

Qt Development in Eclipse—QtCreator

Author: Dr. G. L. Jenkins

OS: Fedora 19

QtCreator: 2.7.0

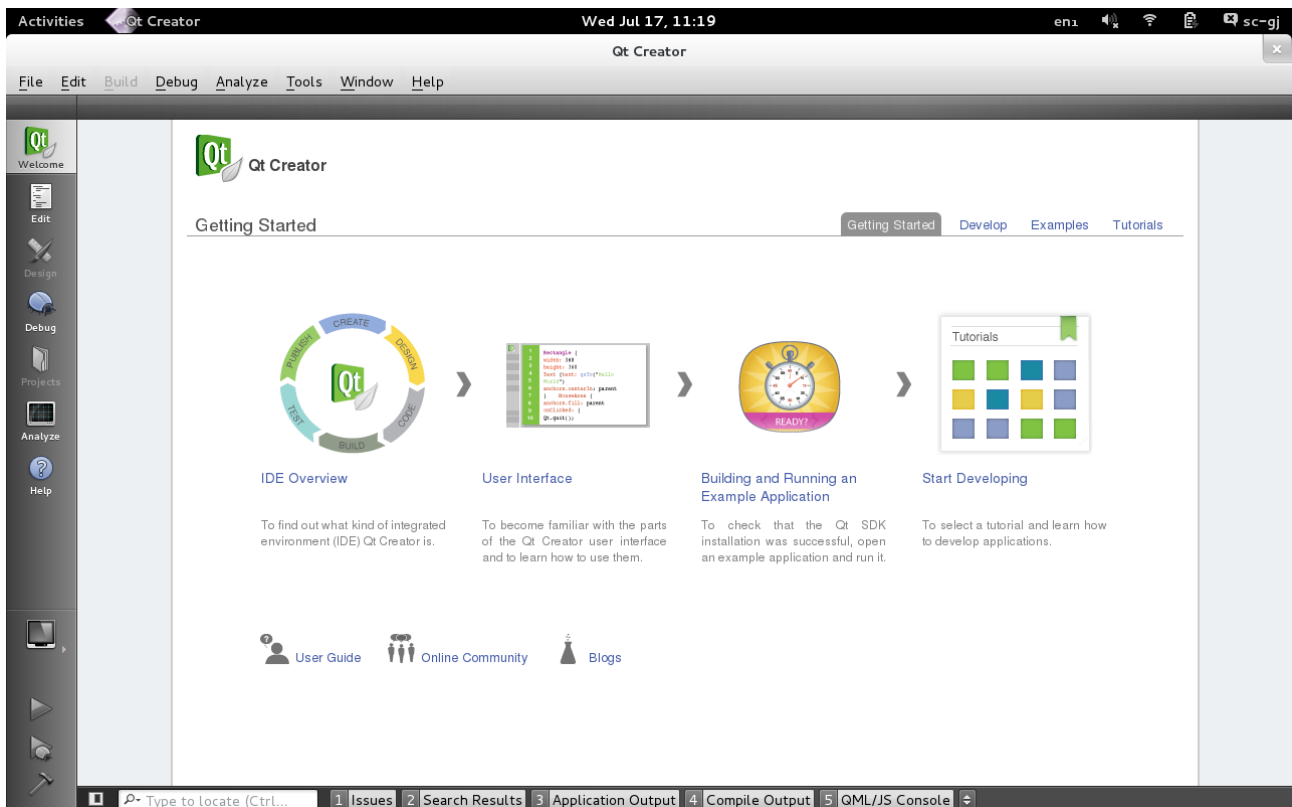
Previously making use of Qt's Plugin Architecture Trolltech (now owned by the Nokia Corporation) produced a plug-in for Eclipse which allowed its use for the development of Qt Applications. Qt Designer was also integrated and provided a single environment to develop both interface and functionality. The plugin is however now defunct and has been withdrawn, Digia the new owner of Qt are promoting the use of their Suite of tools, QtCreator (an editor) coupled with QtDesigner (and interface development tool) and QtAssistant (a guide to the language and libraries).

To Launch

Hover in the top left hand corner or click, then search for qt you should find the complete suite of Qt programs installed. One of my icons is missing, I don't know why. I suspect it has something to do with this laptop running a fugged version of QtCreator prior to it being updated.

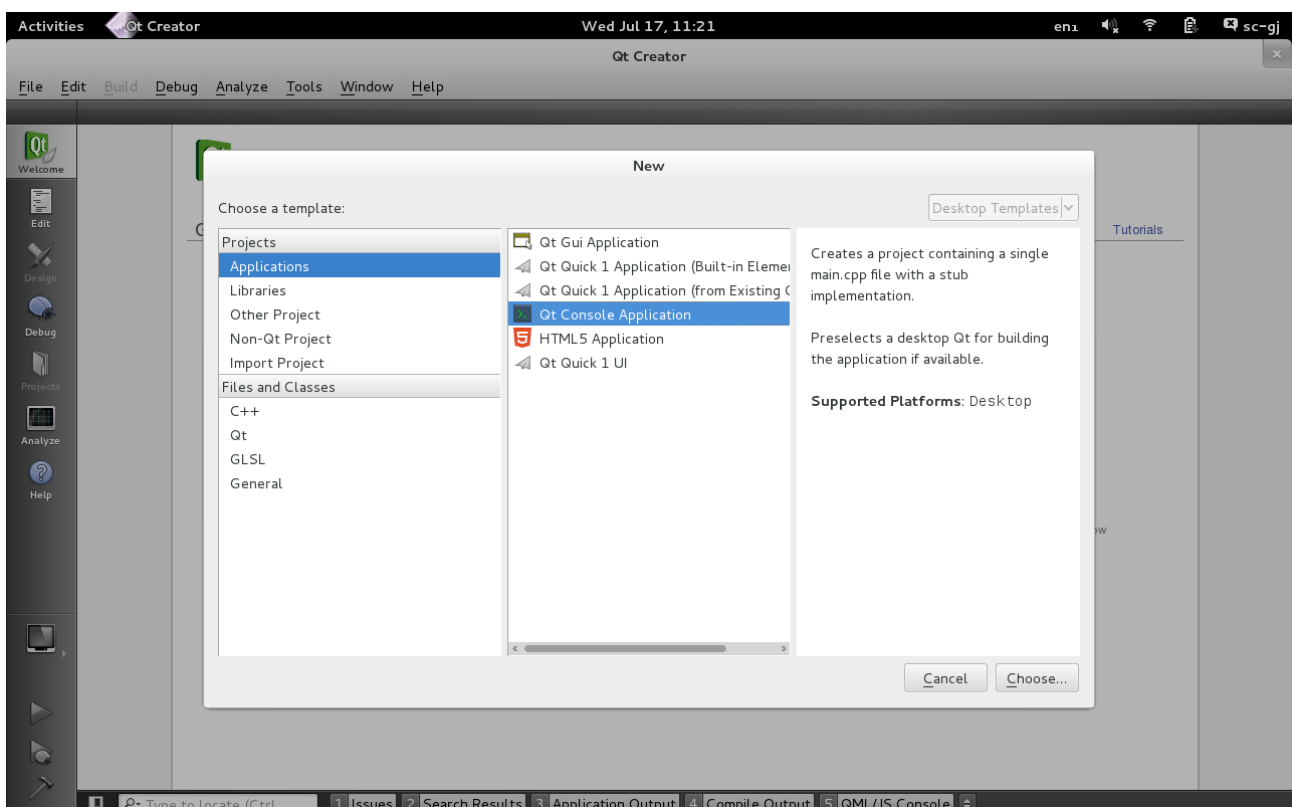


Launch QtCreator, it should look like this.

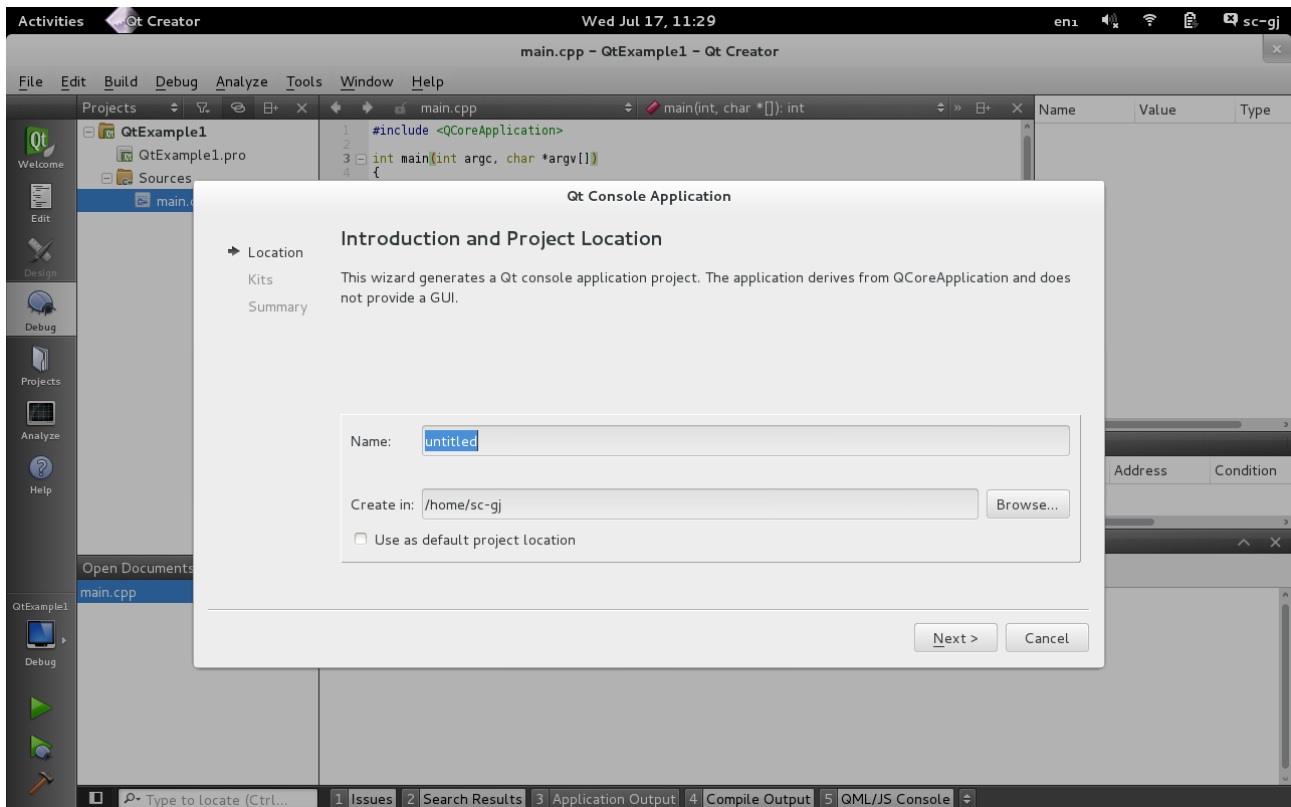


To create a hand-coded Qt project (without the designer) as we have been doing using the command line. Select from the menu **File > New File or Project**. This should give the following dialog with a range of more exotic project choices (the ones shown in the menu are also depended on the current perspective).

