Visual C# Programming Basics

In this tutorial you will learn how to make applications for Windows in C#. You will learn how to use Visual Studio to build simple applications, how to use most of the Windows Forms controls, and several tips on how to publish your application.

Made by Davide Vitelaru

http://davidevitelaru.com/

General Requirements

To follow this tutorial you will need the following items:

-Knowing the basics of at least one programming language (To know what variables, arrays, functions, etc… are)
-A computer running Windows XP/Vista/7
-Microsoft Visual C# Express (Click for download)

You can also use Microsoft Visual Studio Professional, but this is a commercial version of the Visual C# Express, and it has more features. We will not use most of them in this tutorial though.

If you are interested in some specific part of this tutorial, check out the table of contents on the last page because you might find what you are looking for.

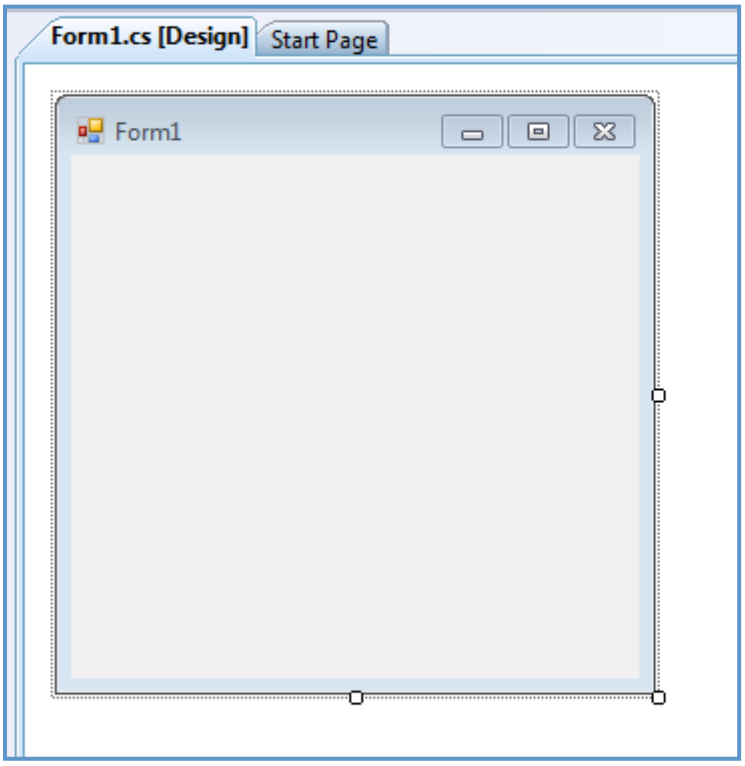
Quick Start – Your first application

In this chapter, you will learn how to make an application in Visual Studio from start to finish. How to code it, design it and publish it.

