Quick Start

from 17. Juli 2012

Here it comes the first small example written in Basic For Qt® by you. All you need to do is to follow the following steps and read the explanations. It will give you some feeling about writing in Basic For Qt® and Qt – a whole new world to explore.

You will need some time to do the task. Plan to have time for about an hour before you have completed this quick start.

Let's go

Start Basic For Qt® by double clicking on the shortcut icon on the desktop, or by clicking your icon on the taskbar.

Create a new project

Select from the menubar *File -> New Project...* Enter a short name for your project in the displayed new project window.

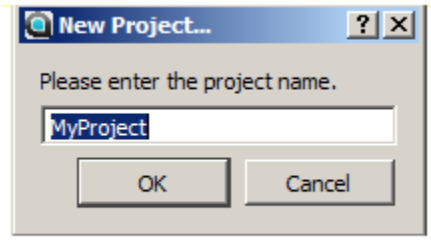

Fig 1

Don't use / . or other special characters. What about the name *Quick Start*?

After pressing OK, a new skeleton project will be created for you (normally on the Desktop of the current user).

The Main Q7Basic Screen

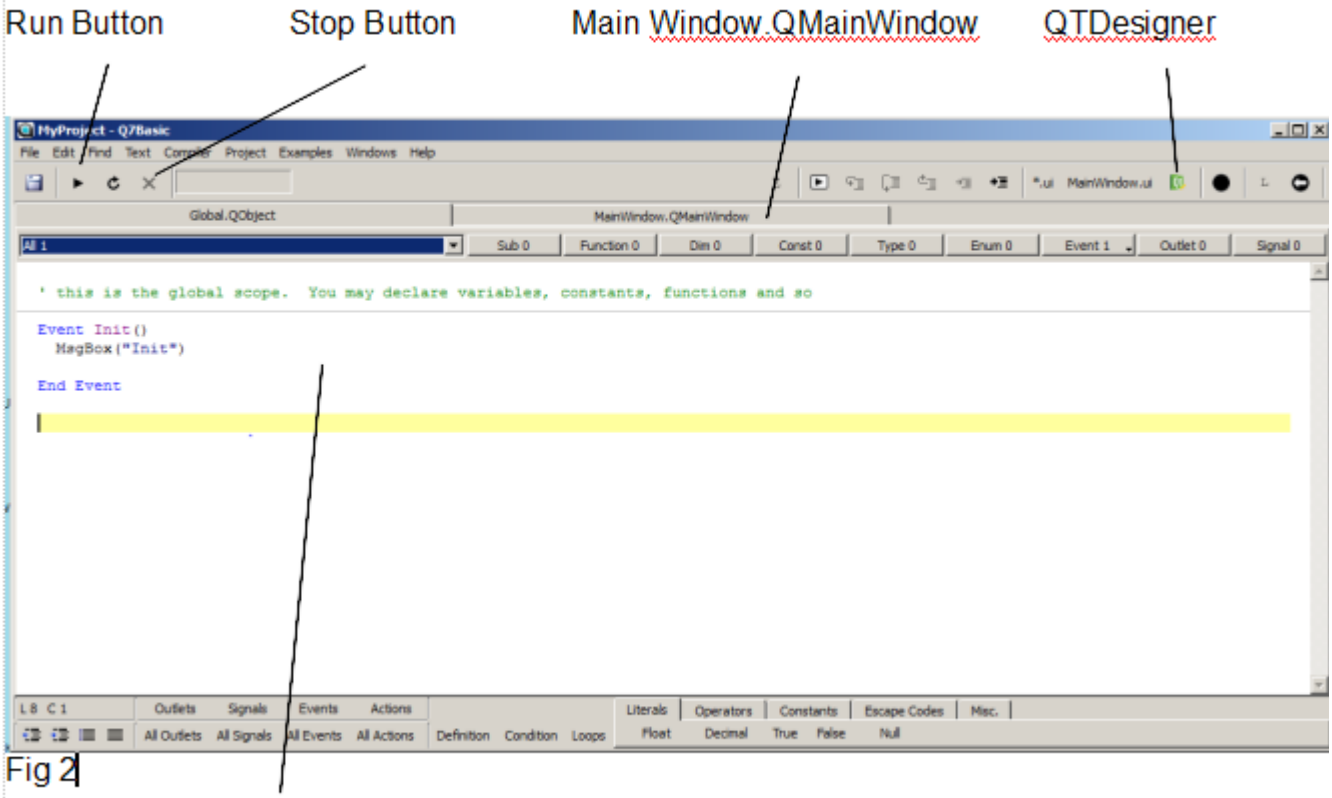

Fig 2

The file **Global.QObject** (normally just called **Global**) is a special file created automatically for you with the special procedure **Event Init()**.

.

The Global.QObject file is actually the place where you may declare variables, constants, functions and so on, which may be used as global elements of your program without the need of an object creation before. It is similar to a module in VB. The **Event Init()** is called on startup of your application. You may freely add code to it. But we won't do much with it in this small tutorial.

First start of your application

Anyway, let's just run this small example and take a look what happens when you hit the run button (in the tool bar of the Basic For Qt® IDE or in the menu bar *Compiler -> Run)*.

You probably noticed the command MsgBox**("Init")** in the Event Init(), which makes the window appearing with the texts **Init** after you started your example.

Stopping your running example

When you hit the Stop button in the toolbar or in the menubar *Compiler -> Stop,* your application will immediately stop execution and return to the Basic For Qt® screen..

First c hange of your application

Let us change the "**Init"** to something more meaningful like "Hello World!". To do so change the **Init** in the MsgBox("Init!") to **"Hello World!"** between Event Init and End Event. Run the example again, if you don't know how, see the paragraph above. Now, **Hello World!**,
appears. Hey, you just changed your first application in Basic For Qt®. Congratulations!