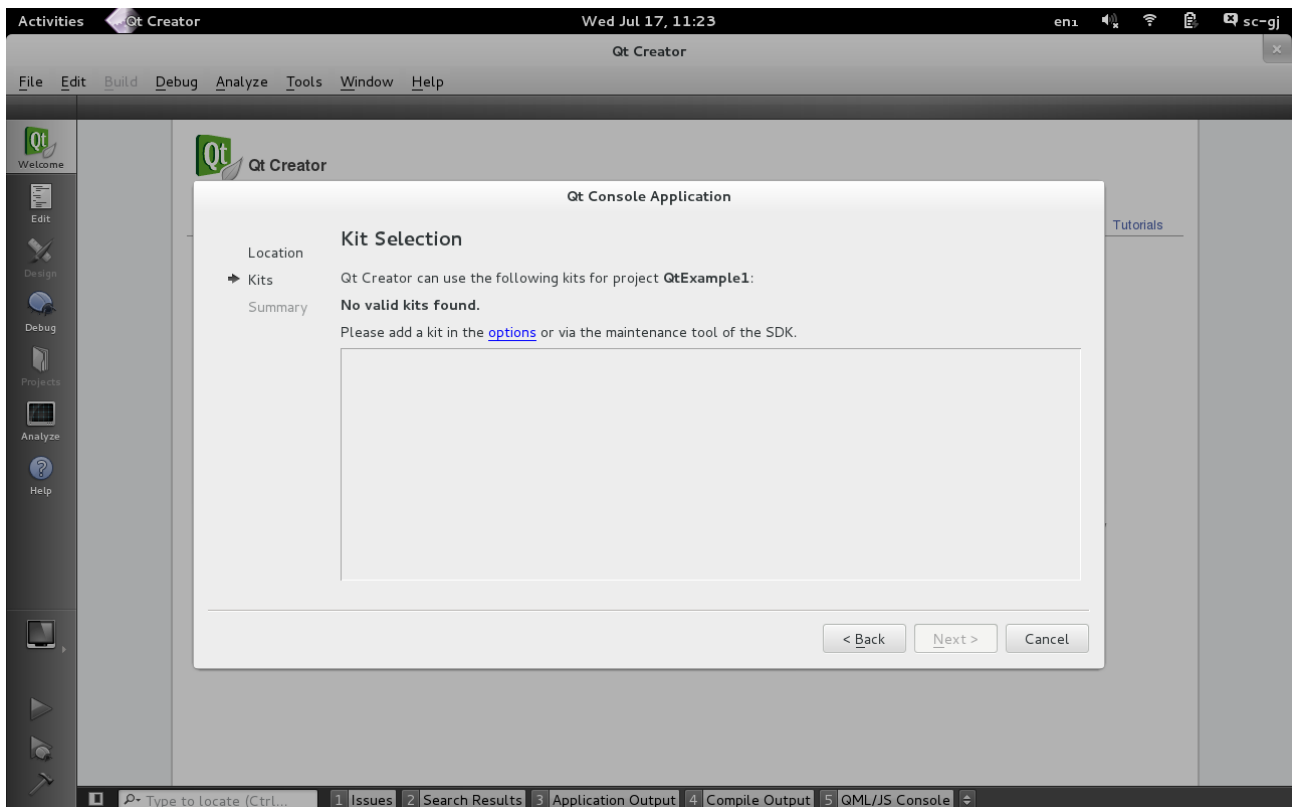
We select **Qt Console project** indicating that we do not want to create any designer files (UI files).



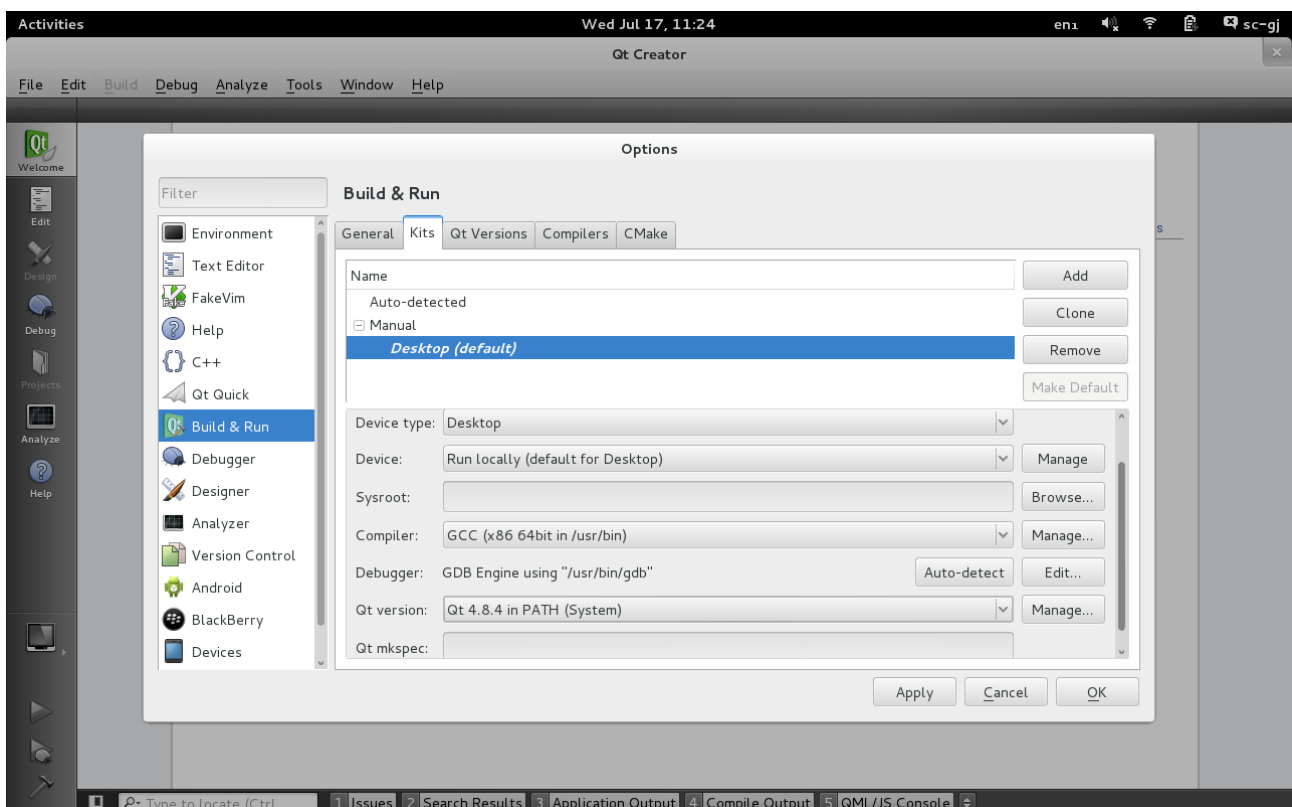
A second dialog allows us to name the project, name it something sensible and put it some location you'll be able to find it again. This is essentially just a folder which the .pro file will be created in.



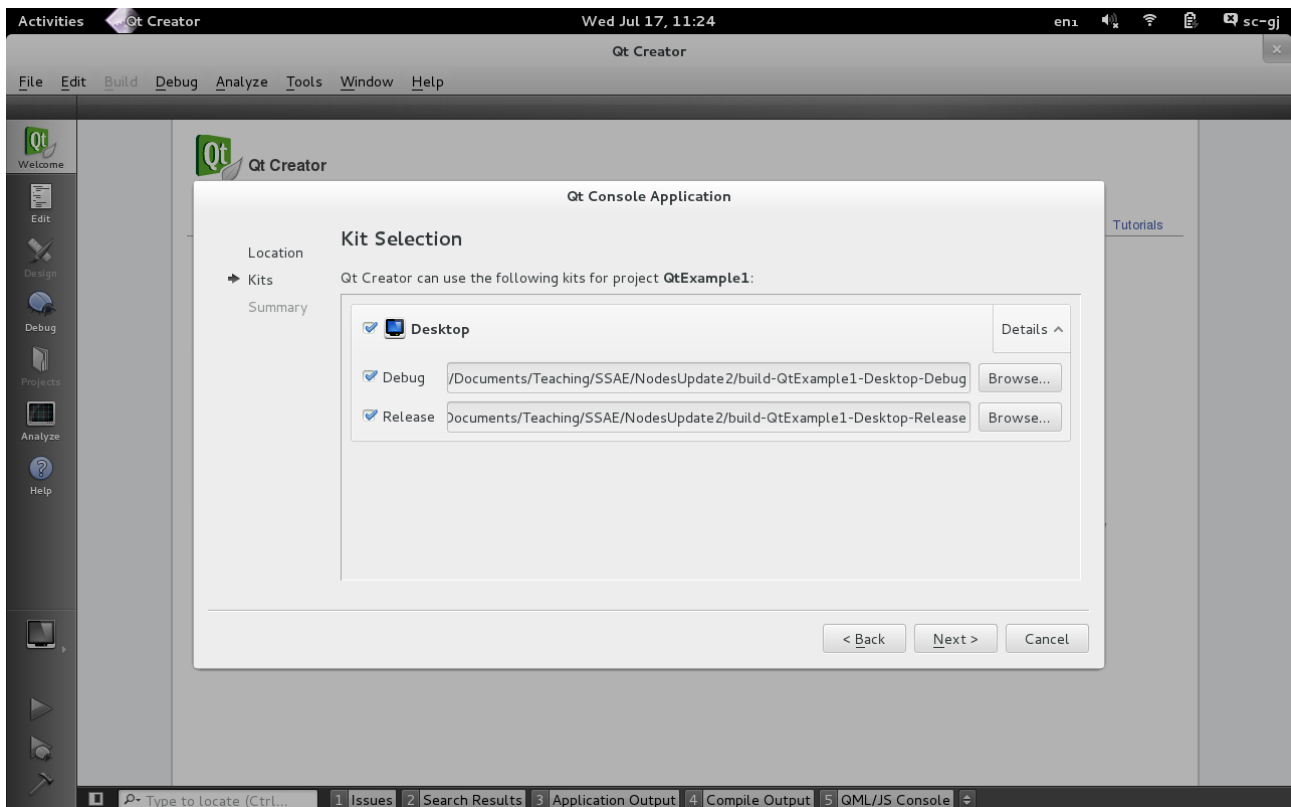
The next dialog asks us which Kit we are using? This relates to the compiler toolchain, we set up two tool chains one for debug and one for release. The tool needs to know what version of Qt, which compiler etc. to use. We don't have to worry about this as QtCreator will automatically detect the installed qt stuff so essentially involves a single selection.



