MODULE 1: INTRODUCTION TO PROGRAMMING IN C#

TUTORIAL 3: WINDOWS FORM APPLICATIONS

Introduction

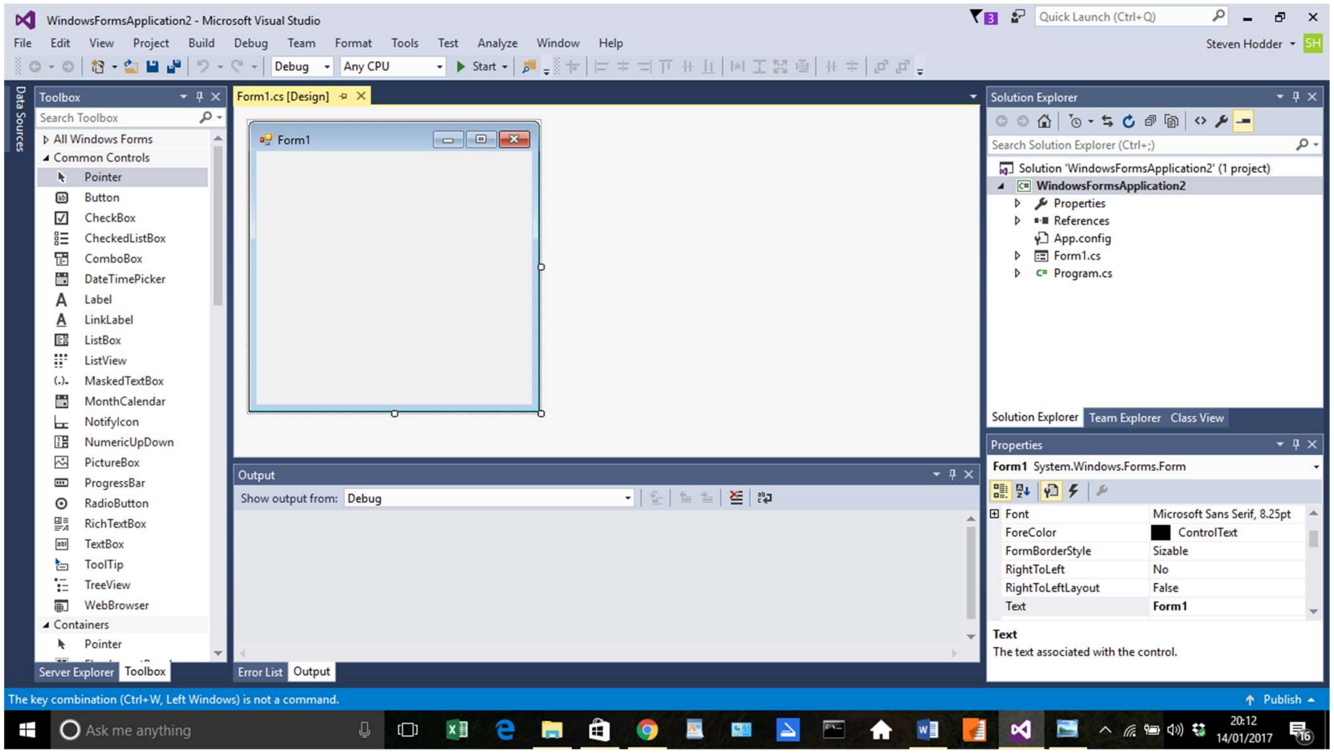
You should now have a reasonable familiarity with the C# language which you have gained by developing Console Applications. Now you are going to move on to develop typical Windows applications that involve forms, buttons, text boxes etc. These programs will require you to develop a new set of skills You will need to be aware of a simple form application where you use a single form that contains buttons and text boxes. This tutorial also considers program structure, scope, naming conventions, event-driven routines, and general good practice. We shall go on to develop applications where a form can show another form and so on.

Windows Form Applications are very interesting to develop but when you do so, think about the user – they may never have seen your form before and need to be guided through it. This is an important issue and we shall return to it later.

A ‘Quick Reference’ Guide is available that provides essential details of the most common controls that you will use.

