Simple Dialog Based Qt Application

Tutorial

By

Adnan Zejnilovic

April 16, 2015

COP 2335

Create a Project

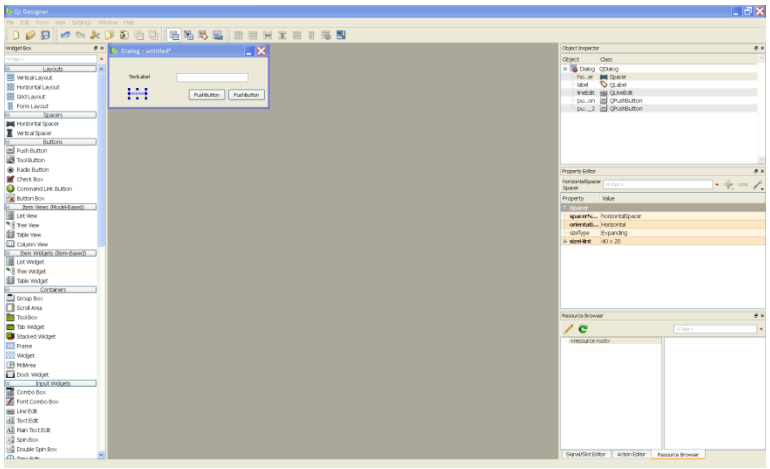
Launch Qt Creator and create an empty project. Create it in directory of your choice.

Design the UI

Launch Qt Designer (Qt>Tools>Qt Designer). Choose Widget. The first step is to create the child widgets and place them on the form:

1. From Qt Designer's widget box drag:
    a. one label,
    b. one line editor,
    c. one horizontal spacer, and
    d. two push buttons.

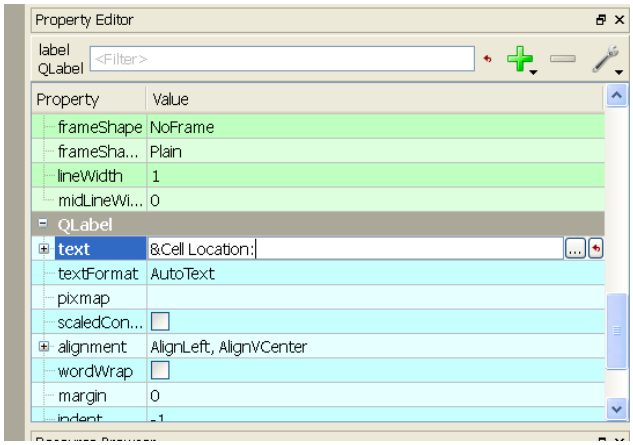
    Drop the widgets roughly where they should be positioned on the form.
2. Now drag the bottom of the form up to make it shorter. This should produce a form that looks like this:



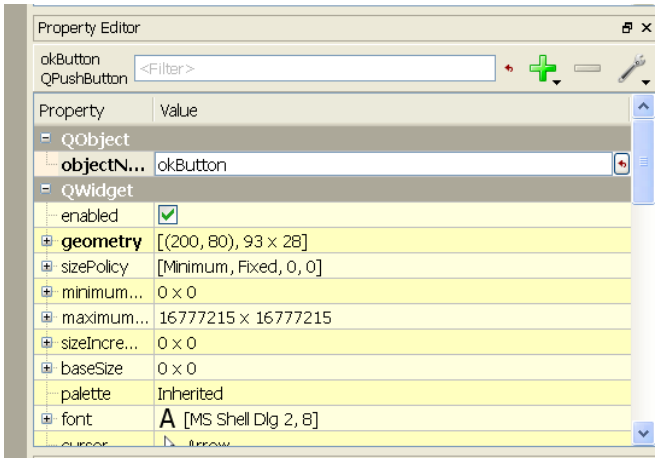
Don't spend too much time positioning the widgets on the form; Qt's layout managers will lay them out precisely later on.

Set Properties

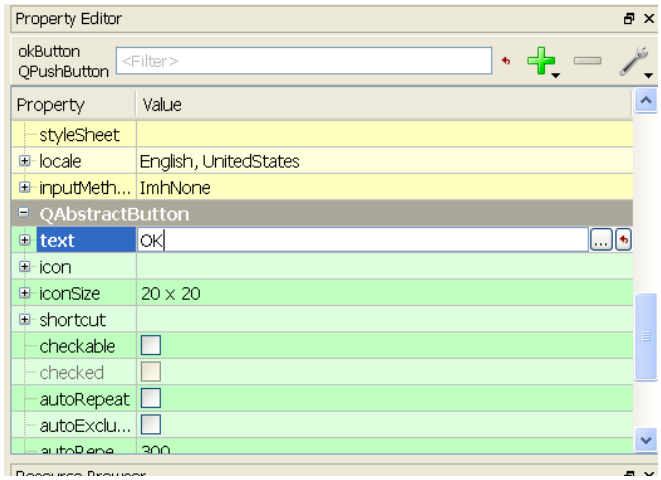
Next, click on the text label. Make sure that its text property is set to “&Cell Location:”



Click the first push button. Make sure that its objectName property is set to “okButon”:



Set the text property of the okButton to “OK”:



Similarly, set the second push button name property to cancelButton and thetext property to “Cancel”.

