

Windows Applications and .Net

C# and .Net provide extensive support for building Windows Applications. The most important point being that they are 'event driven'. All windows applications present a graphical interface to their users and respond to user interaction. This graphical user interface is called a Windows Form ('WinForm' for short). A windows form may contain text labels, push buttons, text boxes, list boxes, images, menus and vast range of other controls.

Windows Form Basics

The .Net provides the WinForm and other controls through base classes in the System.Windows.Forms namespace. The class System.Windows.Forms.Form is the base class of all WinForms in .Net. In order to design a windows application, we need to:

- i). Create a Windows Application project in Visual Studio.Net, or add references to System.Windows.Forms and System.Drawing to your current project.
- ii). Write a new class to represent the WinForm and derive it from the System.Windows.Forms.Form class as shown below:

```
class MyForm : System.Windows.Form
{
    }
}
```

- iii). Instantiate various controls, set their appropriate properties and add these to MyForm's Controls collection.
- iv). Write another class containing the Main() method. In the Main() method, call the System.Application.Run() method, supplying it with an instance of MyForm.

```
class Test
{
    static void Main()
    {
        Application.Run(new MyForm());
    }
}
```

The Application.Run() method registers the form as a windows application in the operating system so that it may receive event messages from the Windows Operating System.

Example

Create an application that presents a simple window with a "Hello Windows Form" greeting at the center.

```
using System;
using System.Windows.Forms;
using System.Drawing;
namespace WindowsApplication
{
    class Test
    {
        static void Main()
        {
            Application.Run(new MyWindow());
        }
    }
}
```

```

    }
    class MyWindow : Form
    {
        public MyWindow() : base()
        {
            this.Text = "Windows Application";
            this.Size = new Size(300, 300);

            Label lblGreeting = new Label();
            lblGreeting.Text = "Hello Windows Form";
            lblGreeting.Location = new Point(100, 100);
            this.Controls.Add(lblGreeting);
        }
    }
}

```

Explanations

The System.Windows.Forms namespaces contains the base classes for windows controls, e.g. Form, Label and Button while the System.Drawing namespace is necessary as it contains the classes related to the drawing of controls. In fact the Size and Point classes are actually defined in the System.Drawing namespace.

In the constructor of MyWindow, the size and title of the form were specified (by setting the size and text properties). And then a text label was created and added to the Controls collection of the Form. The System.Windows.Forms.Label class defines a text label in a Windows application. The text of the Label is set using its Text property, which is of the string type. All the controls contained by a form must be added to its Controls collection.

Adding Event Handling

Add a button labeled 'Exit' to the form. The 'Exit' button will close the application when it is clicked. Push Buttons are instances of the System.Windows.Forms.Button class. To associate some action with the button click, we need to create an event handler and register (or add) it to the Button's Click event. The following is the code for this application.

```

using System;
using System.Windows.Forms;
using System.Drawing;
namespace WindowsApplication
{
    class Test
    {
        static void Main()
        {
            Application.Run(new MyWindow());
        }
    }
    class MyWindow : Form
    {
        public MyWindow() : base()
        {
            // Form
            this.Text = "Windows Application";

```

```

        this.Size = new Size(300, 300);
        this.StartPosition = FormStartPosition.CenterScreen;

        // Label
        Label lblGreeting = new Label();
        lblGreeting.Text = "Hello WinForm";
        lblGreeting.Location = new Point(100, 100);

        // Button
        Button btnExit = new Button();
        btnExit.Text = "Exit";
        btnExit.Location = new Point(180, 180);
        btnExit.Size = new Size(80, 30);
        btnExit.Click += new EventHandler(BtnExitOnClick);

        // Adding controls to Form
        this.Controls.AddRange(new Control[] {lblGreeting,
        btnExit});
    }

    public void BtnExitOnClick(object sender, EventArgs e)
    {
        Application.Exit();
    }
}

```

Explanations

The constructor of MyWindow is used set certain properties of the Form. In addition StartPosition property has been set to specify the position of the form on the screen when the application starts. The type of this property is an enumeration called 'FormStartPosition'.

An an event handler method for this button called BtnExitOnClick() has been created that contains code to exit the application. The event handler has also been subscribed to the btnExit's Click event. Notice that both the label and the button have been added to the form's Controls collection using the AddRange() method of form class that adds an array of controls to the Controls collection of form. This method takes an array of type Control as its parameter.