Windows Form Applications

Open Visual Studio and click on File | New | Project and look under the options for C#. This time, click on Windows Form Application. Your screen should show:

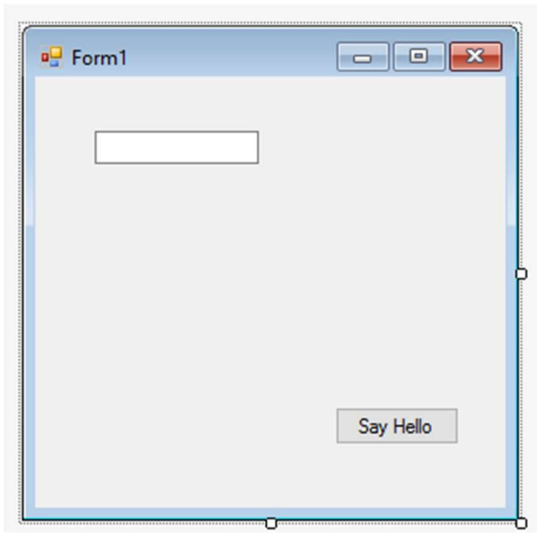


Font	Microsoft Sans Serif, 8.25pt
ForeColor	ControlText
FormBorderStyle	Sizable
RightToLeft	No
RightToLeftLayout	False
Text	Form1

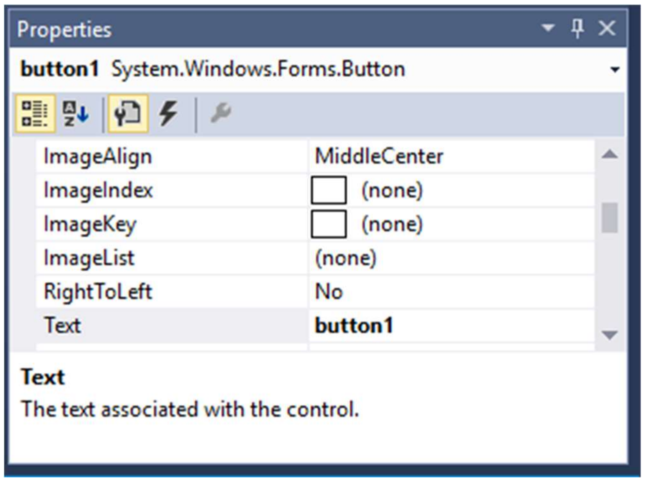
Text
The text associated with the control.

The key combination (Ctrl+W, Left Windows) is not a command.

Let’s start by creating the Hello World program as a Windows Form Application. Our first form will be very simple, using a button and a text box. It should look like:



To achieve this, look at the Toolbox list on the left hand side and first double-click on TextBox. This will create the required text box on your form that you can click and drag to the required position. Now do the same for a Button and move it down to the right. You now have 2 components in your form. By default, they will be called TextBox1 and Button1 – you will learn how to give them more meaningful names later. For now, we wish to change the text that appears in the button from ‘button1’ to ‘Say Hello’ (as shown above). Look at the bottom right of your screen. You should see a properties box for button1, as shown:


Properties

button1 System.Windows.Forms.Button

ImageAlign	MiddleCenter
ImageIndex	(none)
ImageKey	(none)
ImageList	(none)
RightToLeft	No
Text	**button1**

Text
The text associated with the control.

Click on ‘button1’ in the Text category and change it to ‘Say Hello’ and press ‘enter’. The text in your button should now be as required. Now we want to make the button do something! In this case, we want to display the text ‘Hello World!’ in our text box when the user clicks the button. Visual Studio makes this easy for us – simply double-click on the button and you will see a window containing:

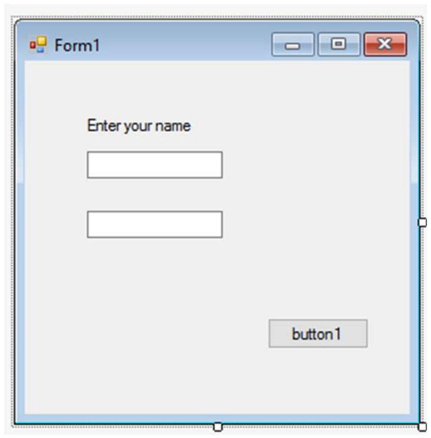
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WindowsFormsApplication4
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {

        }
    }
}
```

In order to make the button do something, we need to add some command to the button1. Change the above code to include:

```
        private void button1_Click(object sender, EventArgs e)
        {
            textBox1.Text = "Hello World!";
        }
```

Now run the program as before by clicking on the green arrow (start) button in the toolbar. Click on the button to show the ‘Hello World’ message in the text box. To close the program, click on the ‘stop button’ (redsquare) in the middle of the menu bar.

You can extend this program by adding another text box with a label ‘enter your name’ and get the program to respond by ‘hello {name}’ in the original text box. Your form should look like:



Double-click on the button and enter the following instruction:

```
private void button1_Click(object sender, EventArgs e)
{
    textBox1.Text = "Hello " + textBox2.Text;
}
```

Closing the program

So far, we have used the button in the toolbar to close the program, but this would not be appropriate for a Windows application, so we need a formal instruction to close our application. So let’s add another button with the label ‘Exit’ and you can again double-click on it to add the following code:

```
private void button2_Click(object sender, EventArgs e)
{
    Application.Exit();
}
```

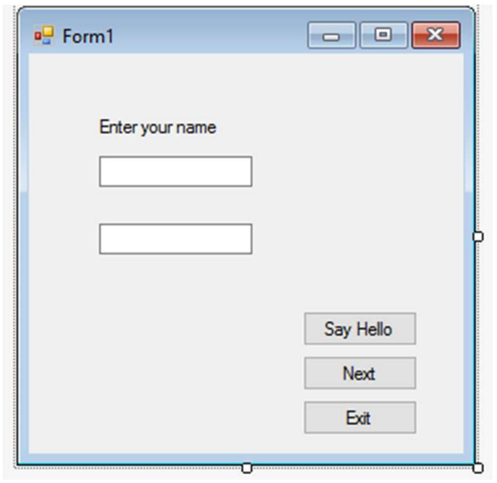
Adding a message box

A typical Windows program will display messages to the user – often to tell the user that they have entered incorrect data, but it might be as simple as a message to confirm an action such as closing the program. It is easily possible to include button options with these messages such as Yes/No/Cancel. We give you a simple example and refer you to a website that gives you further details. For an example, modify your new ‘Exit’ button as follows:

```
private void button2_Click(object sender, EventArgs e)
{
    MessageBox.Show("You are leaving this program!");
    Application.Exit();
}
```

Clearing the text boxes

You might want to enter different names without closing the program and starting again, so it might be useful to add another button e.g. 'Next' which will clear the text boxes. Your new form might look like:



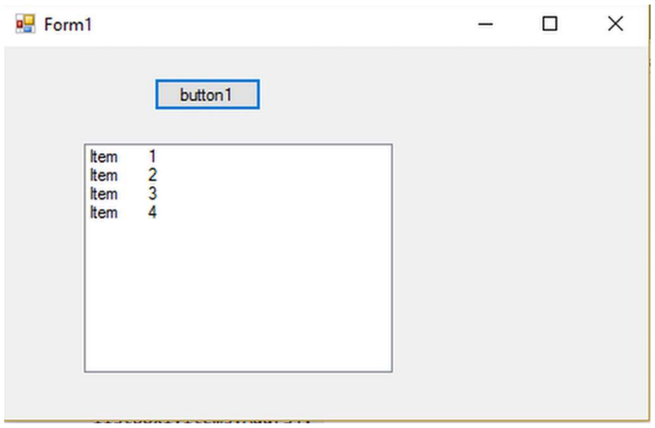
The code for the button would be as follows:

```
private void button3_Click(object sender, EventArgs e)
{
    textBox1.Clear();
    textBox2.Clear();
}
```

We have given you an introduction to Windows Form programming, but there is still very much to learn. You can gain a lot of experience just by experimenting with object properties but we shall help you develop your skills in the following tutorials. You should also look at the on-line resources indicated within this tutorial.

Adding a List Box

A listbox provides a simple and convenient way of displaying a list or table of items. The following example shows a simple form with a listbox and one button:



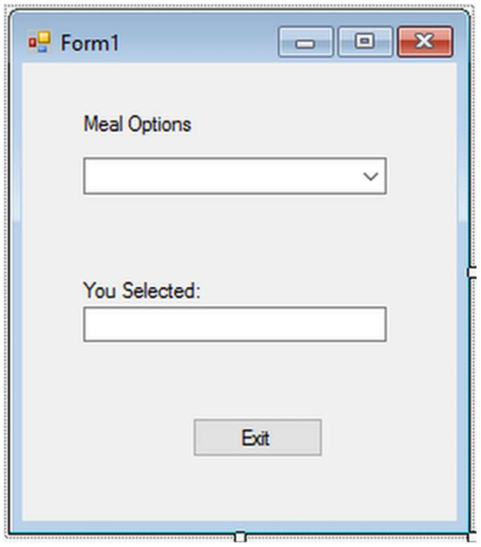
When the user clicks the button, some text and numbers are added to the listbox. If the user clicks the button a second time, it will appear that nothing has happened. However, the listbox has been cleared and repopulated. The code for this is given below:

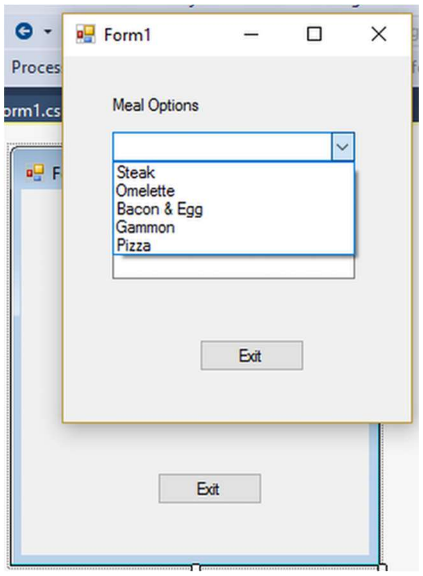
```
        private void button1_Click(object sender, EventArgs e)
        {
            string s = "";
            listBox1.Items.Clear();
            for(int i = 1; i < 5; i++)
            {
                s=string.Format("{0,4} {1,5} {2,1}", "Item", " ",i);
                listBox1.Items.Add(s);
            }

        }
```

Note how we have created a string, s, and then we used the Format method to ensure that the text is neatly presented on each line.

Adding a Combo Box

A Combo Box allows the user to select an option from a ‘pull-down’ list. An example is shown below:





We have to supply the ComboBox with its list of items and, in our example program (see below), we do this by adding the items from an array that has been initialized at the start of the program. By double-clicking on the ComboBox at design time we can create the code for `SelectedIndexChanged` which means that when the user selects an item from the ComboBox, it is automatically displayed in the text box.

```
namespace ComboBoxExample
{
    public partial class Form1 : Form
    {
        String[] meals = { "Steak", "Omelette", "Bacon & Egg", "Gammon", "Pizza" };
        int numMeals=5;
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            for (int i=0; i<numMeals; i++)
            {
                comboBox1.Items.Add(meals[i]);
            }
        }

        private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
        {
            textBox1.Text = comboBox1.Text;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Close();
        }
    }
}
```

Alternatively, we can set the list of items in the Combo Box at design time by selecting the Items (Collections) property of the control. The required list can be entered, line-by-line, and this will appear at run-time.

As with arrays the index starts at 0 and, if we have stored our data in an array, we can relate the selected item directly with the array element by its index e.g. int i = Combo1.SelectedIndex;

Additional Notes

Variable Declaration and Scope

'Scope' refers to the visibility and lifetime of a variable. Consider the following section of code:

```
        private void button1_Click(object sender, EventArgs e)
        {
            int i;
            for( i = 0; i < 6; i++)
            {
                listBox1.Items.Add(Convert.ToString(i));
            }
            listBox1.Items.Add(Convert.ToString(i));
        }
```

This code will compile and run correctly. However, if we declare the variable i within the for statement as shown below, the program will show a build error when we try to compile it.