Windows and Buttons

Time to get a more sophisticated way to get a window on screen. Now, I want you to start **Qt Designer** to create a new main window, with a button on it. The button will change the button's title when clicking on it and show the new text when it is pressed.

Qt Designer is our friend. To start it, make the code file **MainWindow.QMainWindow** the current file by clicking on it (its code must be visible, see image above) then select from the menubar *Project -> Qt Designer For Current File or click on the Toolbar Qt Designer Icon.*
Qt Designer gets loaded with the default GUI file MainWindow.ui.

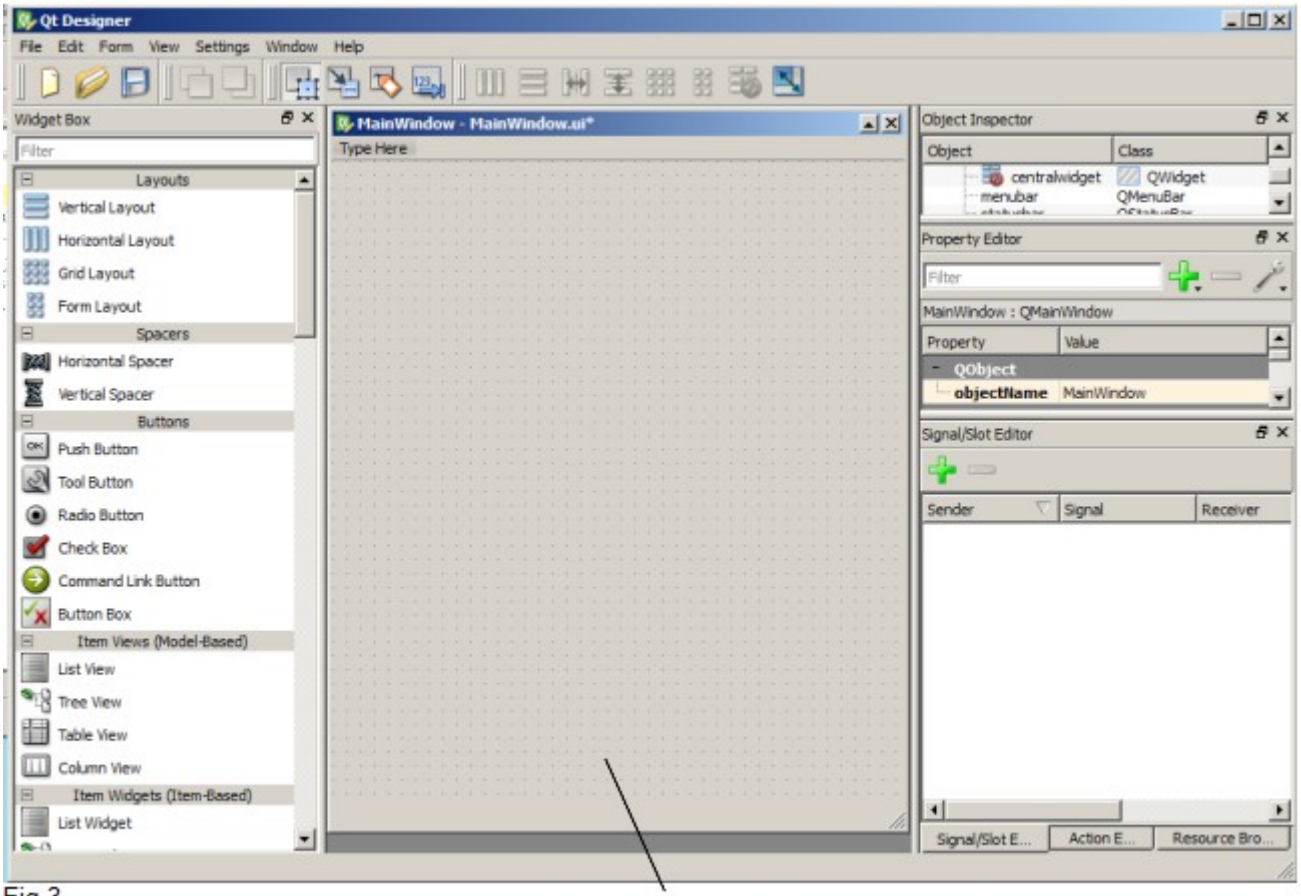

Fig 3

Main Window Form

Qt Designer

To get a quick overview about Qt Designer, read the documentation provided for it, called "Qt Designer" selectable in the menubar in the help menu of the Basic For Qt® IDE.
You are now in Qt Designer.

New Window

After you have opened the Qt Designer, the main window form will appear as an empty window on screen ready to by extended by dropping controls on it as shown.

New QPushButton

Place a `QPushButton` on the window by dragging & dropping it to the new created window. (Press the left button on the Push Button and draw your mouse onto the window and stop pressing the mouse button.)

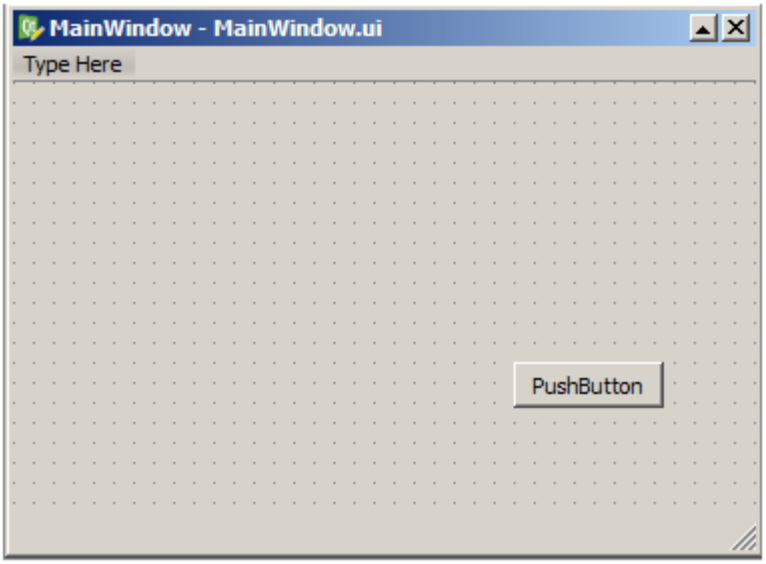

Fig 4

Basic usage of Qt Designer

In Qt Designer you draw your windows and controls, but you do not code there (do not forget to give all objects meaningful object names).