Selection options to to setup the kit.



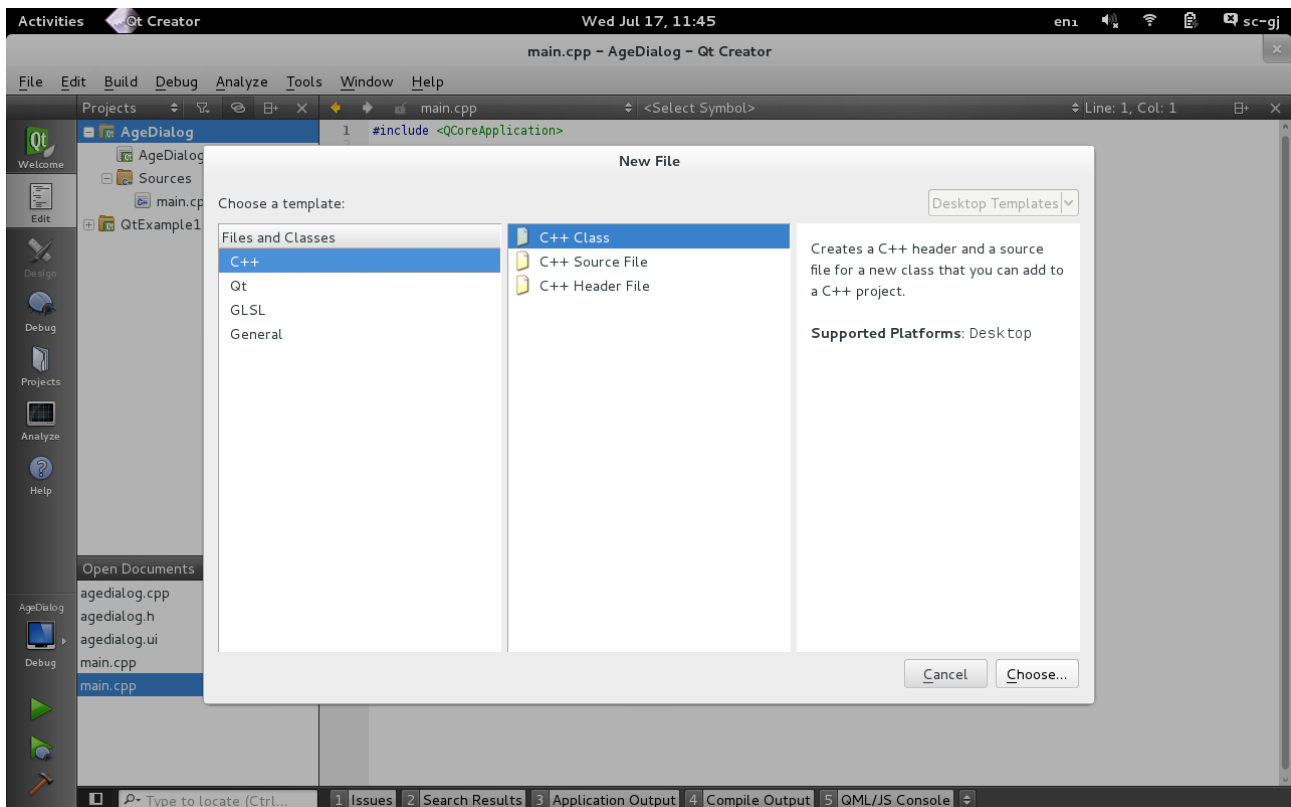
We select the Qt Version from the dropdown menu box and select Apply followed by Ok. We can now select which of the auto detected kits (tool chains) we'd like to use, we'll keep them both we'll be creating debug releases while developing but may eventually need to produce a release version.



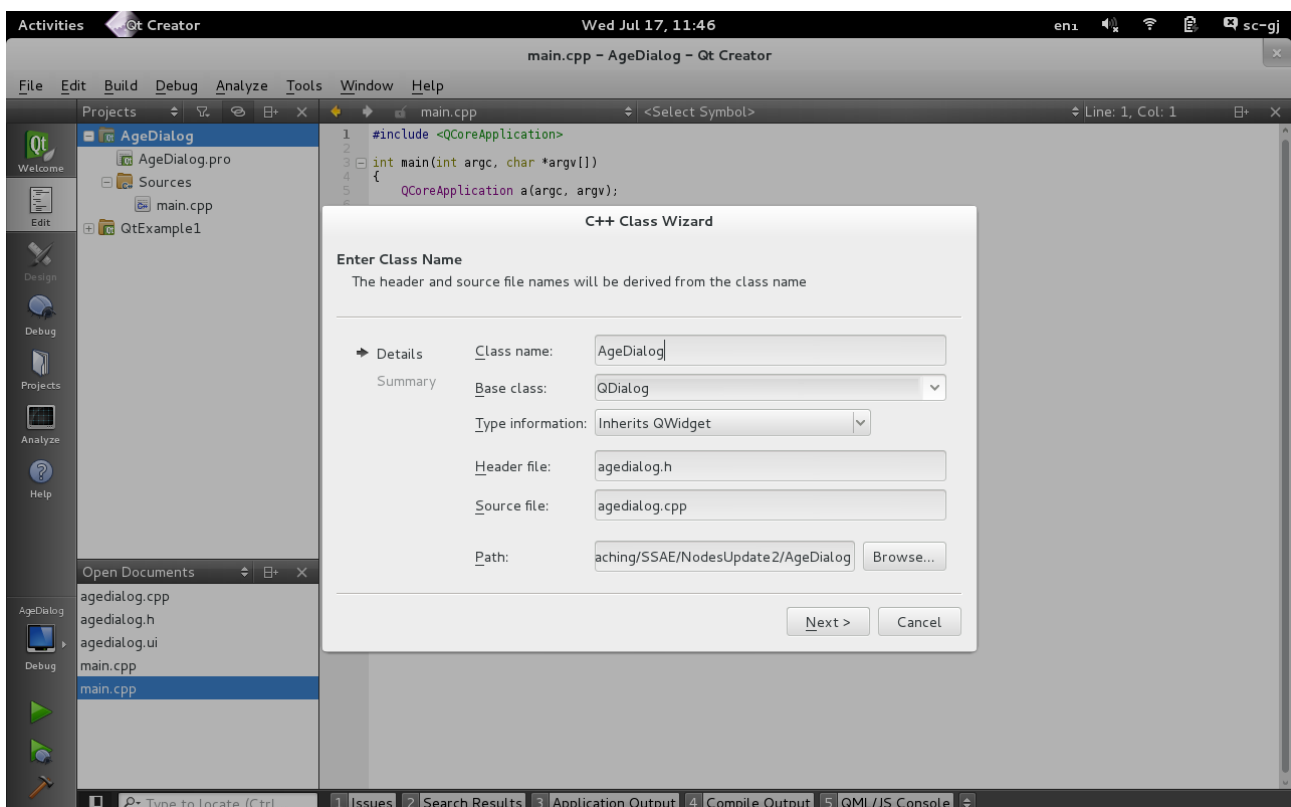
We are then presented with a Qt Project and a simple main file containing the QApplication code we have seen in our own Qt programs. We faced with array of windows specially geared towards the development of Qt programs. We'll look at some of these features later.

For this example we'll take a simple dialog class from one of the earlier lectures and insert and launch this from the Qt Console Project. **Remember that QtCreator is trying to manager the make and Project (.pro) files we must use the add Class option to avoid problems.**

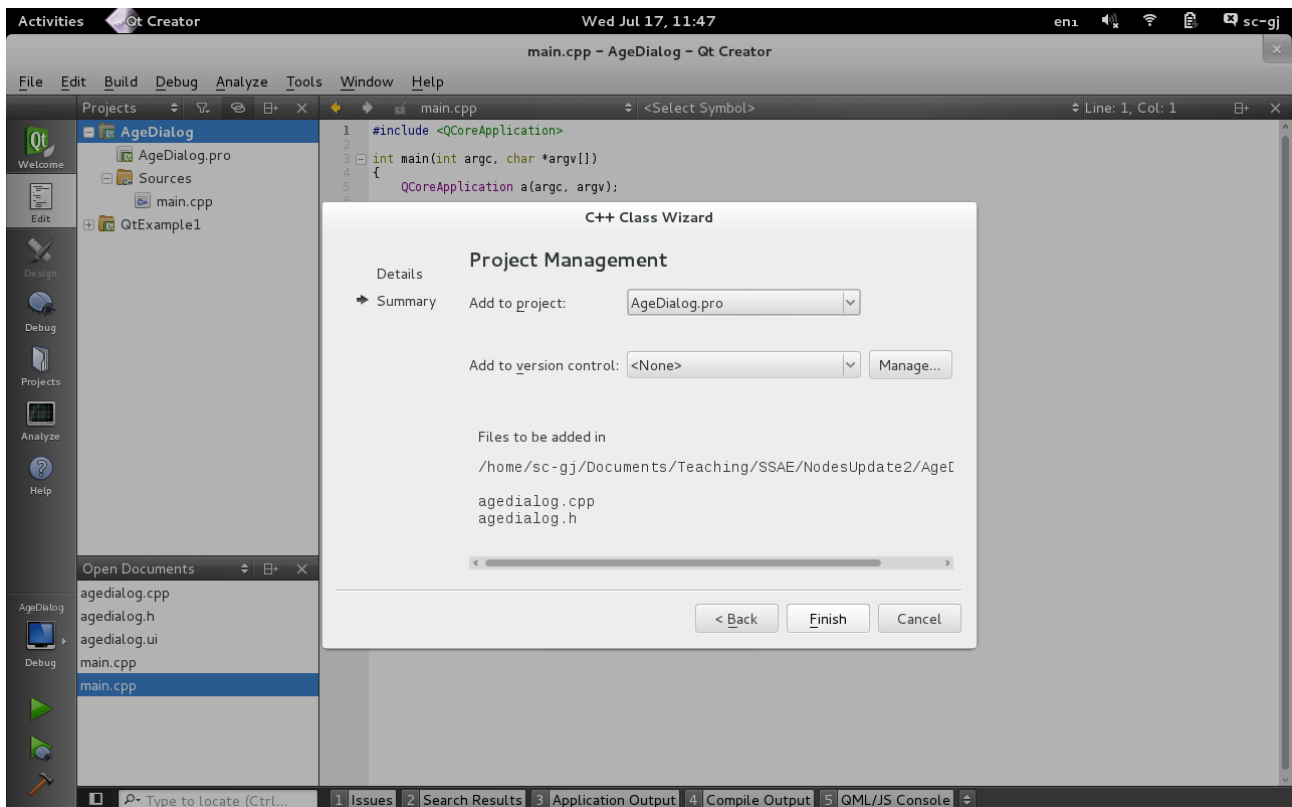
To add the new class select File > New > Class from the application menu. Then C++ and C++ class again.