Example

A windows application that display a message using a message dialog

```

using System;
using System.Windows.Forms;
using System.Drawing;
namespace CSharpSchool
{

```

```

class Test
{
    static void Main()
    {
        Application.Run(new MyWindow());
    }
}
class MyWindow : Form
{
    public MyWindow() : base()
    {
        // Form
        this.Text = "Windows Application";
        this.Size = new Size(300, 300);
        this.StartPosition = FormStartPosition.CenterScreen;

        // Button1
        Button btnExit = new Button();
        btnExit.Text = "Exit";
        btnExit.Location = new Point(180, 180);
        btnExit.Size = new Size(80, 30);
        btnExit.Click += new EventHandler(BtnExitOnClick);

        // Button2
        Button btnMessage = new Button();
        btnMessage.Text = "Message";
        btnMessage.Location = new Point(50, 180);
        btnMessage.Size = new Size(80, 30);
        btnMessage.Click += new
            EventHandler(btnMessageOnClick); // Adding controls to
            Form
        this.Controls.AddRange(new Control[] {btnMessage,
            btnExit});
    }
    public void BtnExitOnClick(object sender, EventArgs e)
    {
        Application.Exit();
    }
    public void btnMessageOnClick(object sender, EventArgs e)
    {
        MessageBox.Show("Hello");
    }
}
}

```

Visual Studio.Net & its IDE (Integrated Development Environment)

Most of the time, you use Visual Studio.Net to develop Windows applications in C#. Visual Studio.Net provides a lot of tools to help develop applications. Visual Studio.Net provides a standard code editor and IDE for all .Net applications, along with a standard debugger, project and solution settings, form designer, integrated compiler and lot of other useful tools.

Components of Visual Studio IDE

1). Form Designer

This is one of the most useful feature of the Visual Studio.Net IDE. The form designer allows you to design the graphical user interface just by placing the controls on the form from the Toolbox. You can set a lot of properties of the form and its controls using the Properties window.

The Visual Studio.Net IDE automatically writes the code in the source file as you place the controls on the form and change their properties.

2). Solution Explorer

The Solution Explorer presents the files that make up the solution in a tree structure. A solution is a collection of all the projects and other resources that make up a .Net application and may contain projects created in different .Net based languages like VB.Net, VC#.Net and VC++.Net.

A .Net solution is saved in a .sln file, a C# project is saved in a .csproj file and C# source code is saved in a .cs file.

3). Properties Window

This window displays properties of the currently selected object. It generally used to you can change the various properties of the form and its controls. Event properties can be changed by switching to the Event Properties pane in the Properties Window.

4). ToolBox Window

This window is used to add controls (tools) to a form by either double-clicking or dragging/dropping (your option on which to use).

5). Integrated Compiler, Solution builder and Debugger

Visual Studio.Net provides an integrated compiler to compile and execute your application during development. You can either compile a single source file or the complete project and solution (a group of files that make up an application). Once you have compiled your application, you can debug it using the Visual Studio.Net debugger. You can even create an installer for your application using Visual Studio.Net

The Toolbox, Properties Window, Help Window, Solution Explorer Window, Output Window and other helping windows in Visual Studio IDE can be set for Docking and Auto hiding. Windows that are set for auto hide appears only when they get focus (e.g. they have mouse pointer over them or receive a mouse click), and hide when they lose focus. The hidden windows are always accessible through the left and right hand panes of the form designer window. If some of these windows are not visible in your visual studio IDE, you can make them visible from the View menu on the standard menu bar.

Adding Event Handling

To add an event handling code for a control, you simply need to double click the control in the designer. This creates a new event handler for the control's particular event.

After the event handler is created you need to write the code to perform a particular application. The IDE does not only create the event handler but also registers it with the control's event e.g. button's Click event

