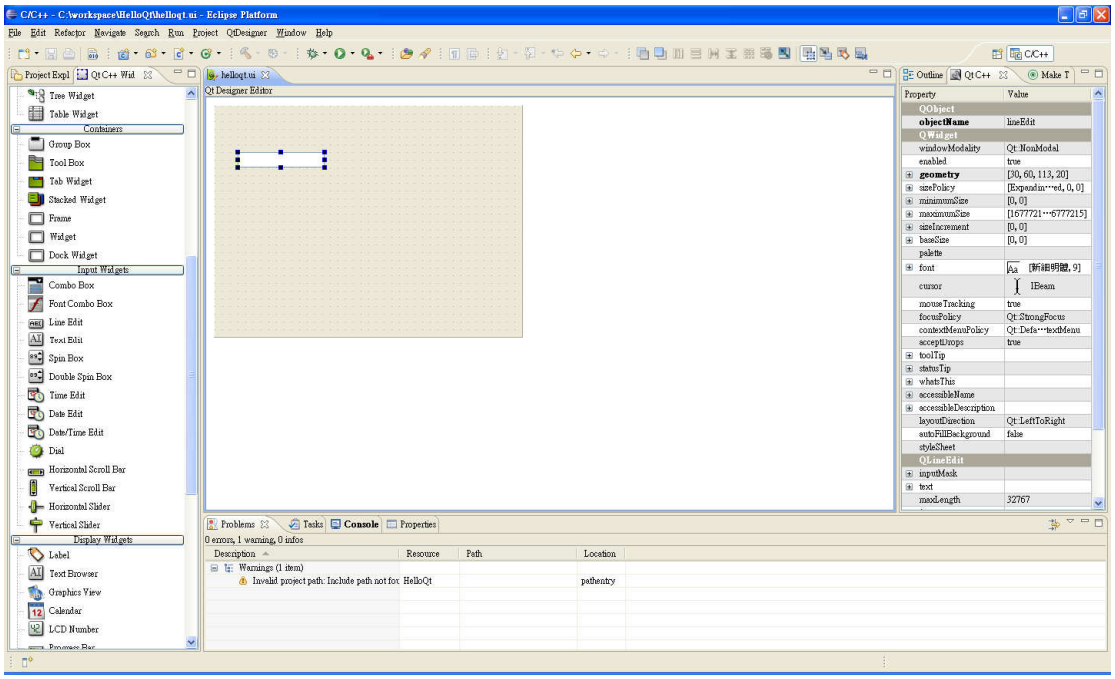
Open Qt C++ Widget Box by going on “Window/Show View/Qt C++ Widget Box”.



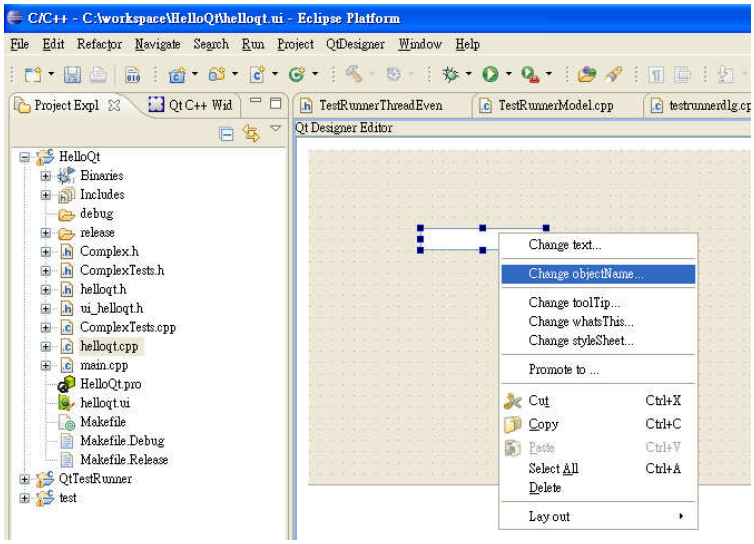
Open Qt C++ Property Editor by going on “Window/ Show View/Qt C++ Property Editor”.



Drag and drop the “Line Edit” widget into the hello.ui.

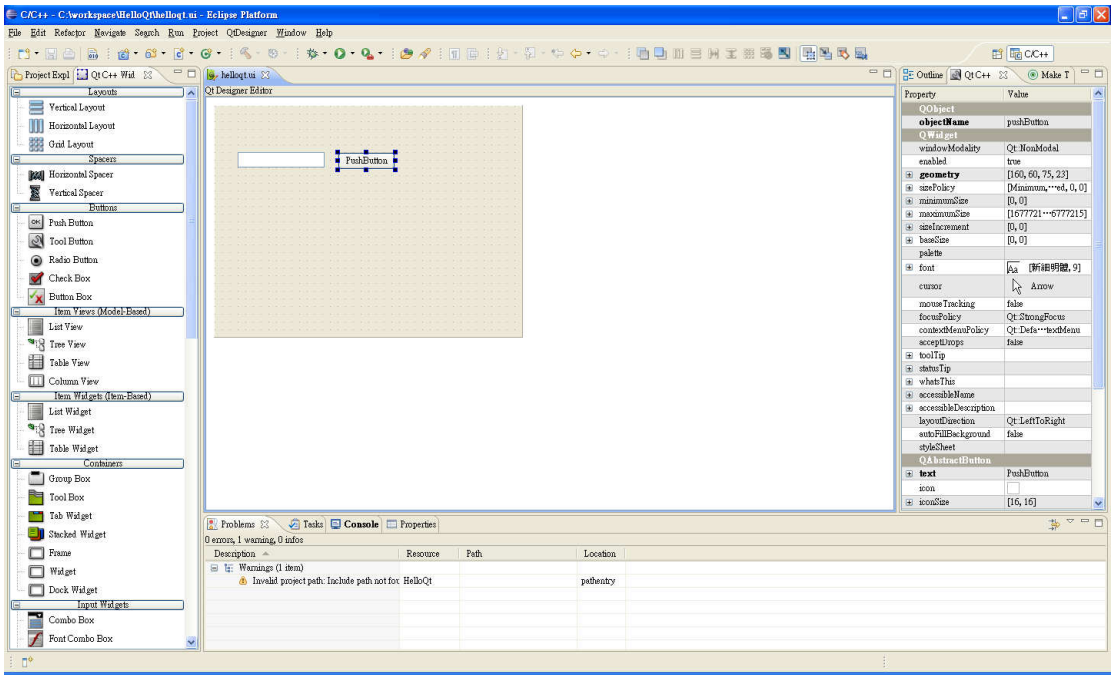


Right-Click on the Line Edit and select “Change objectName…”