This presents us with the following dialog which allows us to select a name and if appropriate a base class (parent class) for the class we are creating.



Ok, so looks good so far. This is actually going to cause us a problem later, but not one we cant' solve.



We're simply going to add the code from an earlier tutorial to these files (and to the main.cpp) to get something working. The code is given below.

```

/*
 * AgeDialog.h
 *
 * Created on: 17-Jul-2013
 * Author: sc-gj
 */

#ifndef AGEDIALOG_H_
#define AGEDIALOG_H_

#include <QDialog>

class QSpinBox;
class QSlider;
class QPushButton;
class QHBoxLayout;

class AgeDialog : public QDialog
{
    /* This is required for any object which
       defines signals and slots */
    Q_OBJECT;

private:
    //Widgets to be placed on the form
    QSpinBox *spinBox;
    QSlider *slider;
    QHBoxLayout *layout;
    QPushButton *exit;

```

```

public:
    //Constructor has a default of null for the parent
    AgeDialog(QWidget *parent =0);

/* The signals keyword is actually a macro it is replaced by
the pre-processor before the C++ compiler sees it */
signals:
    // No user defined signals here at present

/* ditto :-D */
private slots:
    // No private slots at present
};
#endif /* AGEDIALOG_H_ */

/*
 * AgeDialog.cpp
 *
 * Created on: 17-Jul-2013
 * Author: sc-gj
 */

#include <QtGui>
#include "agedialog.h"

//Constructor
AgeDialog::AgeDialog(QWidget *parent) : QDialog(parent)
{
    /*This code is virtually unchanged from the earlier example */

    spinBox = new QSpinBox(this);
    slider = new QSlider(Qt::Horizontal, this);
    exit = new QPushButton("Exit", this);

    /* As in the earlier examples we connect the spin box signal
to the slider slot and vice versa. Here we are using the
connect method associated with this class (though its still
a static method). */

    connect(spinBox, SIGNAL(valueChanged(int)),
            slider, SLOT(setValue(int)));
    connect(slider, SIGNAL(valueChanged(int)),
            spinBox, SLOT(setValue(int)));

    /* This additional button is has its clicked signal attached to the close
slot of the dialog */

    connect(exit, SIGNAL(clicked()),
            this, SLOT(close()));

    spinBox->setValue(28);

    layout = new QHBoxLayout();
    layout->addWidget(spinBox);
    layout->addWidget(slider);
    layout->addWidget(exit);

    /*Previously we attached this to a window widget
we were adding components too now we use the
dialog widget itself (this) */

    this->setLayout(layout);

```

```

}

//Signals, slots, other methods would go here if we had any!!

/*This has to change from QApplication*/
#include <QApplication>
#include "agedialog.h"

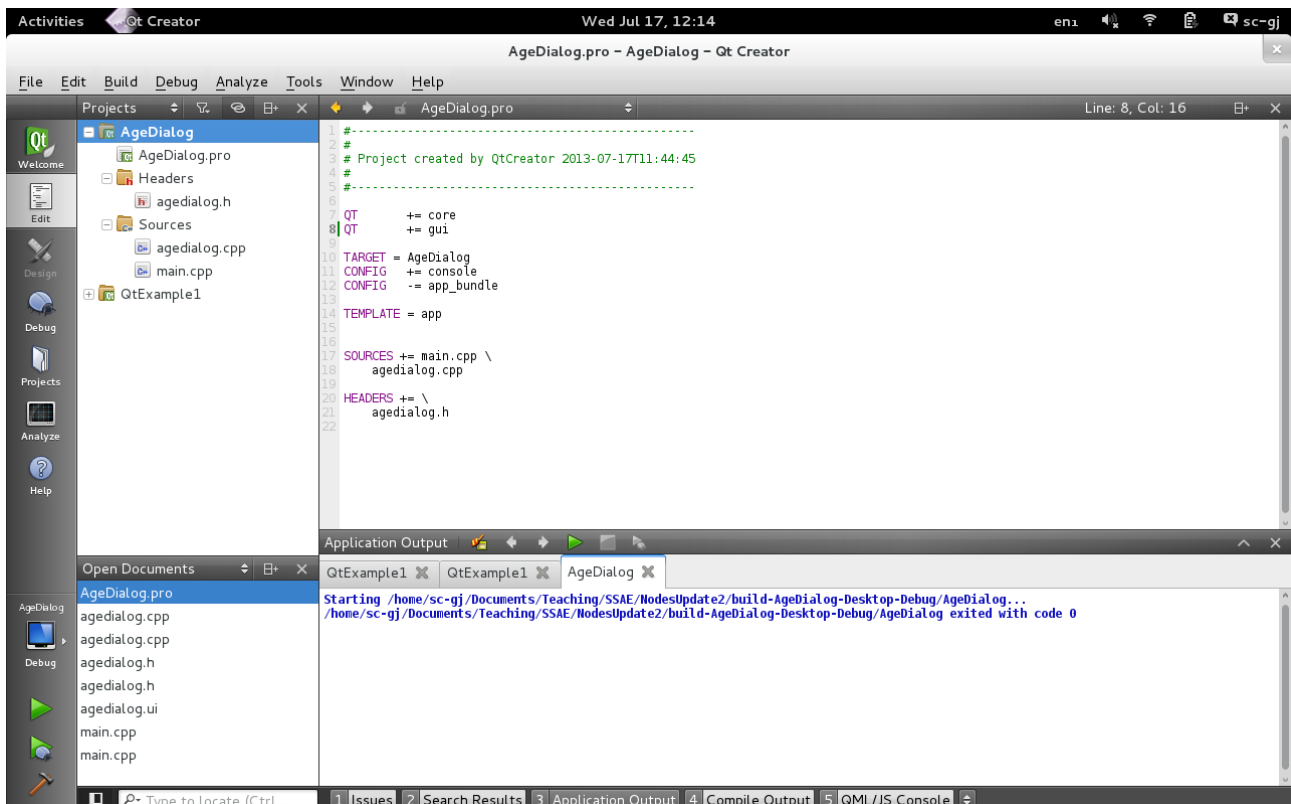
int main(int argc, char *argv[])
{
    /*This has to change from QApplication*/
    QApplication a(argc, argv);
    //Create an age dialog object
    AgeDialog *myDialog = new AgeDialog();
    myDialog->setWindowTitle("Enter your age");
    myDialog->show();

    //start dispatching messages
    return a.exec();
}

```

There are some minor issues, we have to change QApplication to QApplication the former is a cut down version of the latter. Having copied and pasted the code, the compiler throws some errors :- (The reason for this is in the .pro file, we need to ask Qt to do the linking of the appropriate files.

So we modify the .pro file to include the gui stuff.



We can no re-run qmake, clean and build. These are all found in the Build menu. You should be familiar with these from working with the command line stuff, the tool does exactly the same stuff.

//TODO – Need to re-write this.

Qt Designer

Qt Designer's behaviour in terms of the classes it generates from the editor are not what we might expect read the following explanation before proceeding with worksheet!

Qt was designed to be pleasant and intuitive to hand-code and many developers still develop programs entirely in C++. Other programmers find it more productive to use a visual approach to designing dialogs allowing rapid and fluid design.

Qt Designer expands the programming opportunities available to programmers by providing a visual interface to design some or all of the forms in a Qt Application. The files created by the designer are standard C++ and can be used within a conventional tool chain (like the command line one we have been using).

Qt Designer Eclipse Plug-in

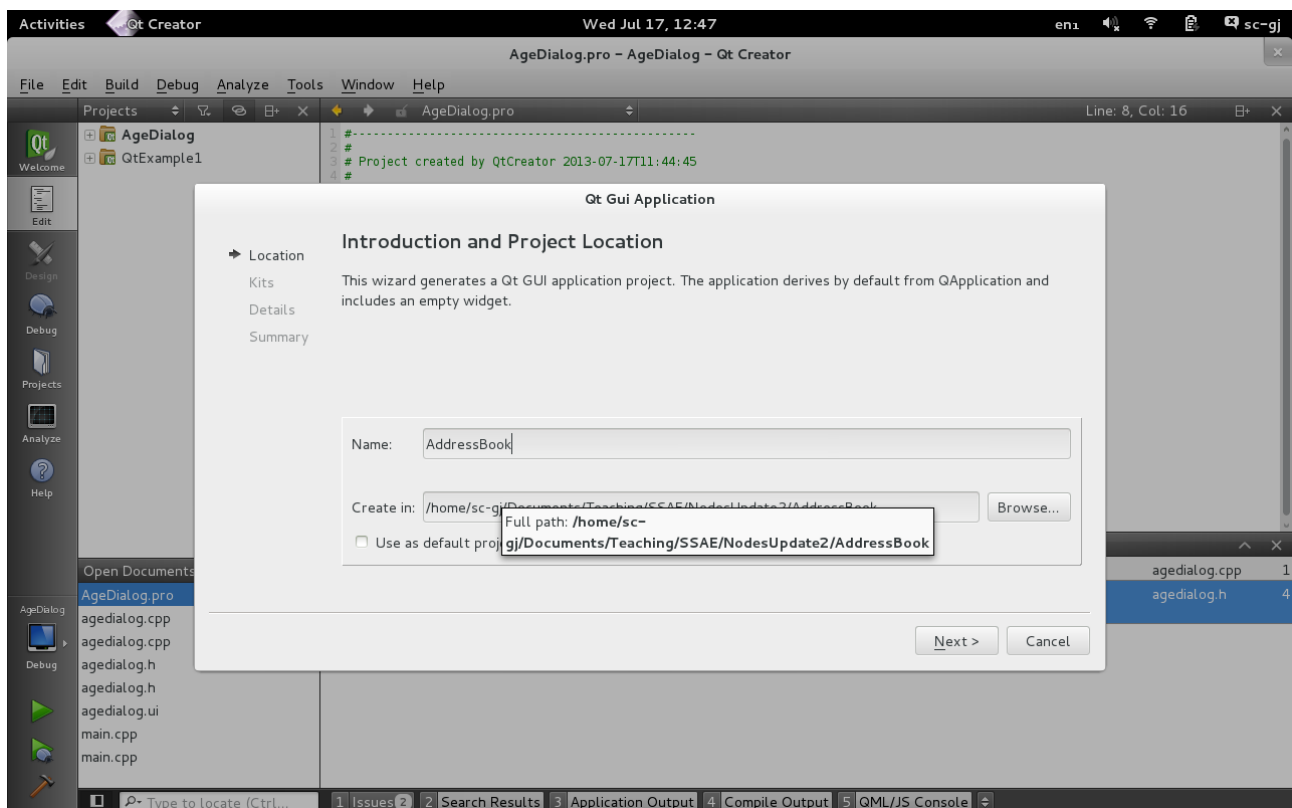
Adapted from: "Qt Eclipse Integration 4.4.1" - Trolltech - 2008 *

* - heavily adapted!!