Lastly, click on the dialog's form background to select it. Then set the objectName property to "GoToCellDialog" and the windowTitle property to "Go To Cell".

All the widgets look fine now, except the text label, which shows &Cell Location.
Click Edit|Edit Buddies to enter a special mode that allows you to set buddies.
Next, click the label and drag the red arrow line to the line edit control, then release the left mouse button. The label should now appear as Cell Location, and have the line editor as its buddy. Click Edit|Edit Widgets to leave buddy mode.

Set Layouts

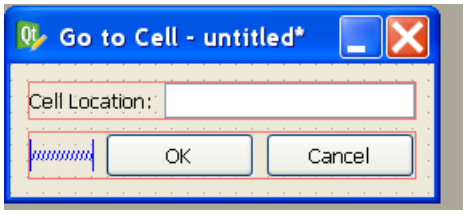
The next step is to lay out the widgets on the form:

1. Click the Cell Location label and press and hold Ctrl key as you click the line editor next to it so that they are both selected. Click Form|Lay Out Horizontally.

2. Click the spacer, then hold Ctrl key as you click the form's OK and Cancel buttons. Click Form|Lay Out Horizontally.

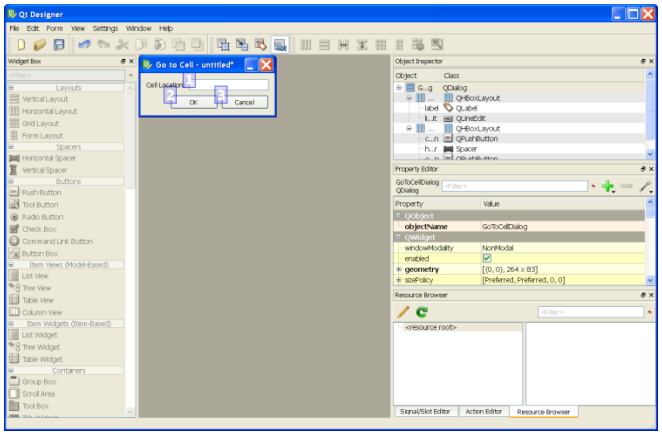
3. Click the background of the form to deselect any selected items, then click Form|Lay Out Vertically.

4. Click Form|Adjust Size to resize the form to its preferred size. You should end up with something like this:



Set the Tab Order

Click on the Edit>Edit Tab Order and set the tabs by clicking on the numbers until you have this:



To get out of the Set Tab mode, click on the Edit>Edit Widgets

Preview the Dialog

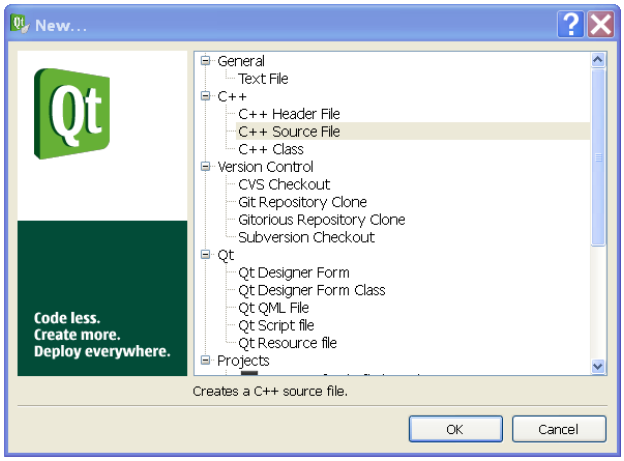
Click on the Form|Preview… to preview the dialog. Make sure that the tab order is working correctly by clicking the Tab key repeatedly. The focus should move from Line Edit to OK and then to Cancel in that order. Close the Preview dialog.

Save the file

Save the .ui file in your working directory.

Coding

Open the Qt Creator and click on File|New File or Project and Select C++ source file. You should get a screen like this (main.cpp):



When the code editor opens, type the following code:

```
#include <QApplication>
#include <QDialog>
#include "ui_gotocell.h"

int main(int argc, char *argv[])
{
   QApplication app(argc, argv);
   Ui::GoToCellDialog ui;
   QDialog *dialog = new QDialog;
   ui.setupUi(dialog);
   dialog->show();
   return app.exec();
}
```

Next we need to run qmake:

Go to DOS prompt, navigate to your working directory and execute the following:

- qmake –project
- qmake GoToCellDialog.pro

The qmake will dete the user interface file (.ui file) and invoke Qt's uic user interface compiler which in turn will convert the .ui file into C++ code resulting in a ui_gotcelldialog.h.