```
private void button1_Click(object sender, EventArgs e)
 {
     for(int i = 0; i < 6; i++)
     {
         listBox1.Items.Add(Convert.ToString(i)); // for statement block
     }  // End block scope
     listBox1.Items.Add(Convert.ToString(i));
 }
```

This is because the variable i ceases to exist beyond the last curly brace of the program block. This is an example of 'block scope'. The principle applies more generally and the scope of a variable goes from where it is defined to the closing curly brace of the code block/method/class/namespace. A variable should only be 'visible' to the section of code that needs it.

Initialisation can also apply to visual components e.g. check boxes, radio buttons, text boxes etc. While the program will default to the settings established by the user at design time, this may not be the case if a form is revisited during the course of a program's operation. Any form should be initialized properly and we should make no assumptions about its settings upon start-up. There are many principles regarding good form design e.g. giving focus to the first item that requires input. You should look at the following website for more details:

With Windows Form applications, you should give all controls a suitable name e.g. btnExit, textBoxUser. This will help to make your program code more understandable as opposed to the default names e.g. button1. When your application requires input, you should ensure that the cursor defaults to the first input box (or other control). This can be achieved by setting the 'tab index' for that control.

On-Line Resources for Further Study

A simple video introduction to List Boxes can be found at:

https://www.youtube.com/watch?v=ZHWoLvUE4VA

For more details on Message Boxes, see:

https://www.dotnetperls.com/messagebox-show

Notes on Good Practice

As with console applications, the same basic rules apply when choosing variable names so that reflect their purpose and that they begin with a lower-case letter but use an upper-case letter for a second part. When declaring variables, consider their scope and put the declaration in the most suitable place.

Variables should be initialized appropriately and this can also apply to visual components e.g. check boxes, radio buttons, text boxes etc. While the program will default to the settings established by the user at design time, this may not be the case if a form is revisited during the course of a program's operation. Any form should be initialized properly and we should make no assumptions about its settings upon start-up. There are many principles regarding good form design e.g. giving focus to the first item that requires input.

With Windows Form applications, you should give all controls a suitable name e.g. btnExit, textBoxUser. This will help to make your program code more understandable as opposed to the default names e.g. button1. When your application requires input, you should ensure that the cursor defaults to the first input box (or other control). This can be achieved by setting the 'tab index' for that control.

Core Practical Exercises

1.1 Write a program featuring two buttons that increment or decrement a stored variable, displaying the result in a text box. The program should display the original value of the variable in the text box at the outset - this can be done in the public Form1() section, following *InitializeComponent().* Note that the textbox contains only 'text' values, so you will need to use the Convert.ToString() and Convert.ToInt32() facilities.

1.2 Modify the previous program to replace the two buttons by a single feature – the numericUpDown control.

1.3 If the currency conversion rate is 1 British Pound = 1.24 US Dollars, write a program that uses a loop to produce a table in a listbox showing the conversion rates between £1 and £100 in £5 increments.

1.4 Write a program that generates 10 random numbers and stores them in a listbox. Create a button to find the largest of these values and output the result using a simple Message Box with an OK button.

1.5 Design and write a 'Currency Converter' program that offers 'pull down' menu options for the currencies and applies pre-stored values for the conversion.

Your program should check for input errors as well as providing a user-friendly interface.
You should obtain the relevant currency conversion details from on-line resources. You can restrict your program to deal with a maximum of 5 currencies.
The program should feature a basic facility to convert from a selected currency into G.B. pounds, but full marks will be applicable to those who can achieve a conversion between any two selected currencies.

Further Exercises

1 Create a program that displays a box such as a 'panel' and allows the user to change the shape of this box by entering new values for its height and width in text boxes and clicking a button to carry out the action.

2. Modify the previous program to obtain the new height and width values using NumericUpDown controls. The values should be initialized using the height and width properties of the panel. You should change the increment values for the NumericUpDown controls to more suitable values as well as setting the Maximimum and Minimum values.

3. Modify the previous program so that the panel automatically changes its shape when the user changes values in the NumericUpDown controls.

4. Write a program to allow the user to enter a 5-letter word in a text box and, when a button is pressed, count the number of the character ‘a’ in the word. Display the result in another text box. What are the limitations of this program?