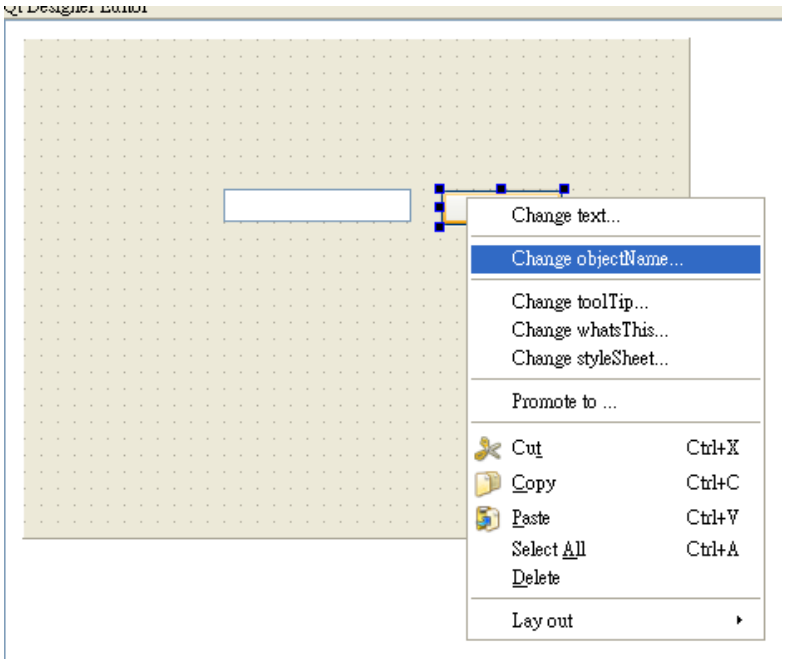


Enter “userName” for the object Name and click **OK**.

Drag and drop the “Push Button” widget into the hello.ui.

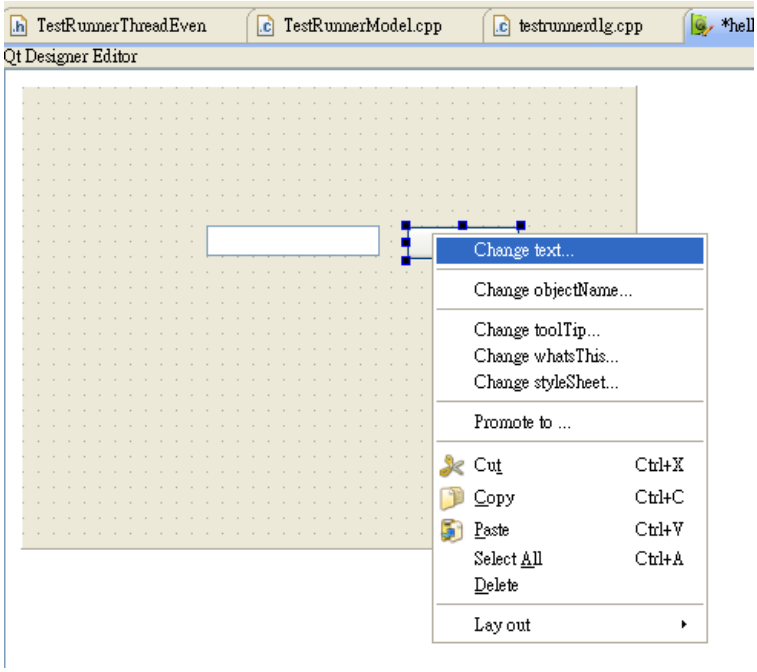


Right-Click on the Push Button and select “Change objectName…”

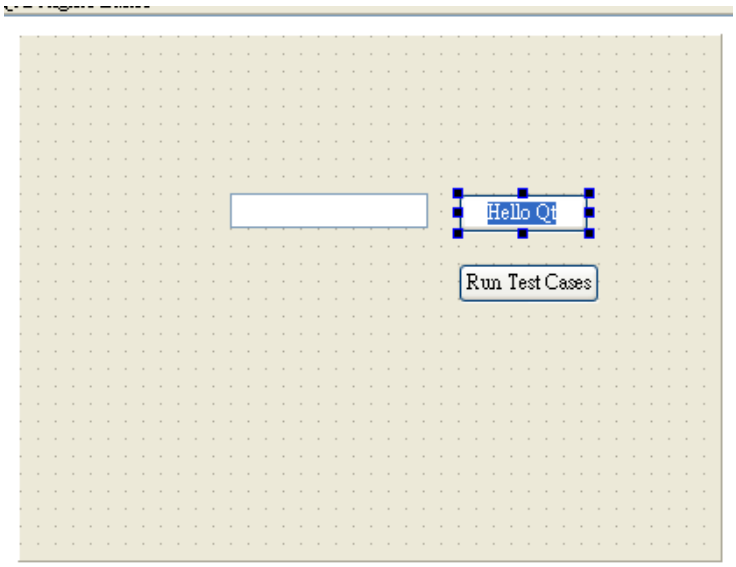


Enter “helloButton” for the object Name and click **OK**.

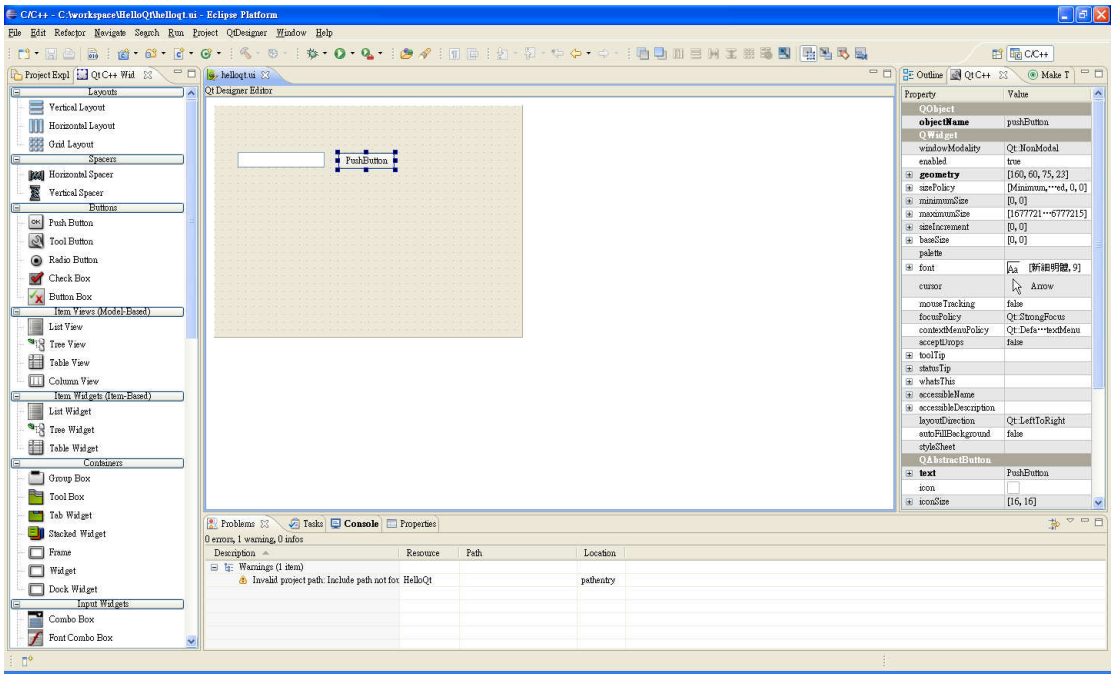
Right-Click on the Push Button and select “Change text…”