Next Step

Save the file opened in Qt Designer by selecting it from the menu bar of Qt Designer : *File -> Save.*

Now, you have created a window and a button on that window, you need to tell the Basic For Qt® IDE what has to be done with them. But before that, it is time to run your example again.

Close the Qt Designer. You will now be back into the IDE of Basic For Qt®. Select run again. What happens? Your message Hello World! will appear again, but after clicking on OK, your newly created window with the button will appear. Great! You managed to get a custom Qt based Window to be shown on your screen.

Writing code for the window

We need a place to write code for the window. **Code is always written in the IDE of Basic For Qt®.** We do not need to create a new source code file in the IDE of Basic For Qt®, because MainWindow.QMainWindow can be used.

(If you need a new code file, you can create it by selecting in the menubar Project -> New File With Super Class -> QObject... enter *a* name.

In the **MainWindow.QMainWindow** add a new line:

```
Outlet pushButton As QPushButton ' if your button is not named pushButton change
it!
```

Where you add this line is regardless. It declares a variable used for a button in a ui file, which can be accessed and used in source code. Within your project, it is used for the button in the newly created window in the ui file.

Create the following lines as well.

```
Signal on_pushButton_clicked(Checked As Boolean) ' if your button is not named
pushButton, change it!
   pushButton.Text = "Just changed the title on the PushButton”
End Signal
```

Your MainWindow.QMainWindow should now look like:

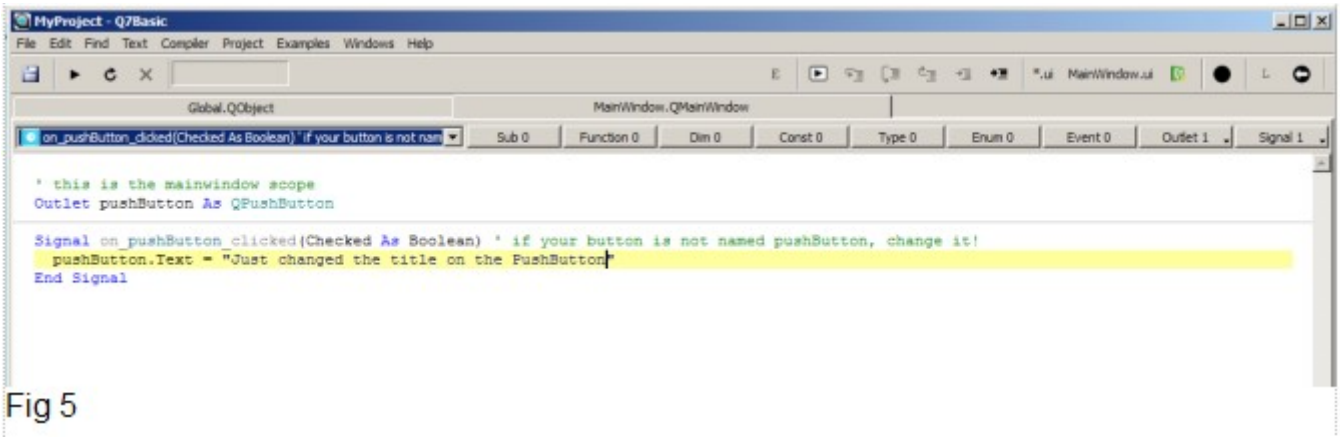

Fig 5

The event follows the form Signal on_..._clicked(Checked As Boolean), where ... stands for the object name set in Qt Designer for that button.

It creates an event procedure used for Qt Designer objects, which is then executed when the pushButton is clicked, when the event is triggered (in our case when the button is pressed).
The text on the pushButton should change from “PushButton” to “Just changed the title of the PushButton”

Do you need to tell Qt Designer about your created Outlet and Signal?

Truly, you do not, because it is automatically done for you. The only thing you need to keep in mind, is to name your objects meaningfully (set the the object name of every control).

We are nearly finished with your example, the last task we need to do is to connect the Outlet written in the source code file with the button created in the window.

In the IDE of Basic For Qt® **build & run your example**. After the first message box is shown, press on the button appearing on the window screen and the title of the button should change after being clicked.

This completes our first program.

An Enhanced Simple Example

In this second example, we will make use of the Widget properties. We will enhance and add to our window, labels, textboxes, and buttons to create our window.