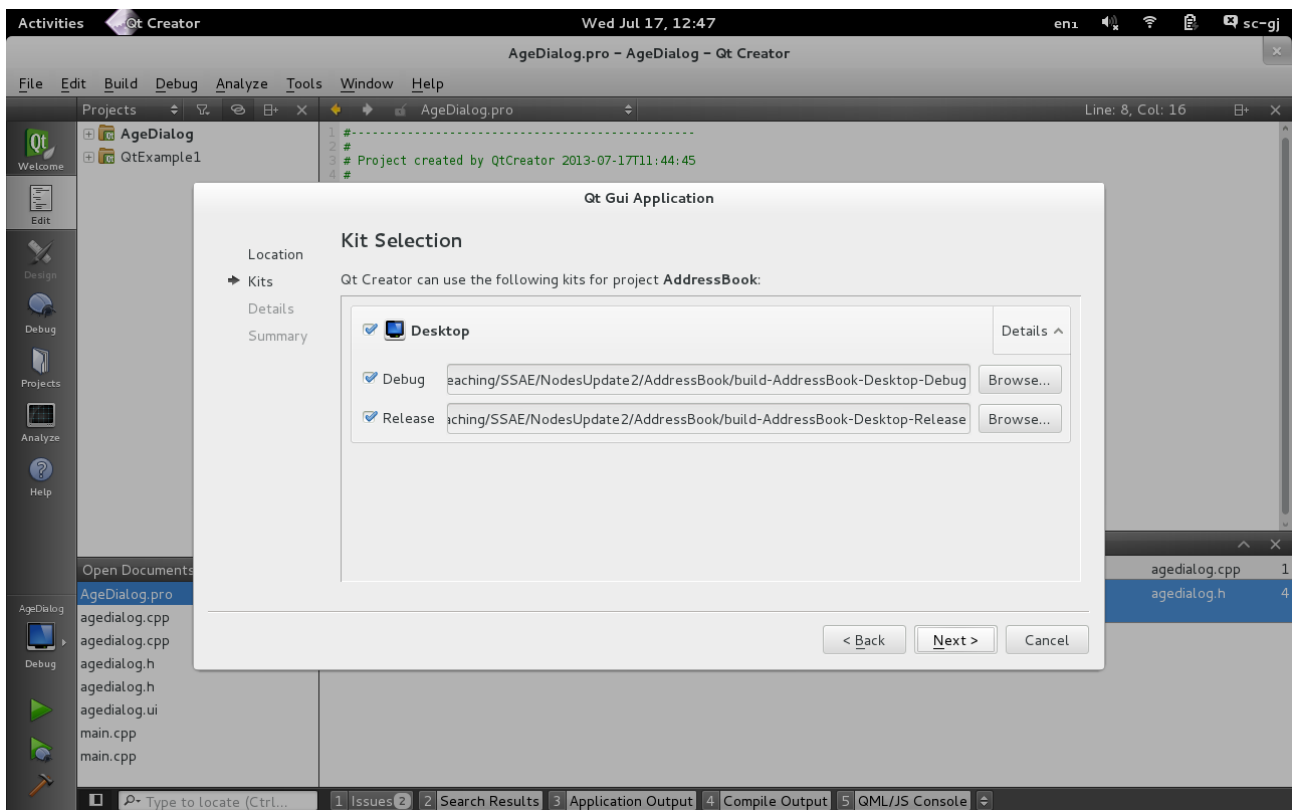
** - adapted again in 2012 , well re-written for QtCreator

Creating a Skeleton Qt Application

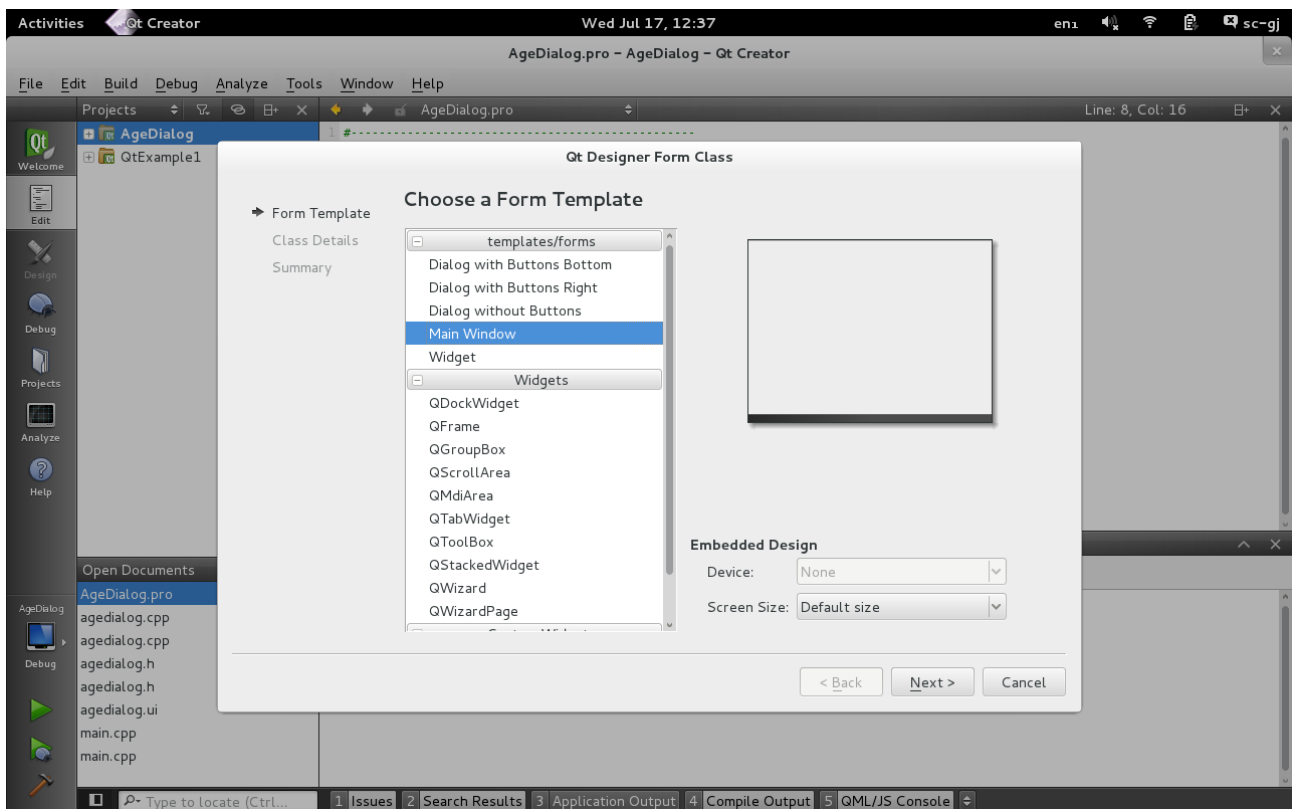
The first step is to create a trivial project. To do so, invoke the **New Project** dialog in QtCreator name the project as before.



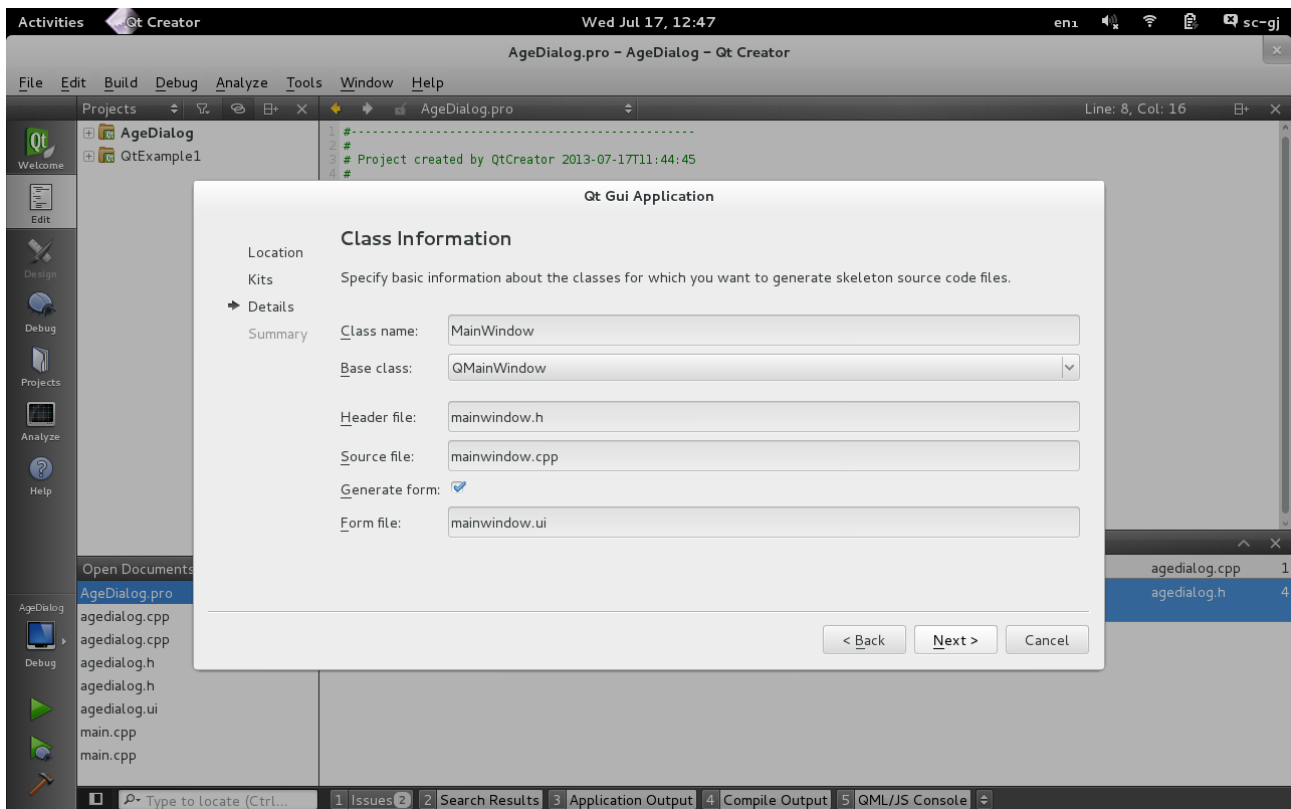
Setup the kit, this should be configured if not see above.