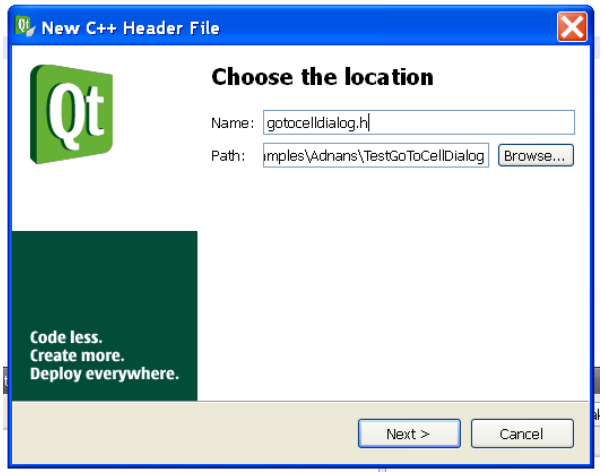
Open the ***gotocelldialog.ui*** and observe that the code is writtenh in the XML format. Next, open the generated ***ui_gotocelldialog.h*** and observe C++ code.

Build, Compile, and Run

At this point you can build and run your code. You should get a dialog box that does not react to user input. The line edit will take any input, whereas its intended input should be only Excel cell addresses. In order to enable the intended behavior, we need to write some code. However, this would involve writing Regular Expression validator, which is not complicated, but it is beyond the scope of this tutorial. Nevertheles, we still need to write SIGNAL and SLOT connectivity code.

Write Additional Code

To complete our tutorial, we are going to add a new header file to the project called gotocelldialog.h:



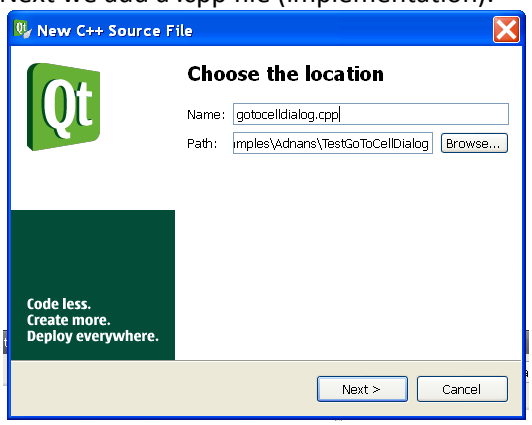
Next, we will add the following code:

```
#include <QDialog>
#include "ui_gotocelldialog.h"

class GoToCellDialog: public QDialog, public Ui::GoToCellDialog
{
    Q_OBJECT
public:
    GoToCellDialog(QWidget *parent = 0);
private slots:
    void on_lineEdit_textChanged();
};
```

The IDE will provide **#ifndef, #define** and **#endif**. Notice that the class inherits from both QDialog and Ui. Thus we have a case of ***multiple inheritance.*** Q_OBJECT is a built in macro that represents the base class of all Qt objects. Recall that in MFC CObject was the base class so this is the same principal.

Next we add a .cpp file (implementation):



Next, we implement code for the SIGNALS and SLOTS. We purposely omitted RegularExpression checking of the input as this is beyond the scope of our Qt coverage. For those of you that wish to learn more about that, you can look up QRegExp and QtRegExpValidator class.

In any case, **gotocelldialog.cpp** code is as follows:

```
#include <QtGui>
#include "gotocelldialog.h"

GoToCellDialog::GoToCellDialog(QWidget *parent)
    : QDialog(parent)
{
    setupUi(this);

    connect(okButton, SIGNAL(clicked()), this, SLOT(accept()));
    connect(cancelButton, SIGNAL(clicked()), this, SLOT(reject()));
}

void GoToCellDialog::on_lineEdit_textChanged()
{
    okButton->setEnabled(lineEdit->hasAcceptableInput());
}
```

Note the inclusion of the <QtGui>. Also, we call setupUi() to initialize the form. Sounds familiar? Remember InitInstance()?.

Last, we connect okButton to accept() QDialog's slot and cancelButton to QDialog's reject() slot. Both functions will close the dialog box but accept() sets the dialog's box result to value to1 and reject() to 0 so we can make a decision in our programming logic accordingly.

Rewrite the main.cpp

Finally, we need to make a few changes to main.cpp:

```
#include <QApplication>
#include <QDialog>
#include "gotocelldialog.h"


int main(int argc, char *argv[])
{
        QApplication app(argc, argv);
        GoToCellDialog *dialog = new GoToCellDialog;
        dialog->show();

        return app.exec();
}
```

At this point, you can compile and run your application. This concludes this tutorial.