Lets Go Again

The first stage in the programming cycle is to come up with an idea of what you wish to produce. Together, let us build a Loan Calculator. The First stage of programming is to draw our potential window screen layout on paper, so, here is our sketch:

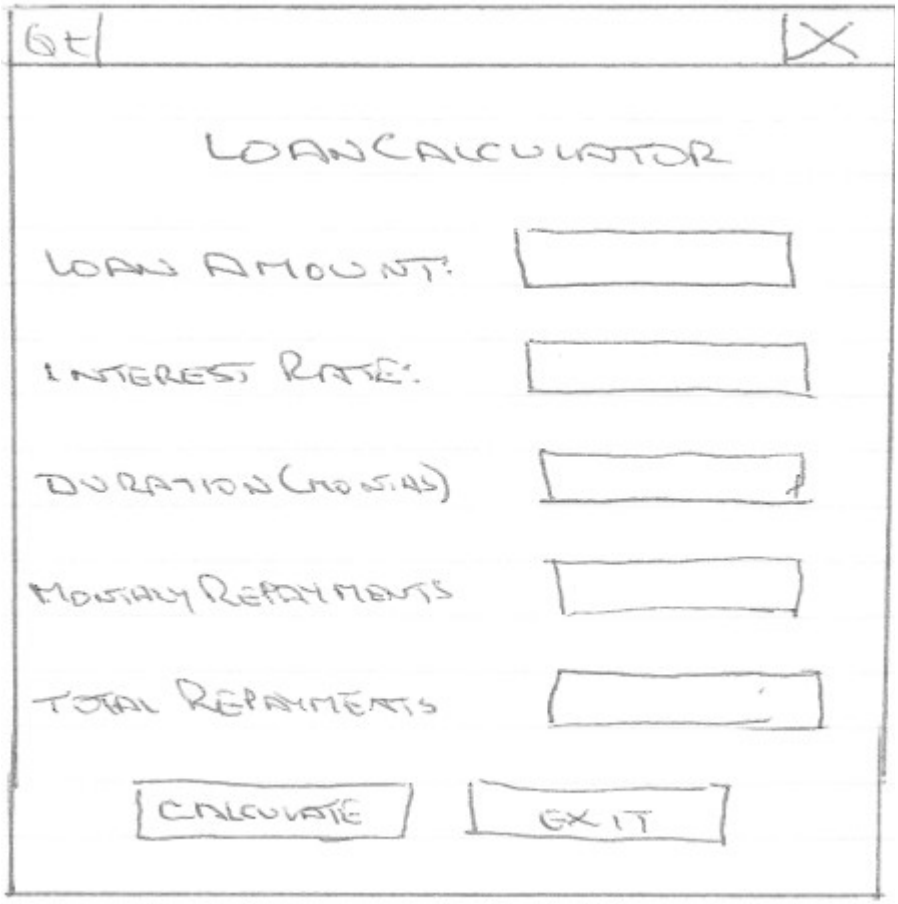

Fig 6

Start up Basic For Qt®, create and name a new project, See Figs 1 and 2. This project is named LoanCalculator. Ensure Global.QObject is selected (See Fig 2) there you can see the line **MsgBox("Init")**, comment out this line as shown in the line above. Start Qt Designer by clicking on the toolbar icon. The Qt Designer will be loaded as per Fig 3.

This is where the fun begins. Now, follow the steps below:

If you examine the Qt Designer, you will that on the left hand side you have a window containing the Qt Designer Widgets, in the centre we have our **default basic window**, this can be changed to suit your type of project, we will stay with the default, to the right we have three stacked windows boxes, the top

one is named Object Inspector, the middle one is named Property Editor and the bottom one is named Signal/Slot Editor. You can adjust the height of these window to suit your needs. Simply place your cursor over the join of any two windows, the cursor will change to a double headed arrow, drag the arrow up or down to size your window boxes. I reduced Signal/Slot Editor and expanded Object Inspector. We are now ready to start and produce a fully working practical program.

We now need to transfer our rough sketch to our window in Qt Designer.

This is stage 2 of our programming cycle. The Design Stage.

Every widget in the Qt Designer is listed in the window on the left, each widget has a number of properties which are shown in the Property Editor. We can change these properties to give meaningful names to our widgets, set the text on labels, pushButtons etc etc. Always examine the properties of any control you use and get an understanding of what you can do within the Qt Designer, but remember, you can also change these properties from your code in Basic For Qt®.

Let start with our basic window. The window comes with a Menu Bar and a Status Bar built into the form. We will not use these in our small sample program.

Click on the window to ensure it is the active widget. The default name of MainWindow can remain, or you can change it to MainCalc, the choice is yours.

The title of the window requires to be changed to indicate what the program is about. Scroll down the window properties until you come to **windowTitle**, change this to **Loan Calculstor,** this immediately changes the title on the window.

The size of the current window is a little to large for our Loan Calculator. Scroll down the properties window and find **geometry** property. This displays the current size of the window, by clicking on the x in the box on the left you can expand the property window as shown in the figure below:

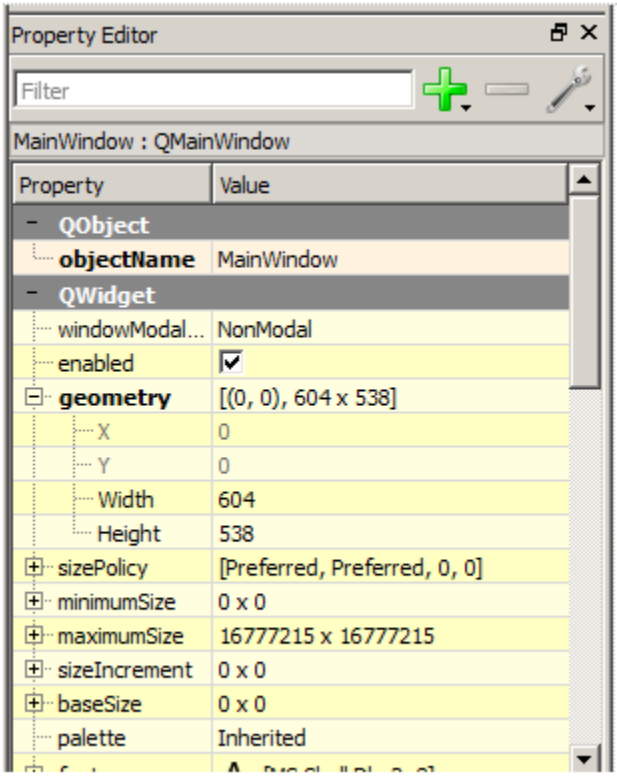

Fig 7

The easiest way to resize the form is by using the mouse, looking at the bottom left of our window, move the window scroll bars so that you can see the bottom right hand corner of the window, then place the cursor on the bottom right hand corner and then move the mouse slowly towards the top left corner of the window, at the same time keeping your eye on the **geometry** properties and size the form to approximately Width = 365 and Height = 445. We can make further adjustments after we have placed our widgets on the form.