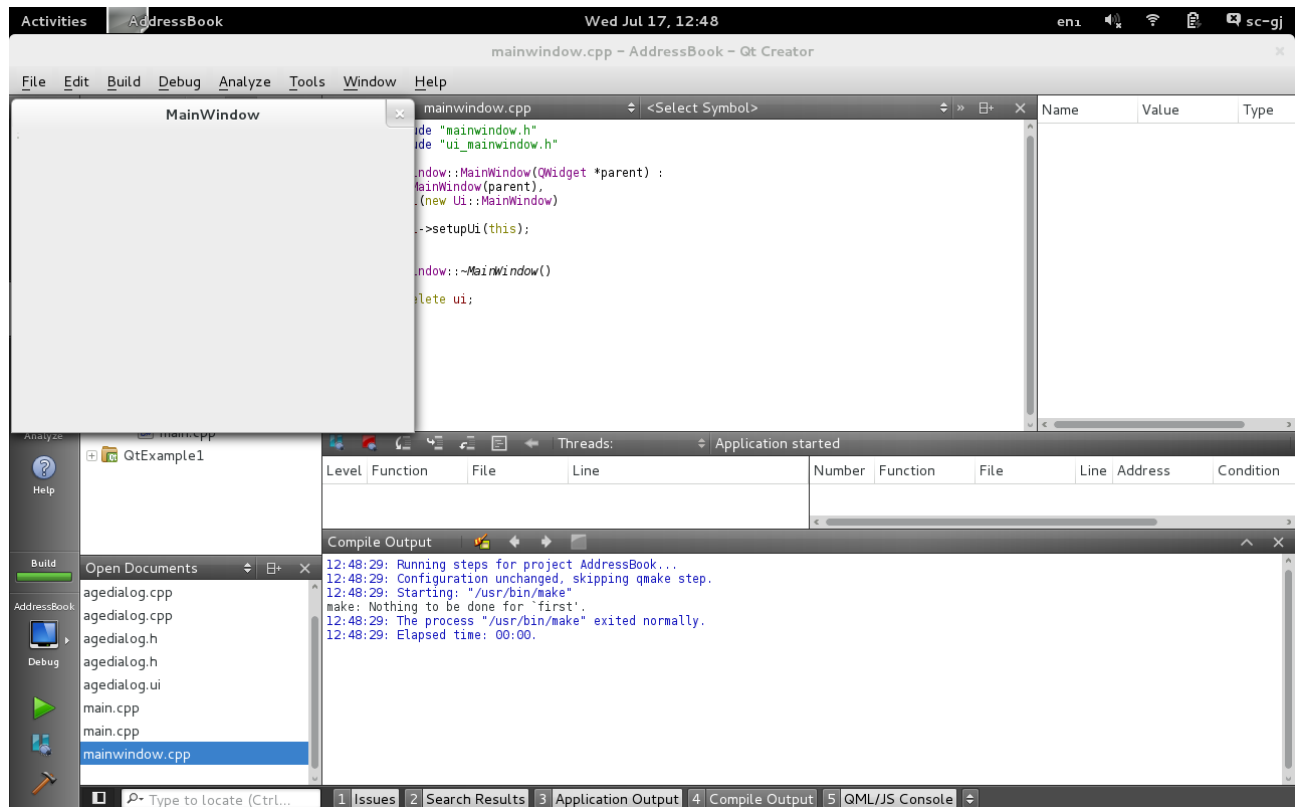
The



The class information includes the name of the class, files to be created and the project to be subclassed. In this case we're using QMainWindow.



The project can be compiled. Go to the **Build** menu and run qmake, clean and build.. After a successful build, the "AddressBook" application can be launched.

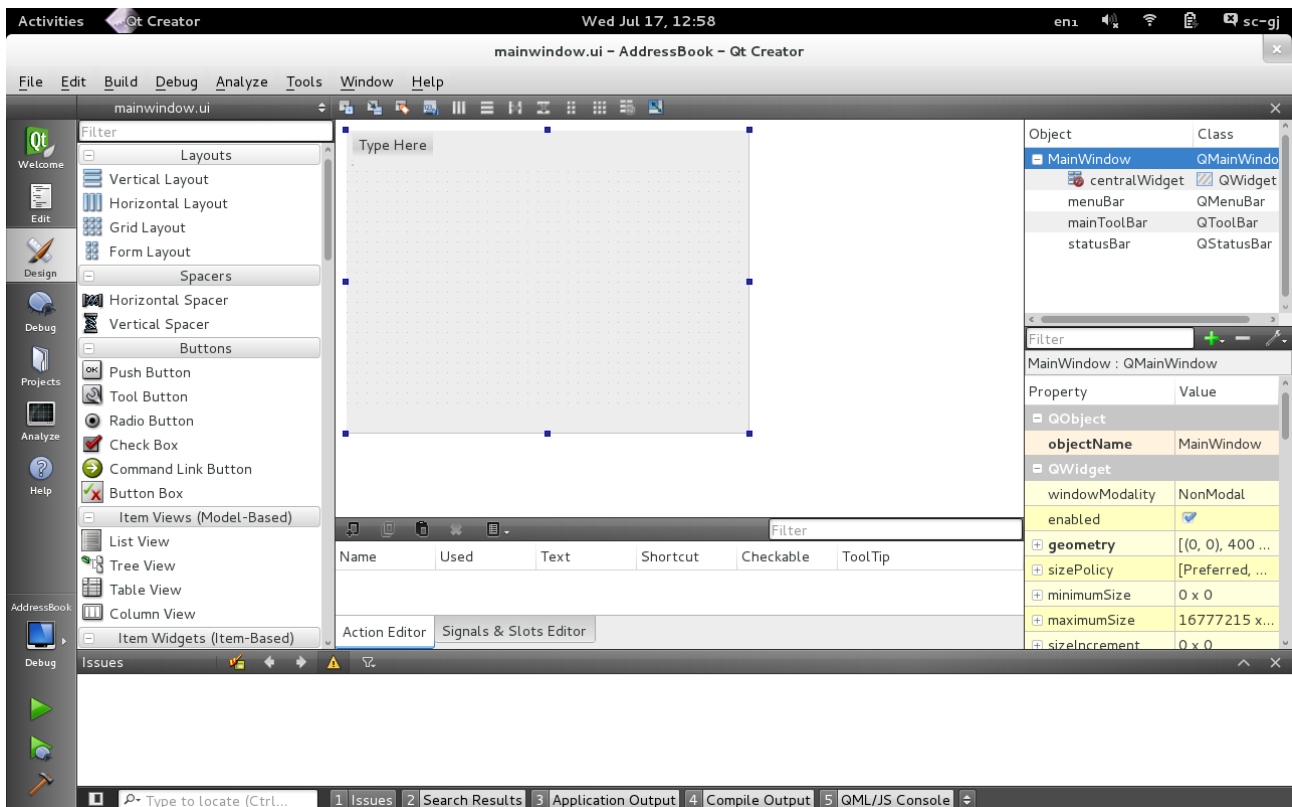


Now you should see the "AddressBook" application which shows only an empty window so far. Close it by clicking the X button in its title bar.

Designing the Main Dialog

In this section, we'll design the application's main dialog using the integrated *Qt Designer* visual form editor. This is only a brief guide for additional refer to the [Qt Designer manual](#) for more information.

To get this going open the forms folder in the project folder and double click on the AddressBook.ui file.



TODO – Finish this.

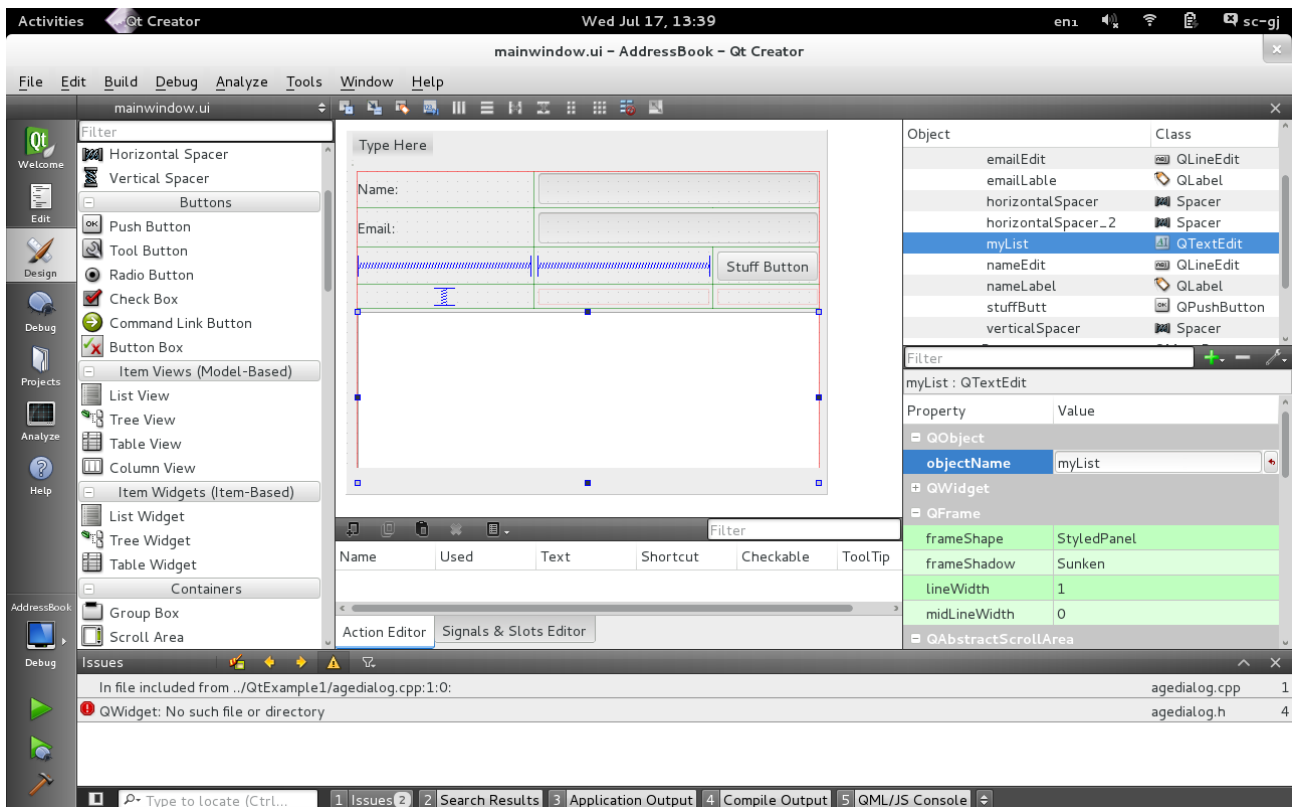
We will start by adding the widgets and setting their properties. Then we will put them into layouts. The screen-shot below represents the finished product don't rush off and build this just yet we'll build it up in stages introducing the designers features as we go.

//Make a simpler version – just a few boxes and fields etc. Connect one button etc.