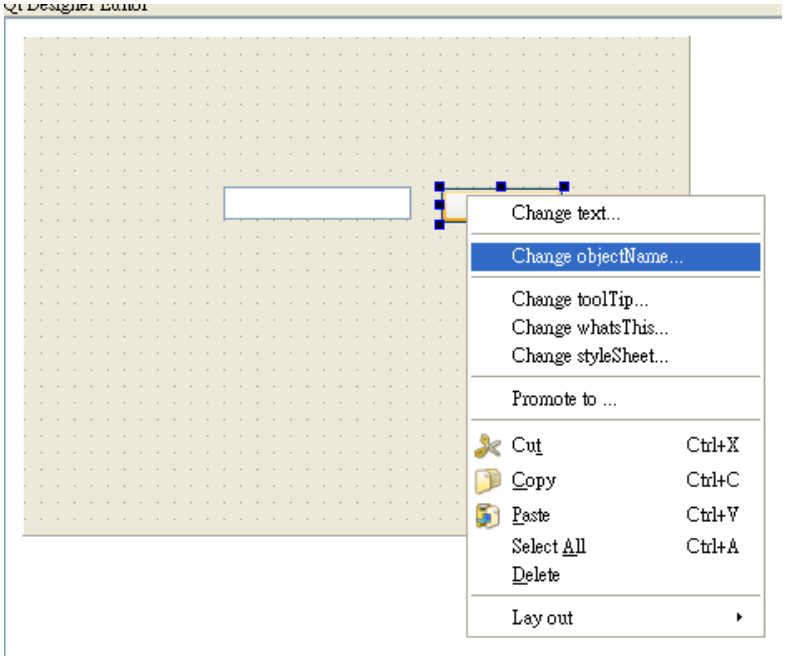
Enter “Hello Qt” for the text of push button.



Drag and drop a new “Push Button” widget into the hello.ui.

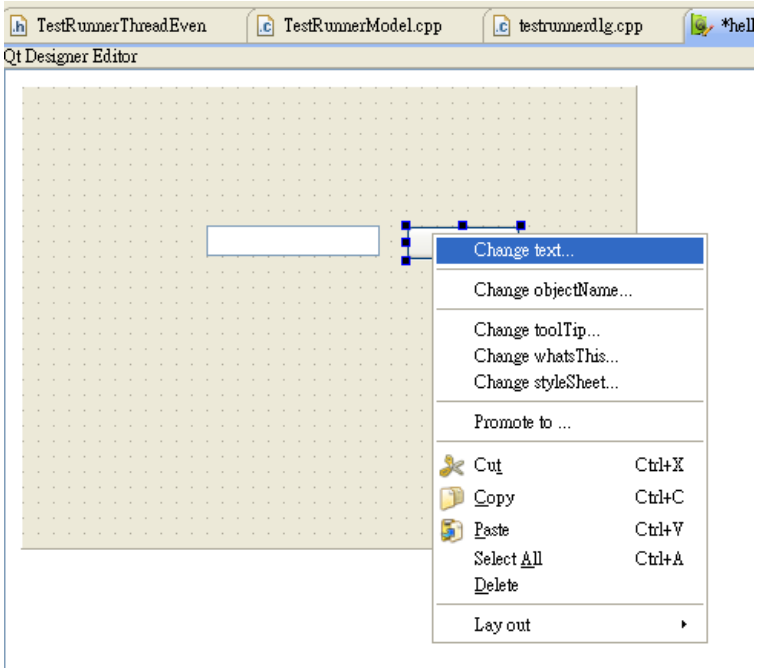


Right-Click on the Push Button and select “Change objectName…”

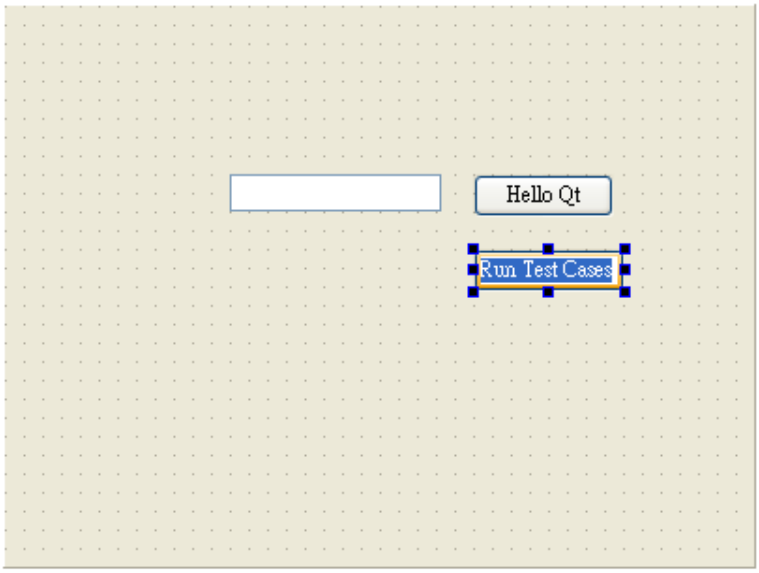


Enter “runButton” for the object Name and click **OK**.

Right-Click on the Push Button and select “Change text…”



Enter “Run Test Cases” for the text of push button.



Modify the helloqt.h

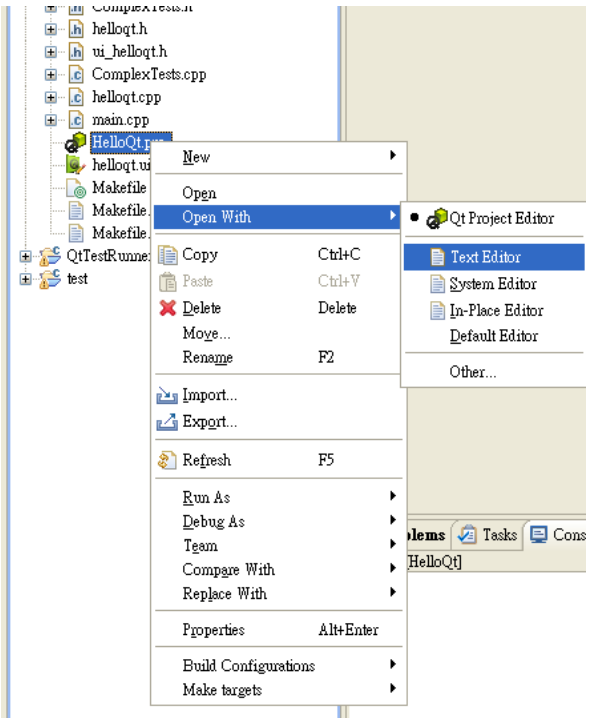
```
#ifndef HELLOQT_H
#define HELLOQT_H

#include <QtGui/QWidget>
#include "ui_helloqt.h"
#include <qmessagebox.h>

class HelloQt : public QWidget
{
    Q_OBJECT

public:
    HelloQt(QWidget *parent = 0);
    ~HelloQt();

private:
    Ui::HelloQtClass ui;
private slots:
    void on_helloButton_clicked();
    void on_runButton_clicked();
    void testrunner();
};

#endif // HELLOQT_H
```