We now need to add to our window 7 x Labels, 5 x Line Edits, and 2 Push Buttons.

Scroll down the widgets and drag a label widget to the window and release the mouse button, now, click on the label, notice the little boxes, these are sizing handles and also indicate the widget is active, now press **CTRL + C**, this will copy the label widget to the clipboard, move the cursor away from the label widget and click the window, now press **CTRL + V**, a new label will be added to the form. Do this **CTRL + V** another 5 times and you will have 7 labels on the window.

As the labels will only be used to identify the lineEdit boxes on the window, other than the label that will contain our title there is no requirement to change the other six default names. Arrange the labels as follows: One in the centre and near the top of the window, change the text property to **Loan Calculator** and the font to size 14 and Bold. Return to the Qt Designer and change the name of the label to **lblTitle.** Click Save and return to **Basic For Qt®.** Click the title label and resize it so that all the text is displayed and reposition centrally. Arrange the remaining 6 labels down the left side of the window. (See Fig 6) above.

Add a Line Edit and using **Ctrl + C** to create another 4 lineEdits and line up your lineEdits to your

labels. Add 2 Push Buttons.

Name your Line Edits and Pushbuttons. Our sample used the following names:

ledAmount

ledRate

ledDuration

ledPaymentsDue

ledRepayments

btnCalculate

btnExit

Your completed screen should be similar to Fig 8:

Notice We have used a three digit code to identify the type of widget, i.e. led = lineEdit, chk = CheckBox and btn = Pushbutton, it is up to each of you to determine how your controls are named. Whatever method you choose, just be consistent.

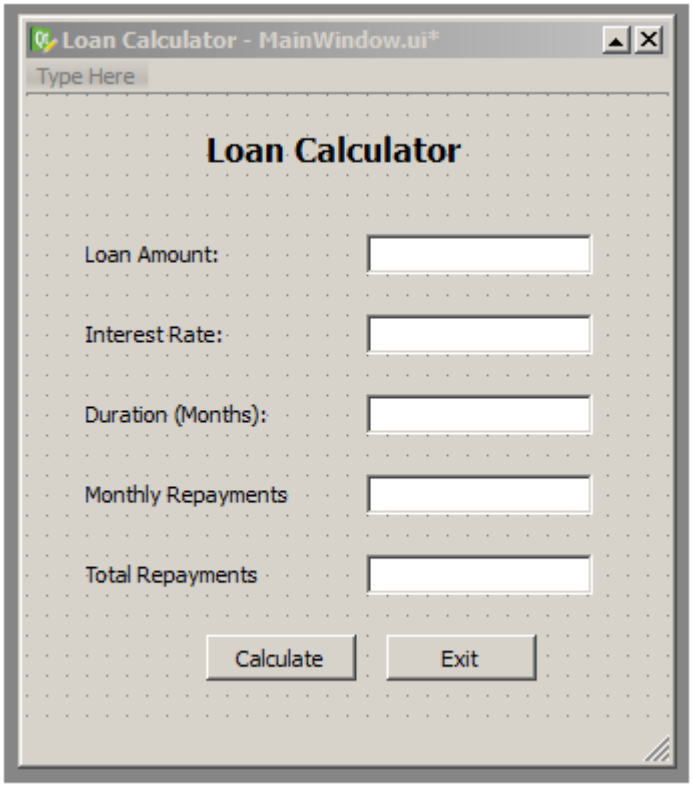

Fig 8

There is one more important step we must complete before we leave the QT Designer.

Our window only has a few controls, but imagine a full sized window with perhaps twenty or more widgets on the window. Widgets may also be contained in various Group Boxes. It is essential that the use of the **Tab Key** on the keyboard will move through any Group Boxes and then the widgets within the Group Box in a correct and logical sequence, This is called the **Tab Order**. Look at the Toolbar in the Qt Designer at icon number eight from the left:

Fig 9

This is the **Tab Order** icon. Click on this icon and it will display the current tab order of the controls on the form. Ensure your control tab sequence is the same as shown in Fig 10 below by clicking on and adjusting the value of the number:

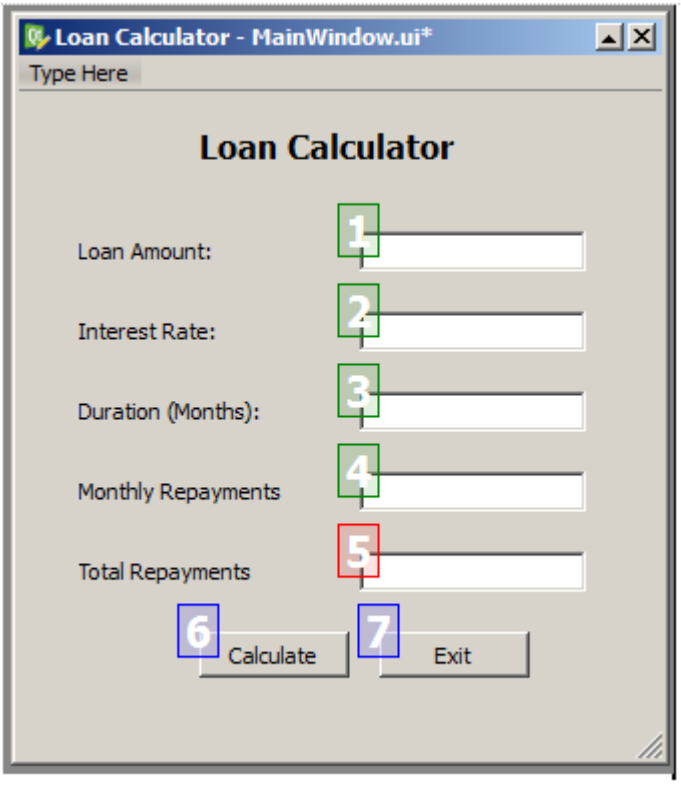

Fig 10

We have now early completed stages 1 and 2 of our programming cycle. Before we leave Stage 2, we should ensure that our form will display correctly. Select all the widgets on the window by placing the curser near the top left corner of the window without touching a widget and depress the left mouse button and move the curser across and down the screen to encompass all the widgets in the window. When all the widgets are selected, release the mouse button and using the direction keys of the keyboard you can move all widgets and position them centrally and vertically on the window.

Now, to check how our window will display on the screen at run time, click on the window once more to make it the active widget then press **CTRL + R.** This will display your window as it will be seen at run time. Close the window by clicking the **X** in the top right corner or by pressing the escape key. Save your work in the St Designer. Now, before you leave the Qt Designer, we strongly recommend that you write down the name you have given to every widget on your window and if necessary any properties you may wish to change from within **Basic For Qt®.**

Before we close the Qt Designer let us give a little thought to other types of program's we might write in the future. What about a small learning program for young children. Would they be interested in a Grey/Silvery screen with white text boxes and all words in Black.

No. They would like to see lots of colours and shapes. So let us look quickly at how we might add a little colour to our current program. It would be nice to have the label title displayed in Red. Perhaps the pushbuttons in Blue with White text. Maybe the window background in colour or with an image.

Did you notice when examining the widget properties that there is no Background, ForeColor or Color property. The only properties I noticed was the Palette and StyleSheet properties. Let us quickly examine them.

Open the Qt Designer, click on the label widget that is currently displaying the title of **"Loan Calculator"**, we have already made this a larger sized text and made it Bold. Select the Palette property and double click on **Change Palette** This will display the following dialog:

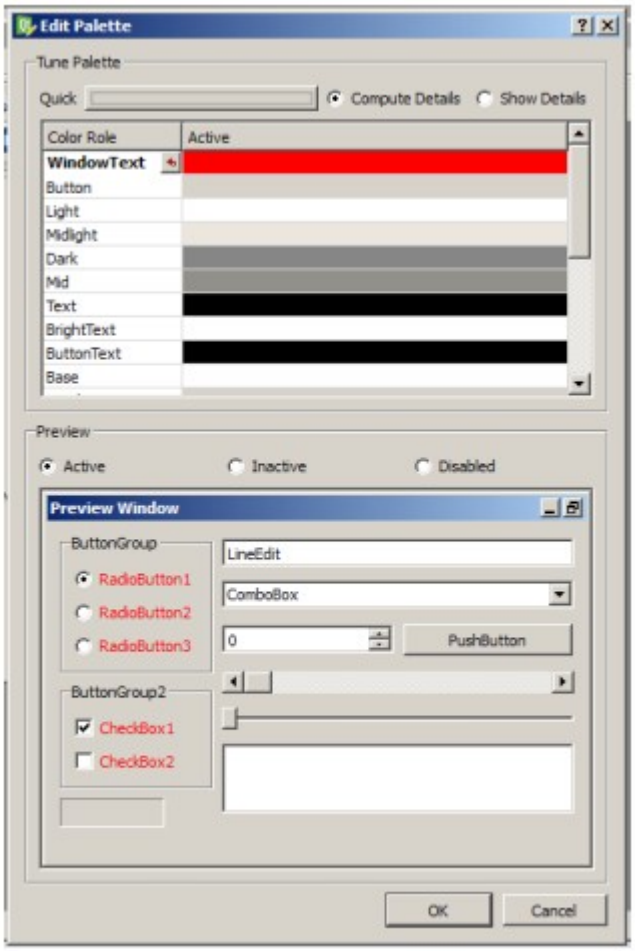

Fig 10(a)

If you look at this line on your computer, you will see the top line Color Role is Black. To change it to **RED,** double click the Active column and select your color from the Select Color dialog window displayed. Tip. Scroll down the Color Role and examine the widgets you can change.. Remember select your widget on youe window first.