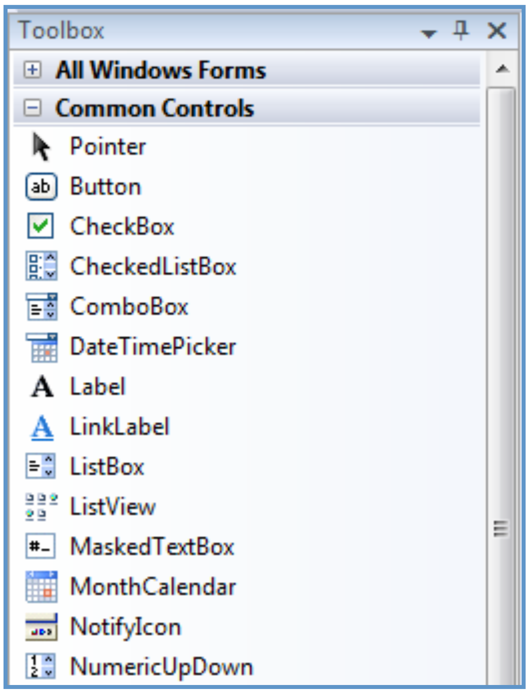
Step 1 – Creating the project

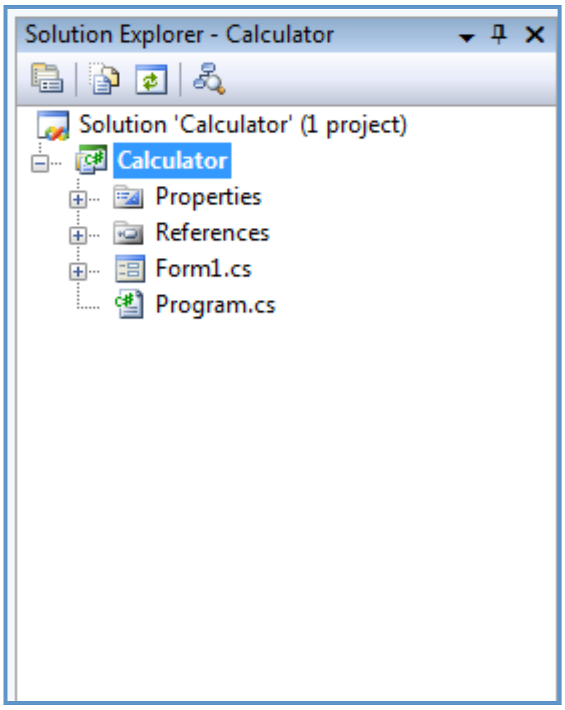
To start, open Visual C# Express or Visual Studio and create a new project by pressing on the "New Project" icon in the upper left corner.

In the window that opens you can select your project type, depending on what you want to program in C#. To make a simple Windows application, select "Windows Forms Application", name your project "Calculator" (because this is what we are going to do) and press "OK".



You now created a new project. You might get all scared by Visual C#'s interface because it is very crowded and you don't know what most of the controls do. Let's take a look at the interface for a bit: the first thing that pop's into your eyes is the form right in the middle. It is an empty form and what you have to do is to take controls from the "Toolbox", the panel from the left, and put them on it.




Solution Explorer - Calculator
Solution 'Calculator' (1 project)
Calculator
Properties
References
Form1.cs
Program.cs

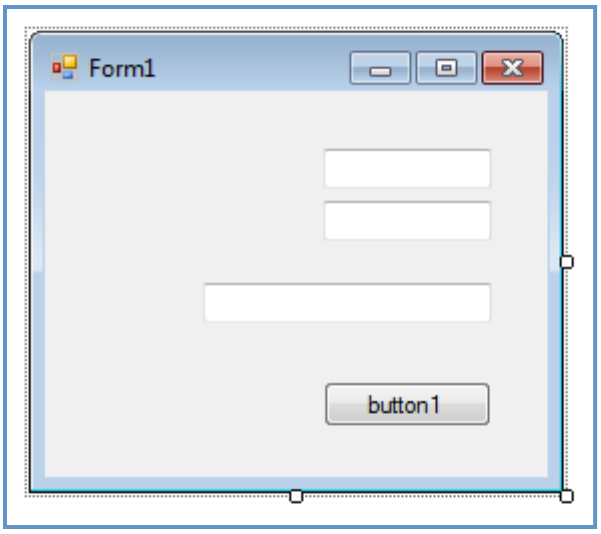
You can see different type of controls in the "Toolbox": buttons, textboxes, progress bars, and you can take all of them and place them on your form. Don't place anything on your form now, if you did, select them and delete them.

On the right you have your "Solution Explorer". When you create a new project, you automatically create a new solution. A solution is a collection of multiple projects, let's say we make an application called "Calculator" (cause this is what we actually do), and "Calculator" is an application project inside the "Calculator" solution. If we want to create a setup for "Calculator", we create the setup project inside the same solution. You will learn what everything in the solution explorer means later.