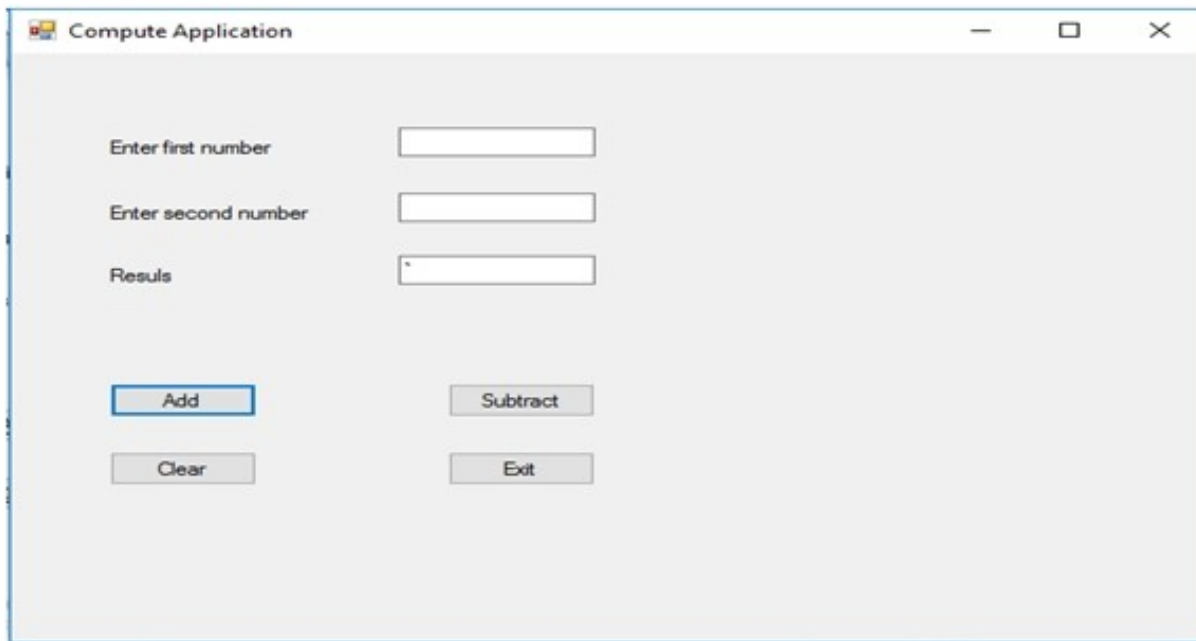
Example

You are required to write a program that will enable user to calculate the sum, product, division and difference of two numbers entered through textboxes after a click of the respective buttons. The results should be displayed by use text box control. The program should enable the user to clear the textboxes ready for the next input.

- i) Sketch your graphical user interface assigning appropriate properties to your form and other controls ii) Write the code for your program and declare you variables appropriately

Solution

i)



Properties

- The first two TextBox controls are used for data entry - use these names: txtNumber1 and txtNumber2 ■
- The third TextBox control is used to display the output. Set the Name property as txtResults
- Name the buttons btnAdd, btnSubtract, btnClear, and btnExit.

ii) //Program code

```
using System;
using System.Windows.Forms;

namespace ComputeFormsApplication
{
    public partial class ComputeForm : Form
    {
        int number1, number2;
        public ComputeForm()
        {
            InitializeComponent();
        }

        private void btnAdd_Click(object sender, EventArgs e)
        {
            int sum;
            number1 = int.Parse(txtNumber1.Text);
```

```

        number2 = int.Parse(txtNumber2.Text);
        sum = number1 + number2;
        txtResults.Text = sum.ToString();
        MessageBox.Show("Sum is : " +
sum);    }

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

private void btnClear_Click(object sender, EventArgs e)
{
    txtNumber1.Clear();
    txtNumber2.Text = "";
    txtResults.Text = String.Empty;
}

private void btnSubtract_Click(object sender, EventArgs e)
{
    int difference;
    number1 = int.Parse(txtNumber1.Text);
    number2 = int.Parse(txtNumber2.Text);
    difference = number1 - number2;
    txtResults.Text = difference.ToString();
}
}
}

```

Controls

Controls are all instances of control classes and the collection of control classes are arranged in a class hierarchy i.e. **System.Windows.Forms.Control** Class.

A summary of some of these controls is presented below.

Control	Description
Label	Used to display some text on the form. Instance of System.Windows.Forms.Label. Important
Button	Used to display a Push Button on the form. Instance of System.Windows.Forms.Button. Important properties are Text, Name and Font. Usually the Click event is handled for the Button.
TextBox	Provides the user with an area to write/edit text. Instance of System.Windows.Forms.TextBox. Important properties are Text (to set startup text and get what the user has entered), Name, Alignment, Multiline (boolean), ScrollBars (a set of scroll bars attached with the text box), ReadOnly (boolean), WordWrap (boolean) and PasswordChar (character used for password masking). Important events are TextChanged (default) and KeyPress.
GroupBox	Used to group other controls. Instance of System.Windows.Forms.GroupBox. Important properties are Text, Visible (boolean) and Enabled (boolean). Usually events are not handled for the GroupBox.

RadioButton	Allows a user to select one out of many available options. RadioButtons are usually used in a group contained in a GroupBox. Only one of the RadioButton can be selected in a group at a time. Instance of System.Windows.Forms.RadioButton. Important properties are Text, Name and Checked (boolean). Important event is CheckStateChanged. Usually the events of a RadioButton are not handled; rather the selected choice is identified on the click of some push button or actions on other controls.
CheckBox	Allows a user to tick or untick a box to state their preference for something. CheckBoxes are usually used in a group contained in a GroupBox. Any, all or none of the checkboxes in a group can be selected. Instance of System.Windows.Forms.CheckBox. Important properties are Text, Name, Checked (boolean) and CheckState. Important events are CheckStateChanged and CheckStateChanging.