Modify the helloqt.cpp

```
#include "helloqt.h"
#include <iostream>
#include <qapplication.h>
#include <cppunit/ui/qt/TestRunner.h>
#include <cppunit/extensions/TestFactoryRegistry.h>

using namespace std;

HelloQt::HelloQt(QWidget *parent)
    : QWidget(parent)
```

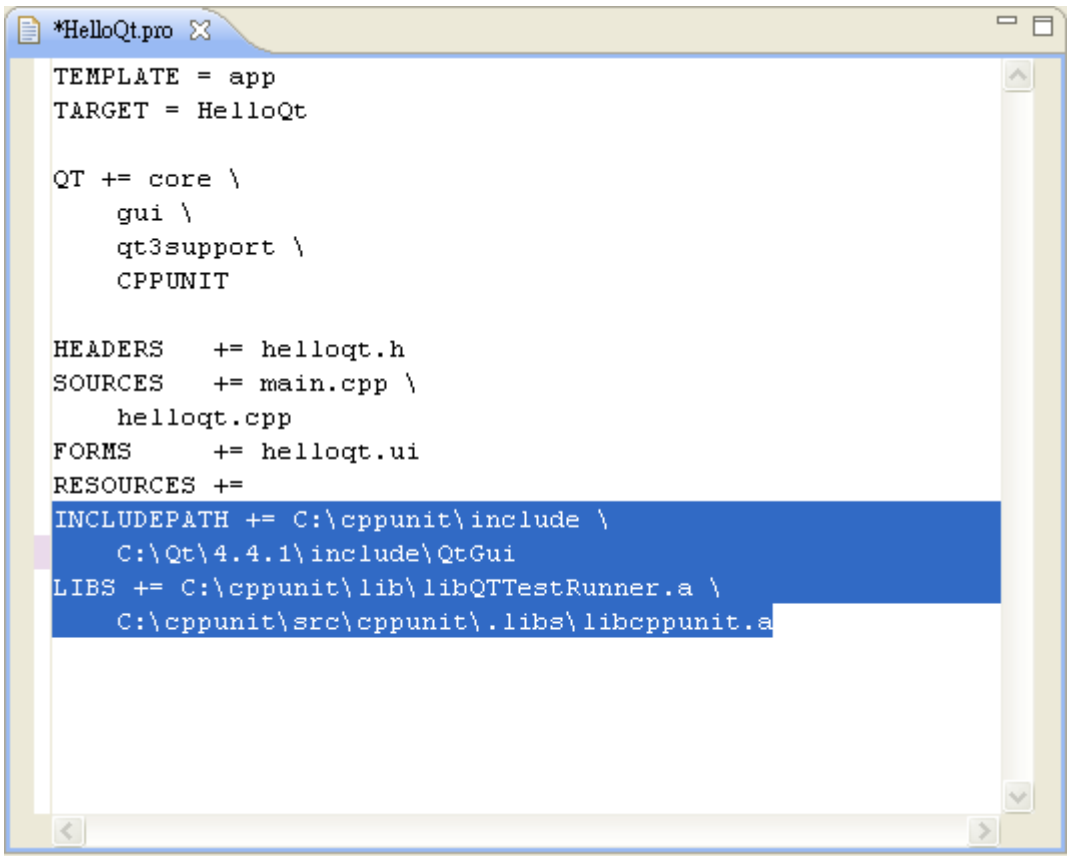
```
{
    ui.setupUi(this);
}

HelloQt::~HelloQt()
{

}

void HelloQt::on_helloButton_clicked()
{
    QString str = ui.userName->text();
    QMessageBox::information(NULL,"Alert","Hello "+str);
}

void HelloQt::on_runButton_clicked()
{
    testrunner();
}

void HelloQt::testrunner()
{
    // Create the QtTestRunner:
    QApplication app();
    CppUnit::QtTestRunner runner;

    // Adds the test to the list of tests to run:
    runner.addTest(CppUnit::TestFactoryRegistry::getRegistry().makeTe
st());

    // Run the GUI:
    runner.run();
}
```

Right-Click on Hello.pro and click “Open With/Text Editor” .



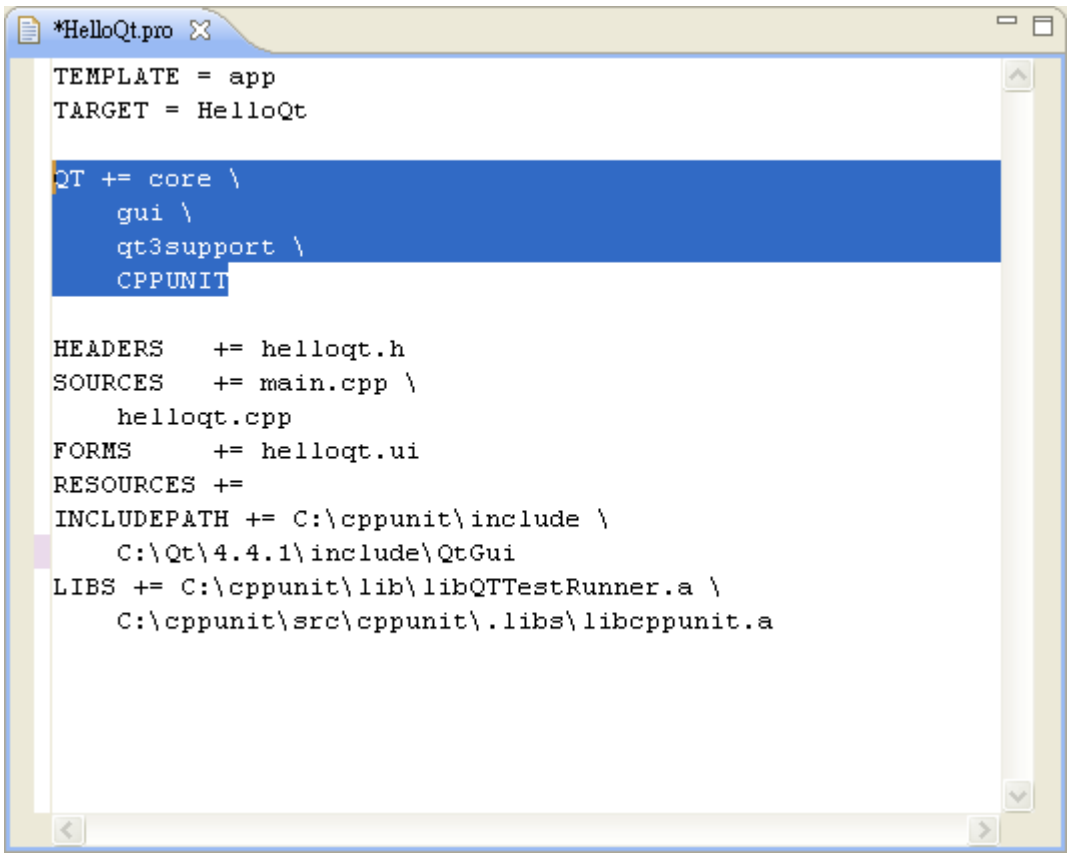
Add INCLUDEPATH and LIBS sections to the end of HelloQt.pro.

```
INCLUDEPATH += C:\cppunit\include \
    C:\Qt\4.4.1\include\QtGui
LIBS += C:\cppunit\lib\libQTTestRunner.a \
    C:\cppunit\src\cppunit\.libs\libcppunit.a
```