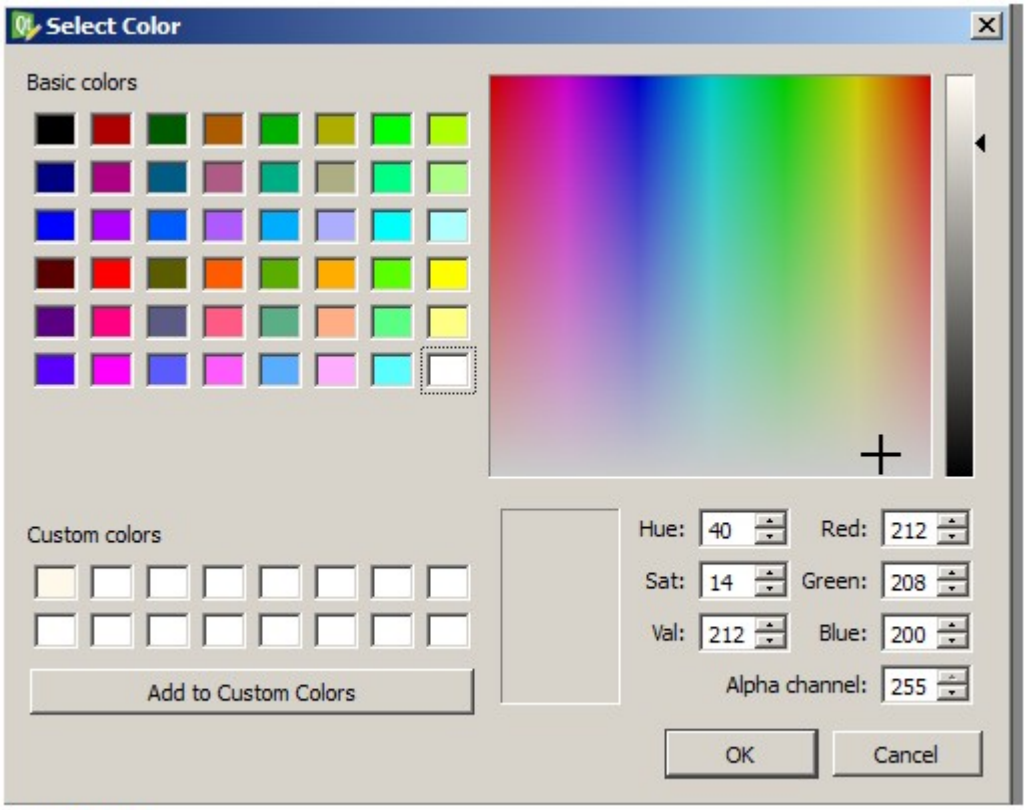

Fig 10)b)

You should now examine the options in the Color Palette.
The exercise to color your pushButtons and or your window background is now in your hands. This is our finished window:

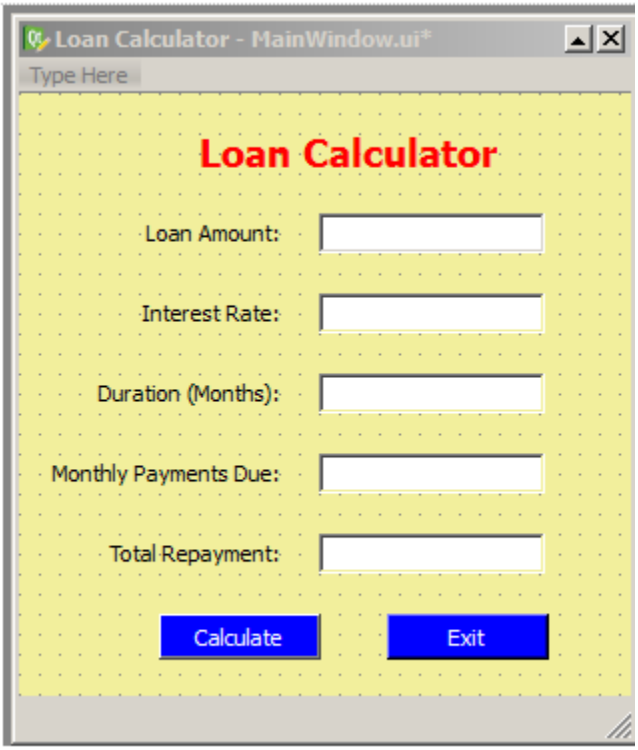

Fig !0(c)

In Basic For Qt®, save your project and then click compile and run your program. Your form should be displayed. Stage 2 in now complete.

Stage 3 is the actual coding to make our interface work from within **Basic For Qt®.** However, there are a few things we must consider before we start.

You will need to understand that declaration will be needed for every control that you wish to use within in your code.

You will need to create and set the type of variables you will require.

You will need to validate the input typed or input into your program, to ensure that what is also input conforms to the variables you have created and of course you will need to know about the programming language you are using to carry out this coding, **Basic For Qt®.**

We will endeavour to inform you step by step as we develop our code for the Loan Calculator program. Perhaps one caveat I, we must mention and will also demonstrate is to use plenty of comments explaining your code. Comments are not compiled and do not slow down your code. I, we have written things in the past and then forgotten about them. At some point in the future you may need to resurrect a program you wrote months or years ago and you may not remember why you wrote it at the time, or why you wrote it the way you did. Comments will help and any other programmer to understand your code.

So, where do we begin: We have a number of lineEdits, and a checkbox that we need to access in our program, code. These must be declared. We declare these items in the MainWindow.QMainWindow.

Refer back to Fig 5 for a brief look. Then ensure that MainWindow.QMainWindow is active by clicking it on the button bar. Now enter the following into your code area:

```
Outlet btnCalculate As QPushButton
Outlet btnExit As QPushButton
Outlet ledAmount As QLineEdit
Outlet ledRate As QLineEdit
Outlet ledDuration As QLineEdit
Outlet chkPayEarly As QCheckBox
Outlet ledPaymentsDue As QLineEdit
Outlet ledRepayments As QLineEdit
Outlet lblTitle As QLabel
```

Notice that we have used the property names we created in the Qt Designer. See the expanded Object Inspector for a full listing of widgets on our window.