This is essentially just a simple application with a few buttons, most of this is only dragging and dropping stuff. The process of adding things is simply one of dragging from the toolbox on the right hand side to the centre, a quick summary of what we're going to add.

Widget	objectName	Text
QLabel	"nameLabel"	"Name:"
QLabel	"emailLabel"	"Email:"
QLineEdit	"nameEdit"	" "
QLineEdit	"emailEdit"	" "
QPushButton	"stuffButt"	"Stuff Button"
QTextEdit	"myList"	" "

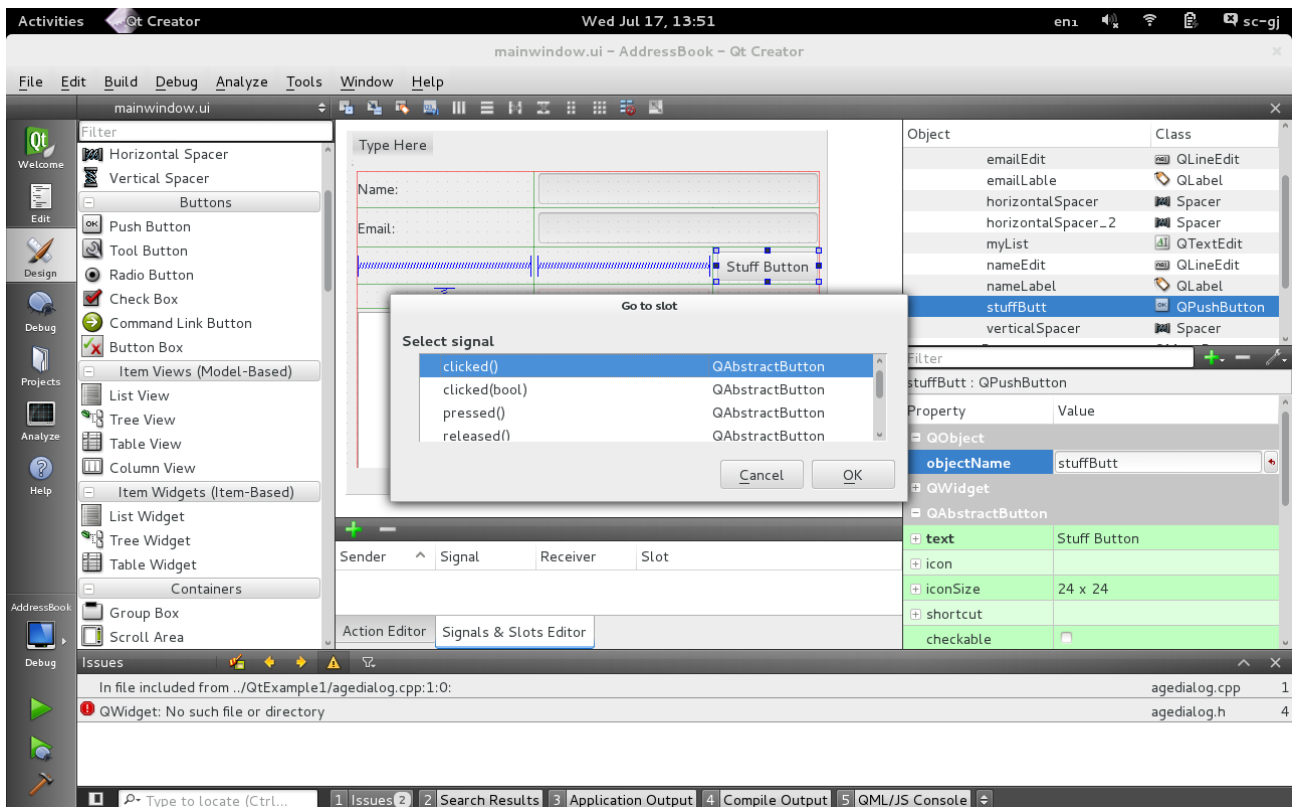
The application will just put the name and email into a QTextEdit. To make things neater I'm going to add a layout first. I won't patronise you by explaining the process of dragging and dropping and changing the properties of objects (properties window is on the right hand side). Its important to add the layout first (we're using a grid layout) then add the components above to the grid layout, this includes some horizontal and vertical spacers – I won't explain these but all you need to do is drag them on.



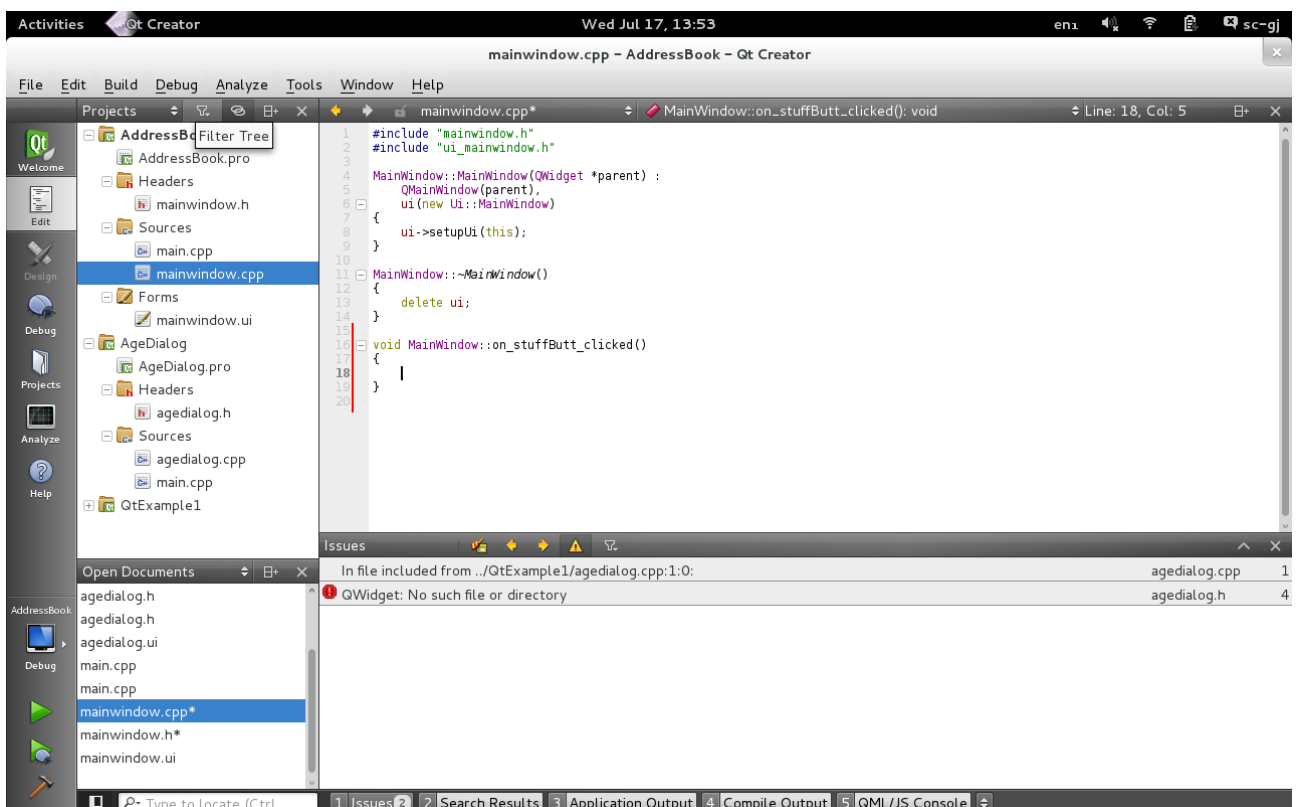
In the above you can see the little application we've knocked up so far. The properties editor is shown on the bottom right while the top left shows the objects in the UI and their types.

There is no magic to what is going on here, we run `qmake` and build the application. QtCreator simply writes a huge load of C++ code and stuffs it into a class in a UI package called `AddressBook`. We can then reference these in our code as `ui::componentName`.

The only thing which is a little strange is the generation of signals and slots, essentially the button signal is attached to a slot which we must provide. Previously this was something you just did and hoped you remembered the weird syntax is used for the name, in the latest version of Qt Creator is more like visual studio (or even *spits* VB). We essentially right click the button and select "Go To Slot ..." from the context menu.

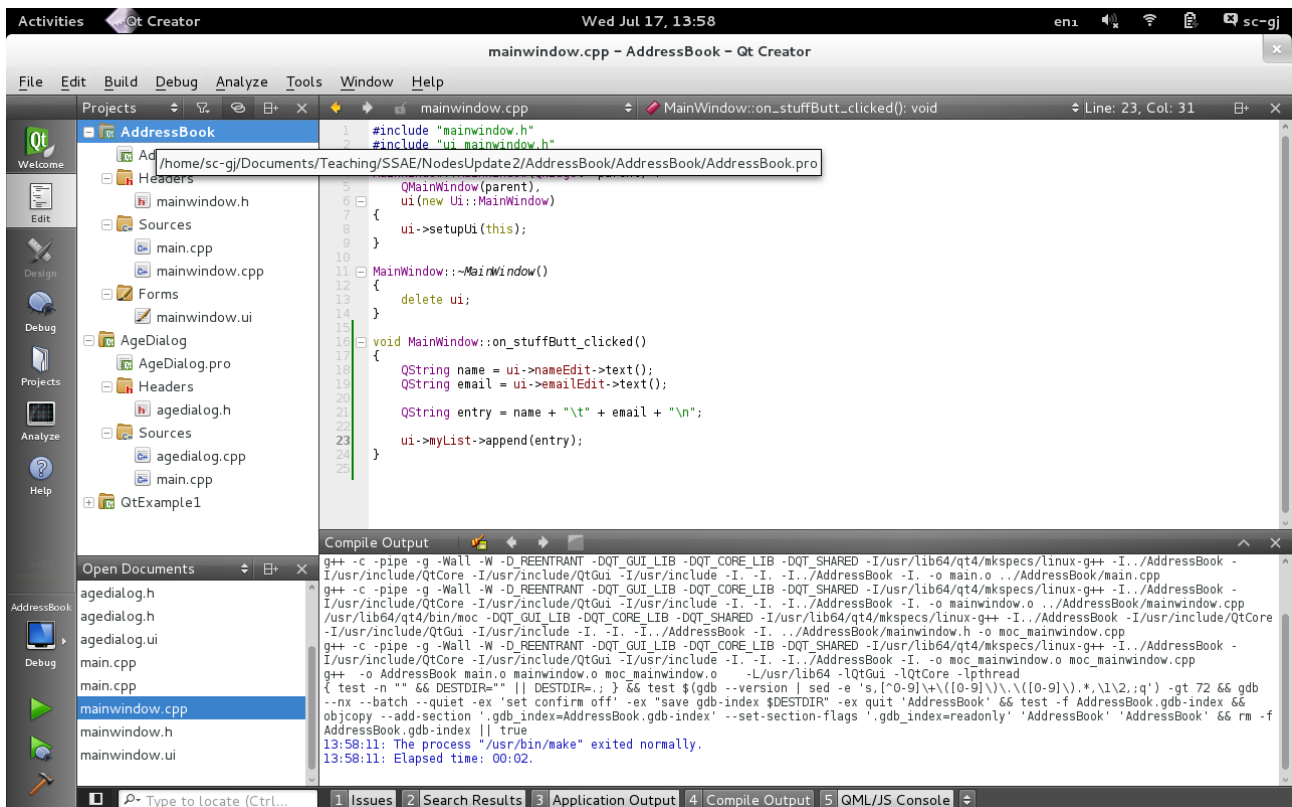


This takes us to the Slot in our class which will be linked to the buttons signal.