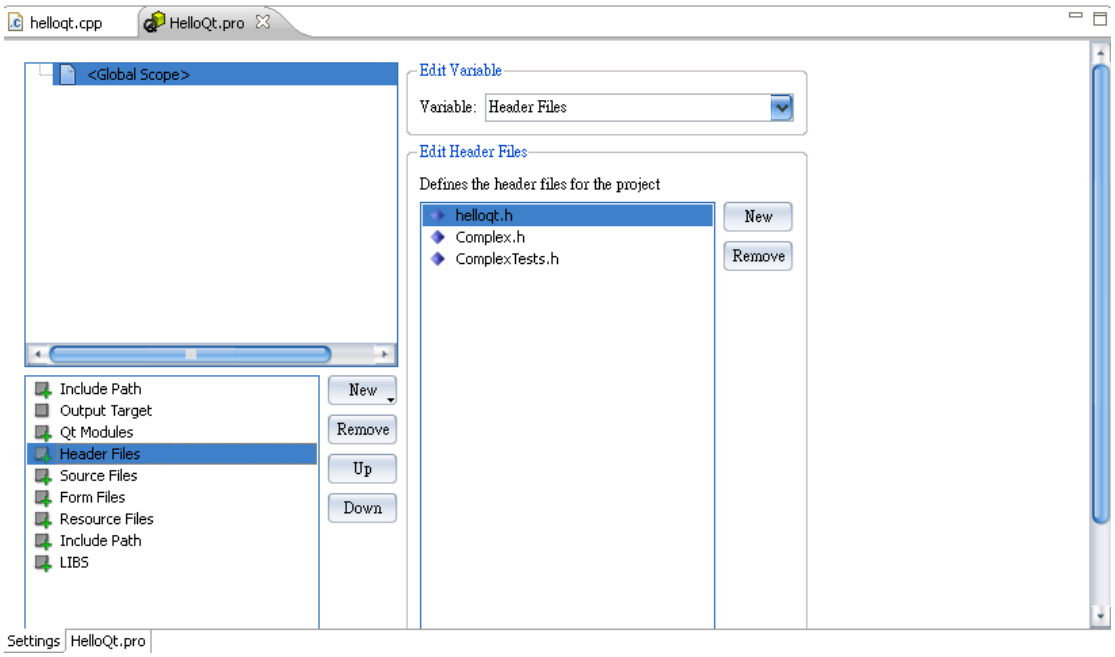


Modify QT section of HelloQt.pro.

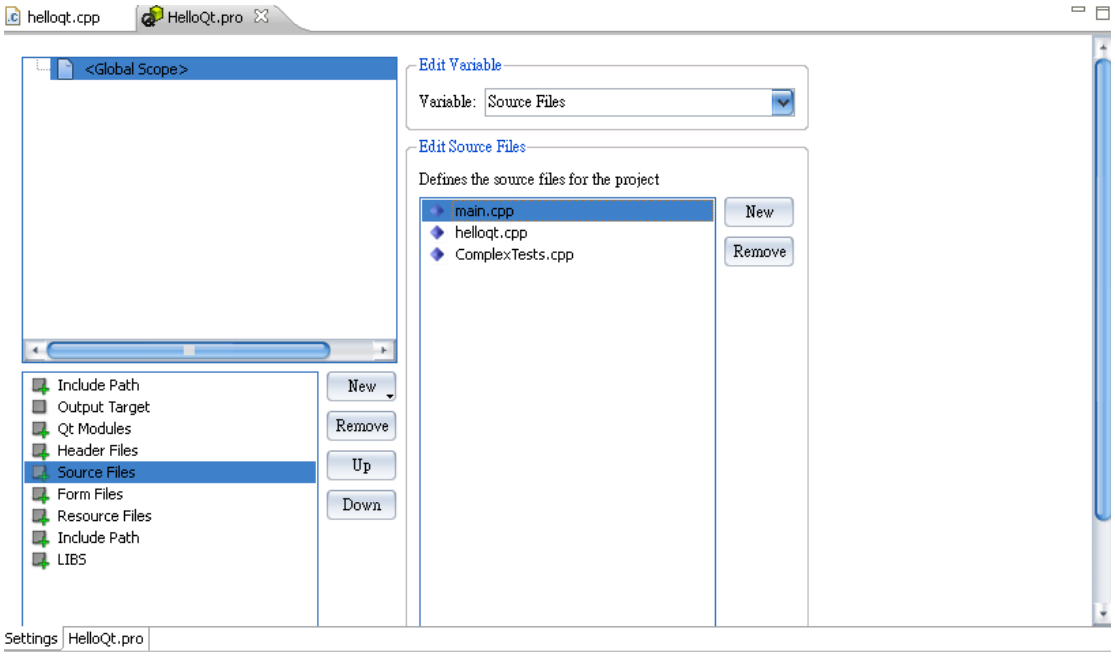
```
QT += core \
    gui \
    qt3support \
    CPPUNIT
```