Step 2 – Designing the form

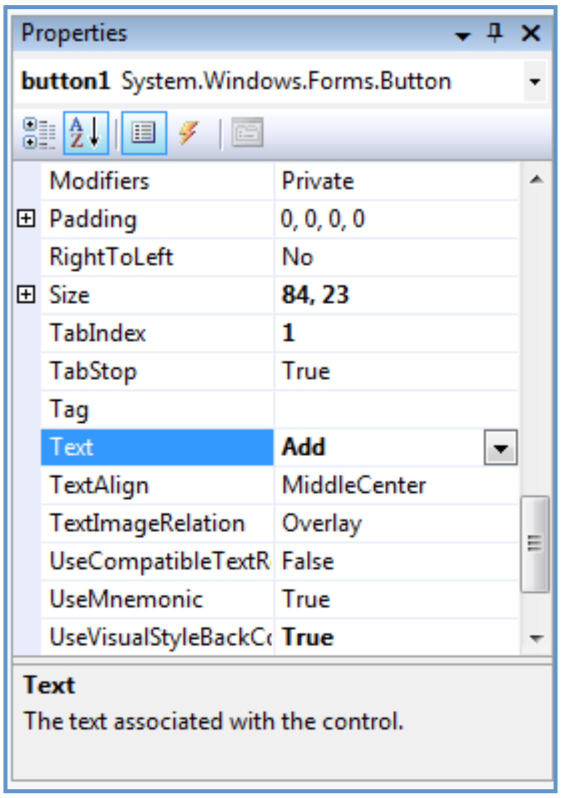
What we want to create is a simple calculator application. What it will do is to add two numbers inserted by the user. To start, we will need three text-boxes: Two for the two numbers that the user wants to add and the third for the result. We will also need a button so that the user can press it and receive he's result.

To do all this, click on the "Text Box" control in the toolbox, and then click on your form. As you can see, a text box appeared on your form. Repeat this step again and create two more text boxes. Align the text boxes the same way I did:



Now, select the button control from the toolbox and create a button on the form.

Good, we now created all the controls we need for our application. But, there is a problem, why is the button named “Button1”? Because this is how it is by default, to change that, we need to change its properties. By default, the properties window is not opened in Visual C#. To open it, go to “View” and click on “Properties”.



button1 System.Windows.Forms.Button

Modifiers	Private
Padding	0, 0, 0, 0
RightToLeft	No
Size	**84, 23**
TabIndex	**1**
TabStop	True
Tag	
Text	**Add**
TextAlign	MiddleCenter
TextImageRelation	Overlay
UseCompatibleTextR	False
UseMnemonic	True
UseVisualStyleBackC	**True**

Text
The text associated with the control.

The properties panel (obviously) shows the select controls properties, such as height, width, color, text, etc… In this case, we only need to change the text since the button can be resized with using the mouse. Click on the button (Make sure you don’t double click it, or its code will open. If that happens, close the tab with the code from the top of the middle-panel). Once clicked, the button’s properties will appear in the “Properties” window. Scroll down and go to “Text”. To its right you will see “Button1”. Change that to “Add”, and press enter.

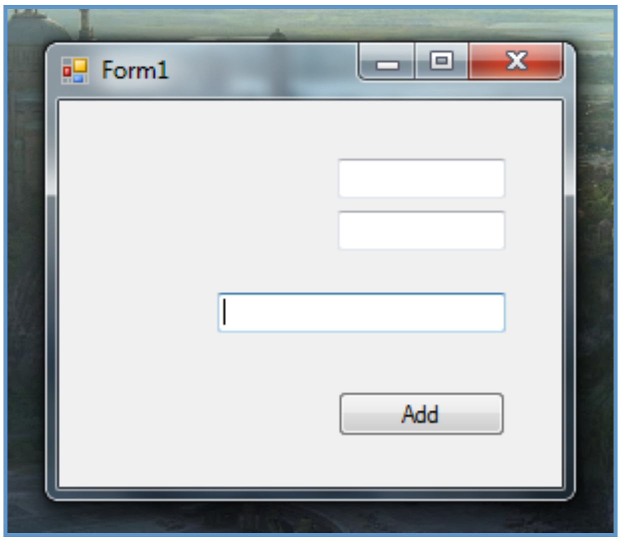
Your button now has “Add” written on it. Very good, this way you can edit every item’s properties, even the text boxes’.