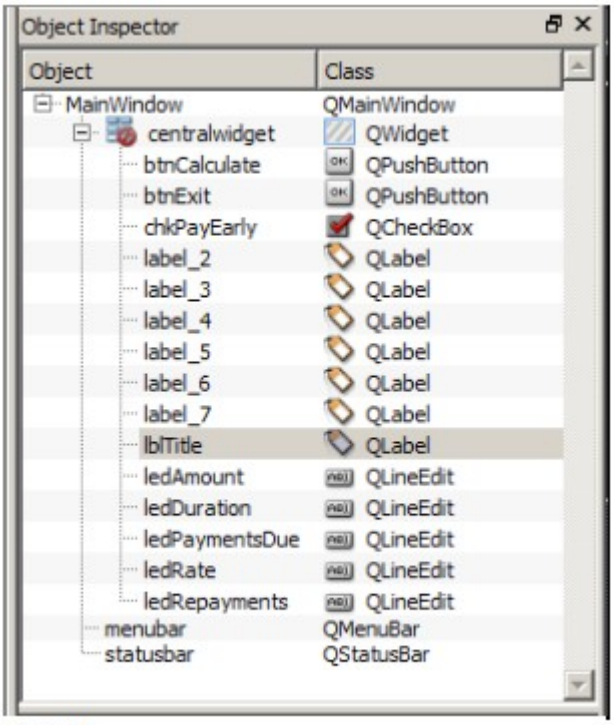

Fig 11

We have two pushButtons on our window. We must create the event's to activate these buttons. An event for a widget is as follows:

```
Signal  on_widgetName_clicked(Checked As Boolean)

 'Code goes here

End Signal
```

Create the events for our two buttons. Our code window should now look like this:

```
' this is the mainwindow scope

Outlet btnCalculate As QPushButton
Outlet btnExit As QPushButton
Outlet ledAmount As QLineEdit
Outlet ledRate As QLineEdit
Outlet ledDuration As QLineEdit
Outlet ledPaymentsDue As QLineEdit
Outlet ledRepayments As QLineEdit

Signal on_btnCalculate_clicked(Checked As Boolean)

  'code goes here

End Signal

Signal on_btnExit_clicked(Checked As Boolean)

  'code goes here

End Signal
```

Fig 12

If you ran your program now you would not see any changes or have any events to perform. So let us add some code to one of our buttons. To keep things simple we will add code to the btnExit to close our program from the pushButton.

Add **Application.Quit()** between Signal and End Signal. Now run your application and click on the Exit button. You have just programed your first event in **Basic For Qt®**. You could if you wish to do so display a message box asking the user if he is sure he wants to quit. Add the following code above the line Application.Quit:

MsgBox("Closing Down", vbYesNo, "Are you sure you wish to Quit")

```
Signal on_btnExit_clicked(Checked As Boolean)
  Application.Quit()
End Signal
```

Fig 13

We now need to add some variables to our code. A variable is used to store information that will be required by the program. Variables are normally allocated to hold a specific type of data, i,e. A string or various types of numeric data, images, records etc. If you are not aware of the variable types in **Basic For Qt®** you will need to read the manuals provided.

The following variables will be required by the Loan Calculator code to perform its calculations and will be added to the btnCalculate code event.:

```
Dim LoanAmount As Double
Dim LoanRate As Double
Dim LoanDuration As Int32
Dim Payment As Double
Dim Totals As Double
Dim DueDate
```

When we enter text into a lineEdit box such as the one named ledAmount, any numeric values must first be converted from the string format to a numeric format before they can be transferred to the variable used to store the number for any calculations. We must also check that the correct type and format of the numbers is correct. This is called validating the user input..

This is the code for our first lineEdit box in the btnCalculate event:

```
Signal on_btnCalculate_clicked(Checked As Boolean)
Dim LoanAmount As Double
Dim LoanRate As Double
Dim LoanDuration As Int32
Dim Payment As Double
Dim Totals As Double

 ' Validate Loan Amount
 If CDbl(ledAmount.Text)Then
   LoanAmount = Val(ledAmount.Text)
 Else
     MsgBox("Input Error","Please enter a valid amount i.e. 10000.00")
 End If
```

Fig 14

The first line is a comment line.
The next five lines checks and converts the data entered into the text box, then assigns it to the variable LoanAmount if it is correct, else, it displays a message box asking you to enter a valid amount.

Let us add the remainder of the code for validating the other data to be input to the program which is very similar to what we have seen:

```
' Validate Loan Interest Rate
If CDbl(ledRate.Text)Then
  LoanRate = Val(ledRate.Text)
Else
    MsgBox("Input Error","Please enter a valid interest rate i.e. 4.8")
End If

 ' Validate Loan duration
If CInt(ledDuration.Text)Then
  LoanDuration = Val(ledDuration.Text)
Else
    MsgBox("Input Error","Please enter a valid number of months i.e. 36")
End If
```

Fig 14(a)

Once all our input data has been validated we can then perform the calculation and display the data values of the calculation:

```
 'If all data is correct proceed with calculation
  Payment = ((((LoanRate * LoanDuration) / 1200 + 1) * LoanAmount) / LoanDuration)
' Monthly repayments
ledPaymentsDue.Text  = String(Payment)
' Total repayments
ledRepayments.Text = String(Payment * LoanDuration)

End Signal
```

Fig 14(b).

Run the program, do not enter any data into the lineEdit textboxes, you should see the message boxes prompting you for your data, run again and enter data. Did you notice that you jumped from Duration (Monthly) to the Calculate button.

This is because the lineEdit textboxes Monthly Repayments and Total Repayments do not expect data to be input, they are display items only.

This was a deliberate design flaw in the design of the program to draw your attention that you must give careful thought to the design of your program sketch and the controls you use. So, what do we need to do now, we can go back and change the lineEdit textboxes to labels, or we can make the lineEdits textboxes read only, this is what we have done.

Go back into Qt Designer, click on the lineEdit textboxs and change the **ReadOnly** to **True** by entering a tick in the box.

You may have noticed that when we added some colour to our window there was no facility to colour the back ground of a lineEdit textbox. It would therefore be very nice if we could change the background colour of the two lineEdit boxes to a Light Grey.

The final piece of code we need to add to our program is to Format the displayed date in our two read only boxes.

Hopefully, you will now have a good chance of getting involved with **Basic For Qt®** and will be able to enhance your knowledge by reading the manuals provided and by joining and using the forum.

Copyright