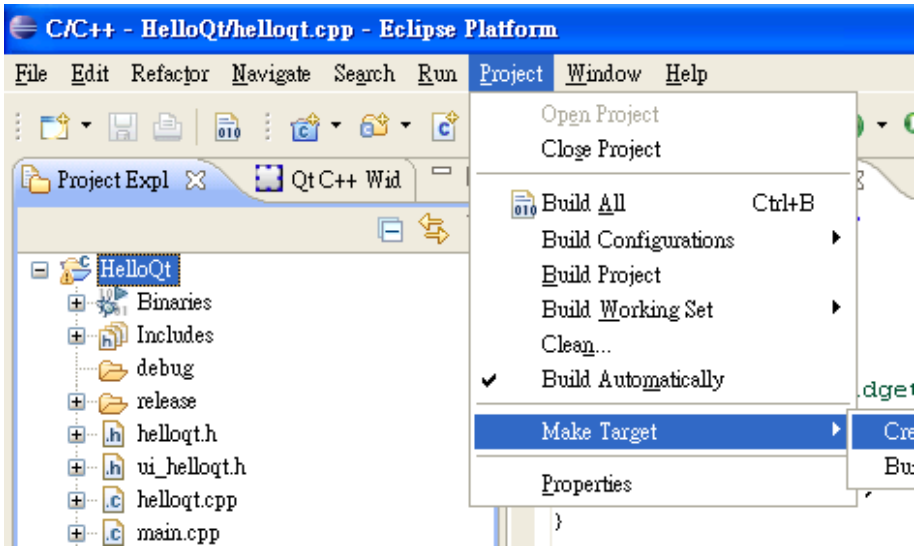
Add the **Complex.h** and **ComplexTests.h** to the **HelloQt.pro/Header Files**



Add the **ComplexTests.cpp** to the **HelloQt.pro/Source Files**

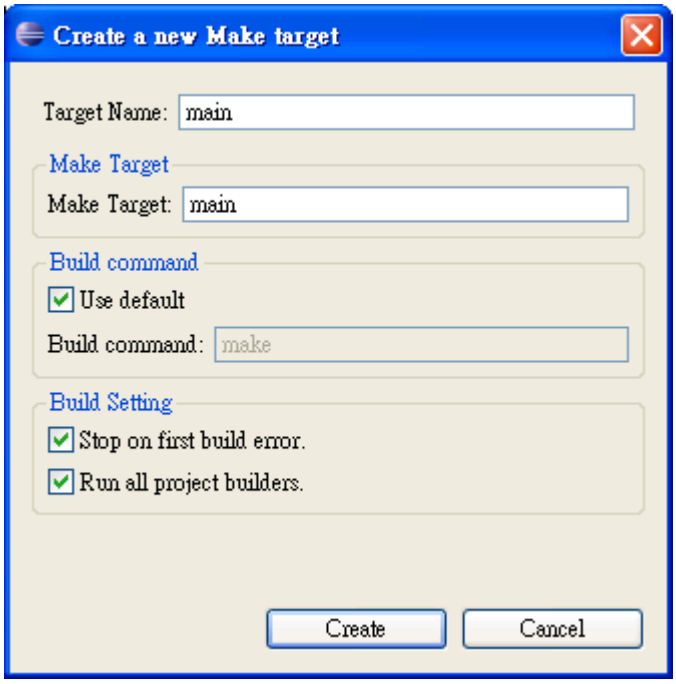


Click on “Project / Make Target / Create…”

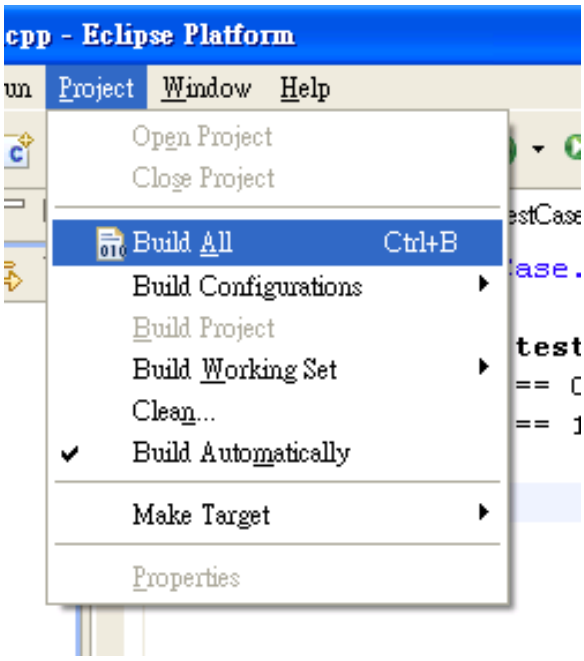


In the **Target Name** field, type “main”.

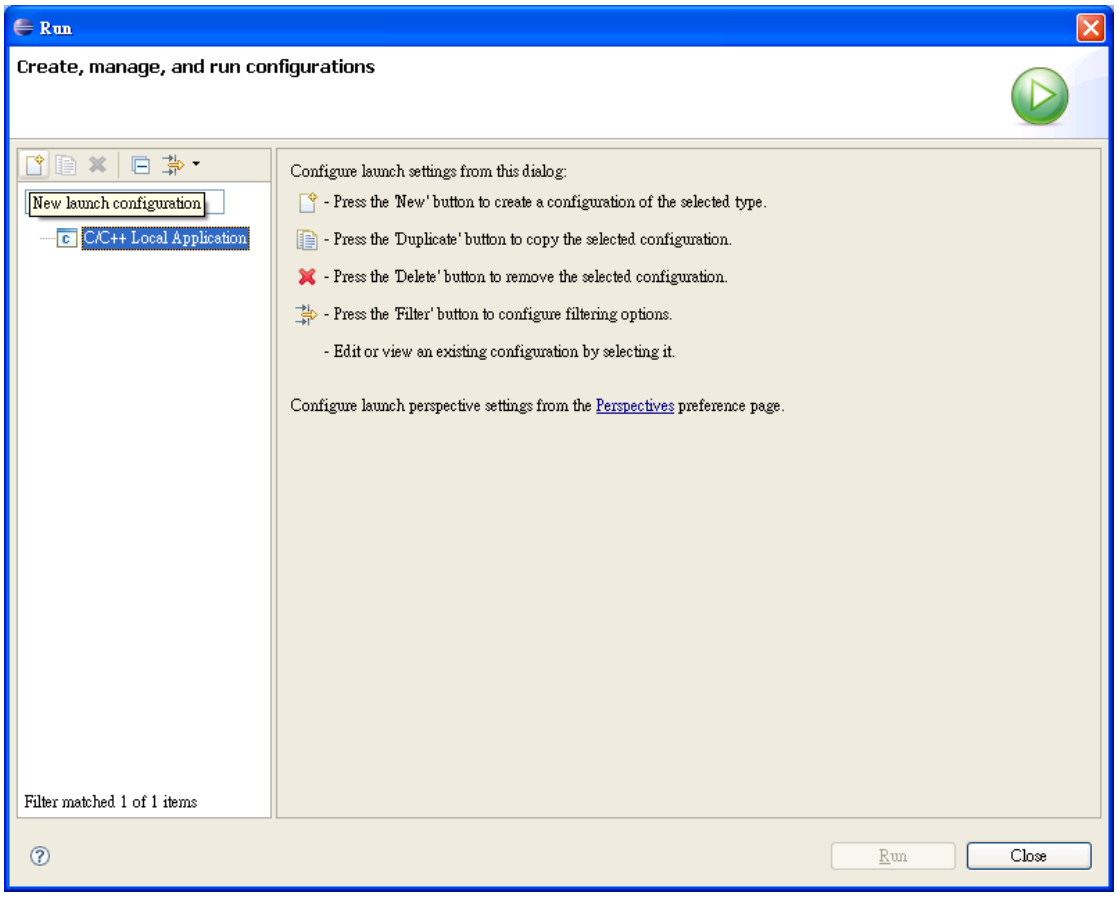
In the **Make Target** field, type “main”, and click **Create** button.