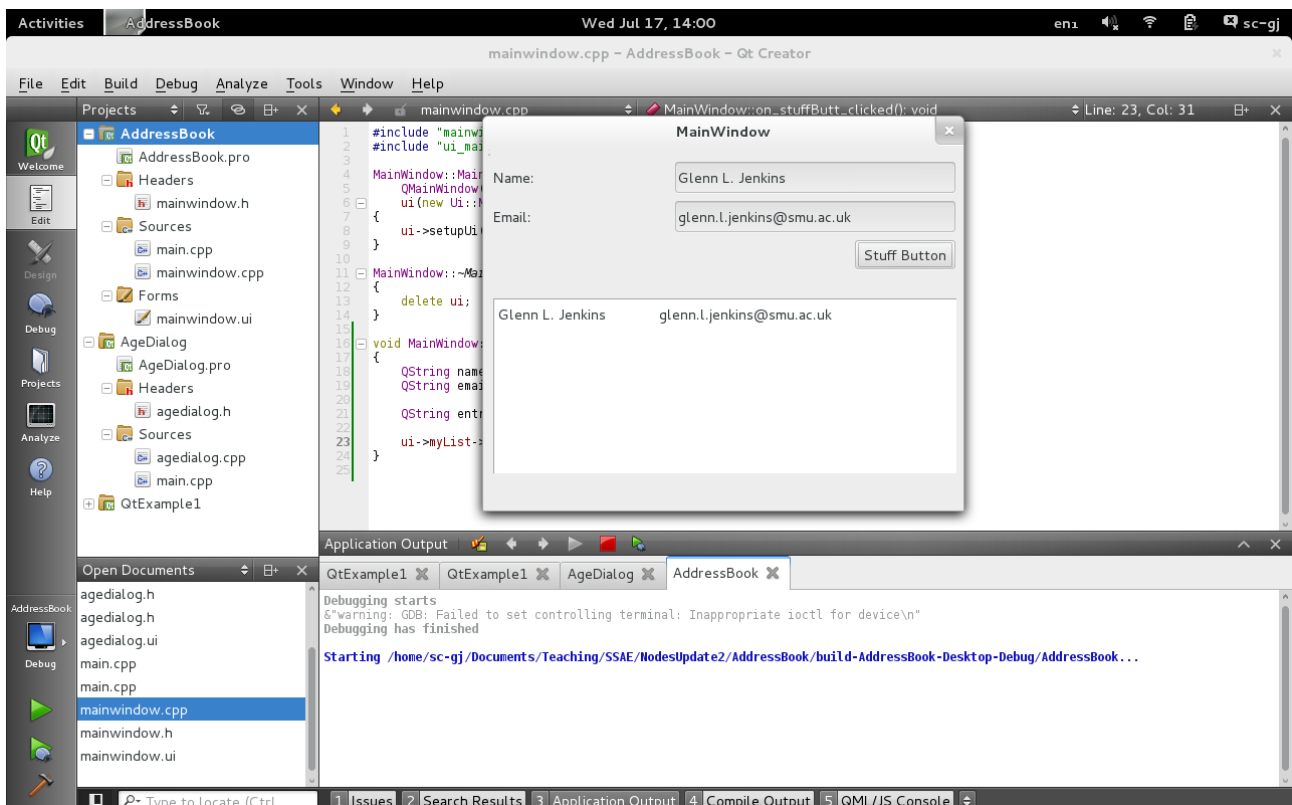
* The definition has also been added to the header file.

To complete our application we add some code to take the content of the email and name lineEdits and add them to the myList textEdit.



Compiling the program with the logic added, shouldn't give any errors. Though sometimes an out of date make file or old object file can cause issues. In such cases re-run qmake, clean the project and build.

Running the application gives:



Final Observations

This is a minimal example, I had hoped to do more but time constraints limit what I can give you. Everything we can do at the command line can be done using the tools, in most cases much faster. This is the purpose of IDE and designer type tools, to aid the code writing process. They should never replace it, if your only adding to code you don't understand how will you diagnose problems?

Below is the header file for the ui class generated by QtCreator, note that it simply does all the stuff we normally do ourselves.

```
/
*****
** Form generated from reading UI file 'mainwindow.ui'
**
** Created by: Qt User Interface Compiler version 4.8.4
**
** WARNING! All changes made in this file will be lost when recompiling UI file!
*****
/

#ifndef UI_MAINWINDOW_H
#define UI_MAINWINDOW_H

#include <QtCore/QVariant>
#include <QtGui/QAction>
#include <QtGui/QApplication>
#include <QtGui/QButtonGroup>
#include <QtGui/QGridLayout>
#include <QtGui/QHeaderView>
#include <QtGui/QLabel>
#include <QtGui/QLineEdit>
#include <QtGui/QMainWindow>
#include <QtGui/QMenuBar>
#include <QtGui/QPushButton>
#include <QtGui/QSpacerItem>
#include <QtGui/QStatusBar>
#include <QtGui/QTextEdit>
#include <QtGui/QToolBar>
#include <QtGui/QWidget>

QT_BEGIN_NAMESPACE

class Ui_MainWindow
{
public:
    QWidget *centralWidget;
    QWidget *gridLayoutWidget;
    QGridLayout *gridLayout;
    QTextEdit *myList;
    QSpacerItem *horizontalSpacer;
    QLabel *nameLabel;
    QPushButton *stuffButt;
    QSpacerItem *verticalSpacer;
    QLabel *emailLable;
}
```

```

QLineEdit *nameEdit;
QSpacerItem *horizontalSpacer_2;
QLineEdit *emailEdit;
QMenuBar *menuBar;
QToolBar *mainToolBar;
QStatusBar *statusBar;

void setupUi(QMainWindow *MainWindow)
{
    if (MainWindow->objectName().isEmpty())
        MainWindow->setObjectName(QString::fromUtf8("MainWindow"));
    MainWindow->resize(478, 361);
    centralWidget = new QWidget(MainWindow);
    centralWidget->setObjectName(QString::fromUtf8("centralWidget"));
    gridLayoutWidget = new QWidget(centralWidget);
    gridLayoutWidget->setObjectName(QString::fromUtf8("gridLayoutWidget"));
    gridLayoutWidget->setGeometry(QRect(10, 0, 461, 311));
    gridLayout = new QGridLayout(gridLayoutWidget);
    gridLayout->setSpacing(6);
    gridLayout->setContentsMargins(11, 11, 11, 11);
    gridLayout->setObjectName(QString::fromUtf8("gridLayout"));
    gridLayout->setContentsMargins(0, 0, 0, 0);
    myList = new QTextEdit(gridLayoutWidget);
    myList->setObjectName(QString::fromUtf8("myList"));

    gridLayout->addWidget(myList, 4, 0, 1, 3);

    horizontalSpacer = new QSpacerItem(40, 20, QSizePolicy::Expanding,
QSizePolicy::Minimum);

    gridLayout->addItem(horizontalSpacer, 2, 0, 1, 1);

    nameLabel = new QLabel(gridLayoutWidget);
    nameLabel->setObjectName(QString::fromUtf8("nameLabel"));

    gridLayout->addWidget(nameLabel, 0, 0, 1, 1);

    stuffButt = new QPushButton(gridLayoutWidget);
    stuffButt->setObjectName(QString::fromUtf8("stuffButt"));

    gridLayout->addWidget(stuffButt, 2, 2, 1, 1);

    verticalSpacer = new QSpacerItem(20, 40, QSizePolicy::Minimum, QSizePolicy::Expanding);

    gridLayout->addItem(verticalSpacer, 3, 0, 1, 1);

    emailLable = new QLabel(gridLayoutWidget);
    emailLable->setObjectName(QString::fromUtf8("emailLable"));

    gridLayout->addWidget(emailLable, 1, 0, 1, 1);

    nameEdit = new QLineEdit(gridLayoutWidget);

```

```

nameEdit->setObjectName(QString::fromUtf8("nameEdit"));

gridLayout->addWidget(nameEdit, 0, 1, 1, 2);

        horizontalSpacer_2 = new QSpacerItem(40, 20, QSizePolicy::Expanding,
QSizePolicy::Minimum);

gridLayout->addItem(horizontalSpacer_2, 2, 1, 1, 1);

emailEdit = new QLineEdit(gridLayoutWidget);
emailEdit->setObjectName(QString::fromUtf8("emailEdit"));

gridLayout->addWidget(emailEdit, 1, 1, 1, 2);

MainWindow->setCentralWidget(centralWidget);
menuBar = new QMenuBar(MainWindow);
menuBar->setObjectName(QString::fromUtf8("menuBar"));
menuBar->setGeometry(QRect(0, 0, 478, 27));
MainWindow->setMenuBar(menuBar);
mainToolBar = new QToolBar(MainWindow);
mainToolBar->setObjectName(QString::fromUtf8("mainToolBar"));
MainWindow->addToolBar(Qt::TopToolBarArea, mainToolBar);
statusBar = new QStatusBar(MainWindow);
statusBar->setObjectName(QString::fromUtf8("statusBar"));
MainWindow->setStatusBar(statusBar);

retranslateUi(MainWindow);

QMetaObject::connectSlotsByName(MainWindow);
} // setupUi

void retranslateUi(QMainWindow *MainWindow)
{
    MainWindow->setWindowTitle(QApplication::translate("MainWindow", "MainWindow", 0,
QApplication::UnicodeUTF8));
        nameLabel->setText(QApplication::translate("MainWindow", "Name:", 0,
QApplication::UnicodeUTF8));
        stuffButt->setText(QApplication::translate("MainWindow", "Stuff Button", 0,
QApplication::UnicodeUTF8));
        emailLable->setText(QApplication::translate("MainWindow", "Email:", 0,
QApplication::UnicodeUTF8));
    } // retranslateUi

};

namespace Ui {
    class MainWindow: public Ui_MainWindow {};
} // namespace Ui

QT_END_NAMESPACE

#endif // UI_MAINWINDOW_H

```