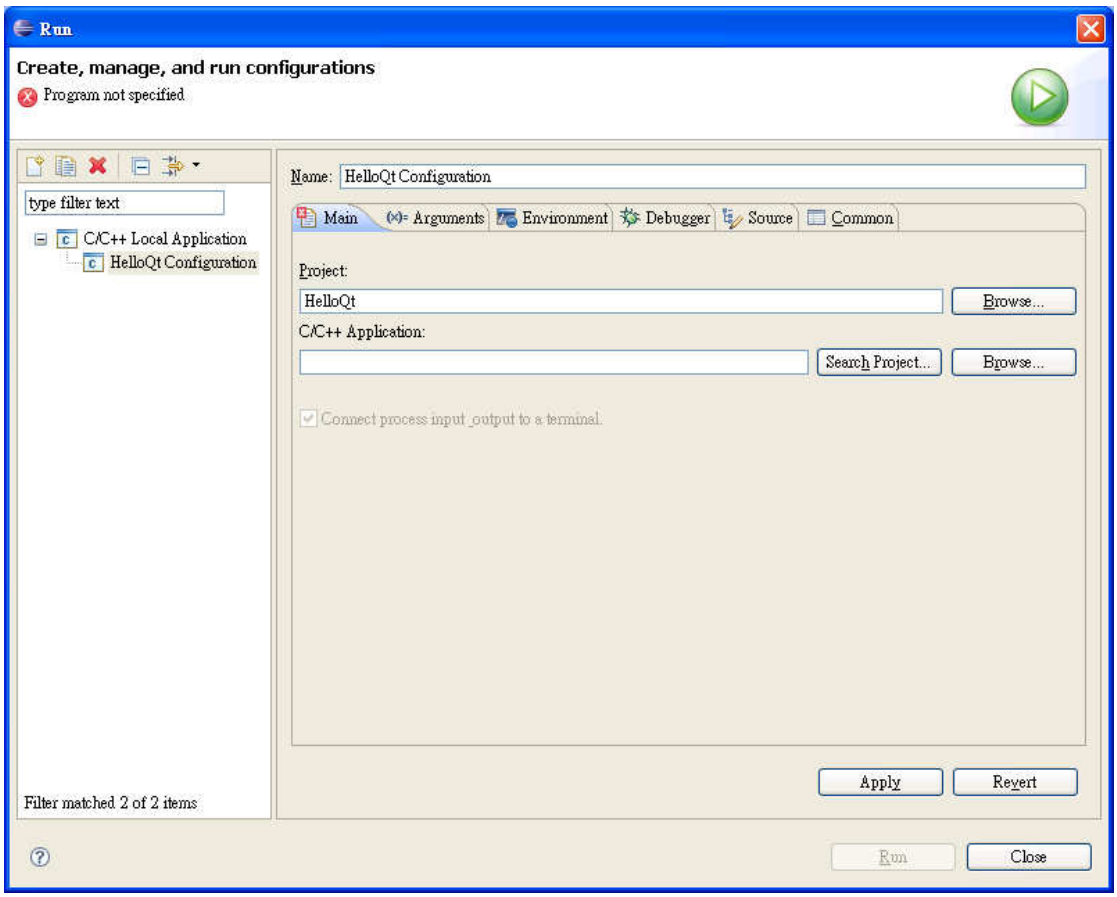
Click on “Project/Build All” to compile source code.



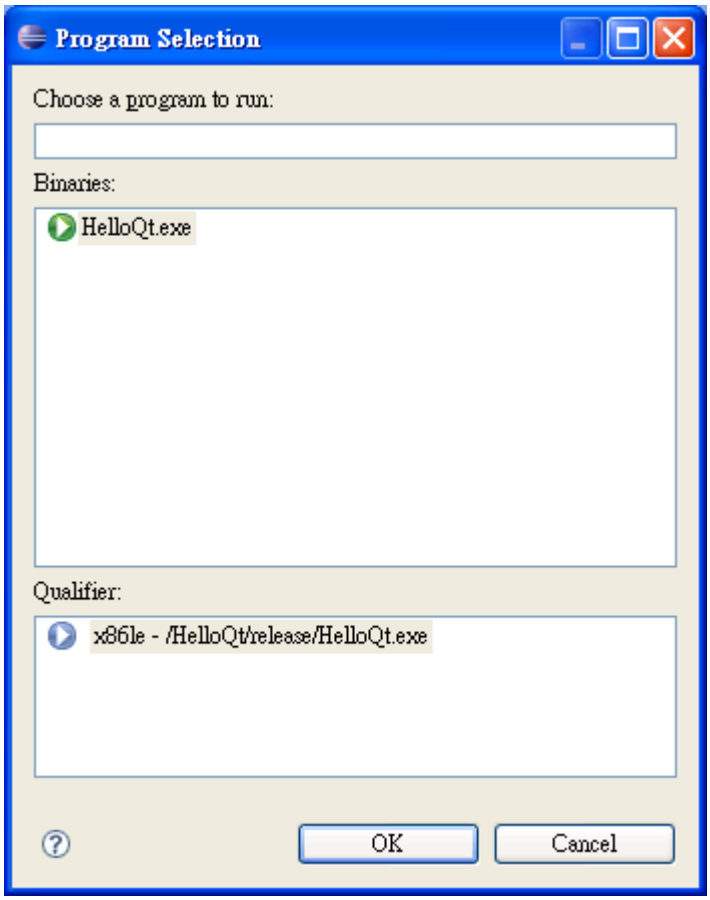
Click on “Run/Open Run Dialog…”. In the Configurations window, click on **C/C++ Local Application**, and then the **New launch configuration** button.



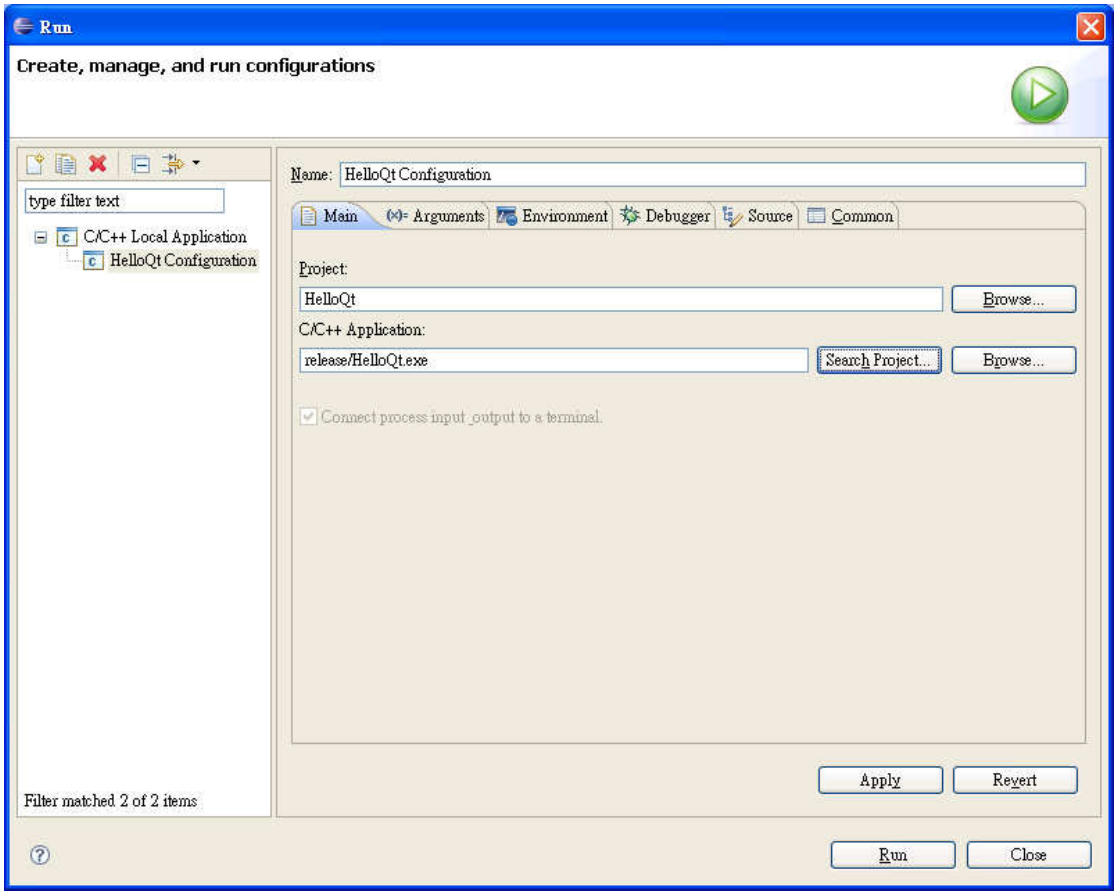
Click on the **Search Project** button:



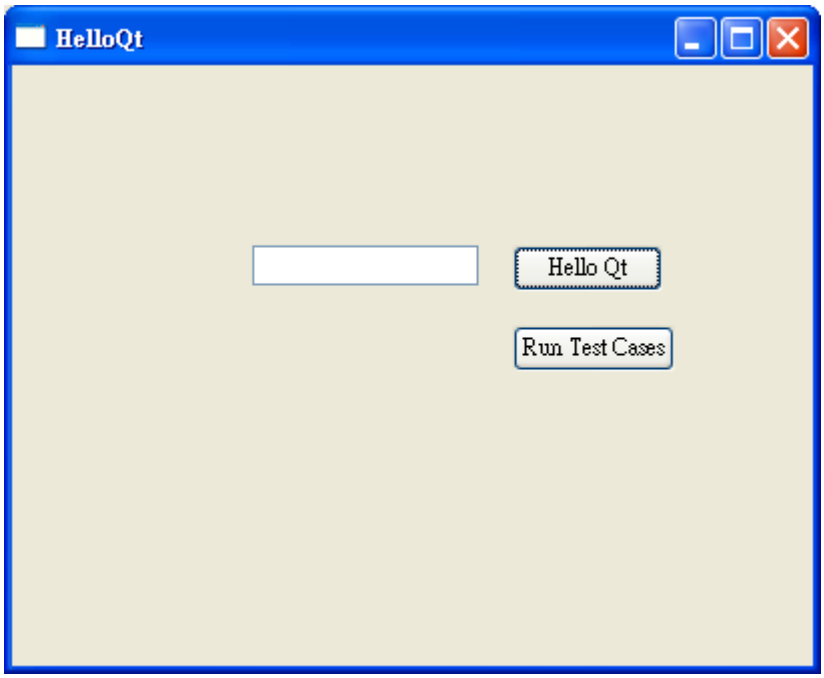
Click on the **OK** button:



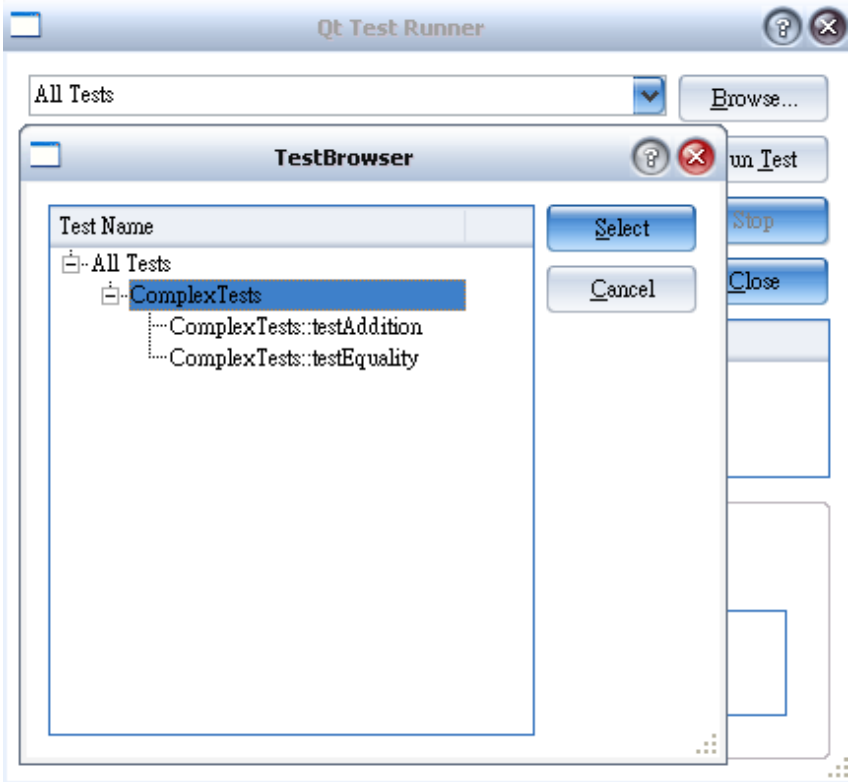
Now click the **Run** button to run your application:



You should get the following screen:



Click **Run Test Cases/Browse/Select**



Click **Run Test**