Also, you might notice that the form’s name is “Form1”. Try to do something about it.

How To: Click on an empty space on the form, change the form’s text property to “Calculator”.

Step 3 – Debugging the application

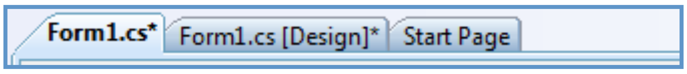
Of course, we added the controls on our form, but the button doesn’t do anything since we didn’t “tell” it do to anything. You can see your program running by pressing the “Debug” button, the green arrow in the toolbar (▶). When you click the debug button, you application will start. It should look something like this:



You probably tried to click the button already, and noticed how well he is doing his job. Debugging is the best method to test your application before publishing it. Every time you make a change to it, you debug it to see if it worked. Once you finish the application, you build the entire project turning everything into one executable, and probably make a setup for your application.

Step 4 – Coding the application

To make the application work, you obviously have to write some code. If you are still debugging your application, close it and go back to your project. Now, double-click on your button to open the code window. As you can see, everything is tabbed in the project. You can always go back to the form designer by clicking its tab on the top.



With all the amount of code that you have in front of you, you might get scared (again!). Don’t worry you will get used to it, and to even more than that. If you are reading this tutorial, let’s hope you already know the basics of another programming language, if not it will be hard for you to search Wikipedia for every word you don’t understand, but it would be useful.

The first statements in the code import some items from the .NET Framework. What is that you might ask, the .NET Framework is a large library of coded solutions to common programming problems that manages the execution of programs written specifically for the framework. To be clearer, it is a large amount of code already written for you, so you can build programs with a pretty user interface and other things. It helps you very much because it reduces the amount of things that you need to learn to create the entire user interface. We should all thank Microsoft for it:

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

By “import some things” from the .NET Framework, I meant importing some classes. Those classes can be the forms, the user controls, and all the other things that helped us by now creating the program. You will learn the meaning of them later.

For now, let’s see the rest of the code:

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

        private void button1_Click(object sender, EventArgs e)
        {

        }
    }
}
```

The “`public Form1()`” statement is a class that is executed when we start the program; actually, when we open the form named “Form1.cs” (“.cs” is from C Sharp). In case you did not know, in C# the code is usually put between curly braces just like in Java and C++.

The “`private void button1_Click(object sender, EventArgs e)`” is that class that is executed when we click the button. Inside it, we will write the code that will add the two values from the text boxes.

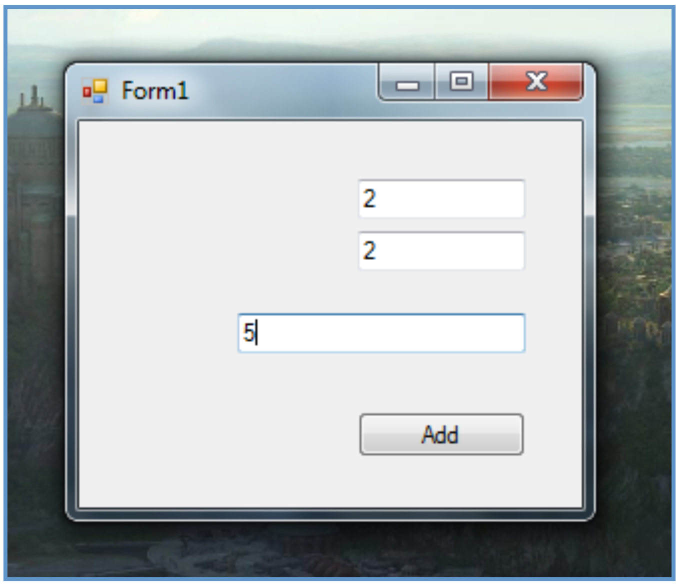
Note: In C#, two slashes (//) represents the beginning of a comment. A comment is a piece of code that is not executed by the compiler, it is used to help you organize you code, and so that other programmers will understand what every piece of code means. We will use comments inside our codes for better explanation.

To make that button add the two values and return the sum we need, we have to grab the text content from the two text boxes, turn it to integers, add them, and change the text of the third text box to the sum. It is very simple:

```
            double val1, val2; //We declare two double type variables

            //We assign to the first variable the value of the text box
            //Since the text box cand hold a string, it must be converted
            //to a double to assign it to "val1".
            //Note that we assign using "=" as an operator

            val1 = Double.Parse(textBox1.Text);
            //Double.Parse("string") converts the string put into the brackets
            //and assigns it to a double

            //Same thing for the second variable
            val2 = Double.Parse(textBox2.Text);

            //Now we are doing the exact oposite, we take the two
            //double values and we convert their sum to a string
            //using the .ToString() command
            textBox3.Text = (val1 + val2).ToString();
```

Now that we finished you might want to debug your project and see if it works.

What we did is easy to explain; we declared two variables and assigned the values of the two text boxes after we converted them from strings to integers. Then, we changed the text of the third text box into the sum of the two variables, and we converted it to a string in the same time. And we did all of this at the click of a button.



Step 5 – Publishing you application

What you have to do is to create an icon for your application, change its publishing settings and make a setup for it, but we will skip these steps as they are not related to basic C# programming.

Note: Check out the "Advanced Visual C# Programming" tutorial once you finish this one.

Understanding Visual C#

Great, now that you made your first application you can go even deeper into C# and understand how most things work, and some extra things that will make your application work better.

Control names

First of all, one important thing that you have to know is that every item on your form has a name. And I am not talking about "Text Box" or "Button", but about their "Name" property.

Go back to your forms designer and click on the first text box. In the property window, you will see that its name property is “textBox1”. In our previous code, we took the value from this text box by the following method:

```
            val1 = Double.Parse(textBox1.Text);
```

How does this work? Let’s forget about the conversion for a few seconds, and see what we actually assigned to the variable (considering that the text box already holds a double value).

```
            val1 = textBox1.Text;
```

We assigned the specific property of a control to a variable. The correct syntax is:

```
            variable = control.property;
```

This is how it always works. Of course, we can do the exact opposite and assign to the control’s property a certain variable just like we did earlier:

```
            textBox3.Text = (val1 + val2).ToString();
```

The names are used to distinguish between the controls on the form.

You can do the same thing with buttons, just click on a button, and change its name property. It is recommended that you rename your buttons so that you know what each of them does.

Events and classes

When you double click a button, you are automatically redirected to your code where a new function appears. That function will be executed when you click the button. You can run any other function when you click the button, all you have to do change the event that occurs when you click it. To do that, go to its property window, and click on the “Events” button ( ). On the left side of the table that just appeared bellow you will see the events, and on the right side you will see what happens (What function is executed) when that event occurs.

As you can see, when we **Click** the button, the **button1_Click** function is executed. That function has been automatically written for you when you double-click the button. Of course, the same function can be executed no matter what event occurs, for example, go to the **MouseHover** event, and type **button1_Click** to its left. From now, you don’t have to click the button, all you have to do is place the cursor above it and the **button1_Click** function will run, adding the two numbers.

You can also declare a new function to do the job, and call it when you click the button. Type somewhere below the **button1_Click** function:

```
        double AddNumbers(double val1, double val2)
        {

        }
```

This is how we declare a new function in C#. The correct syntax is:

```
        value_returened function_name(parameter_type parameter_name)
        {
            code
        }
```

The **AddNumbers** function will return the sum of the two numbers. Type inside the two curly braces:

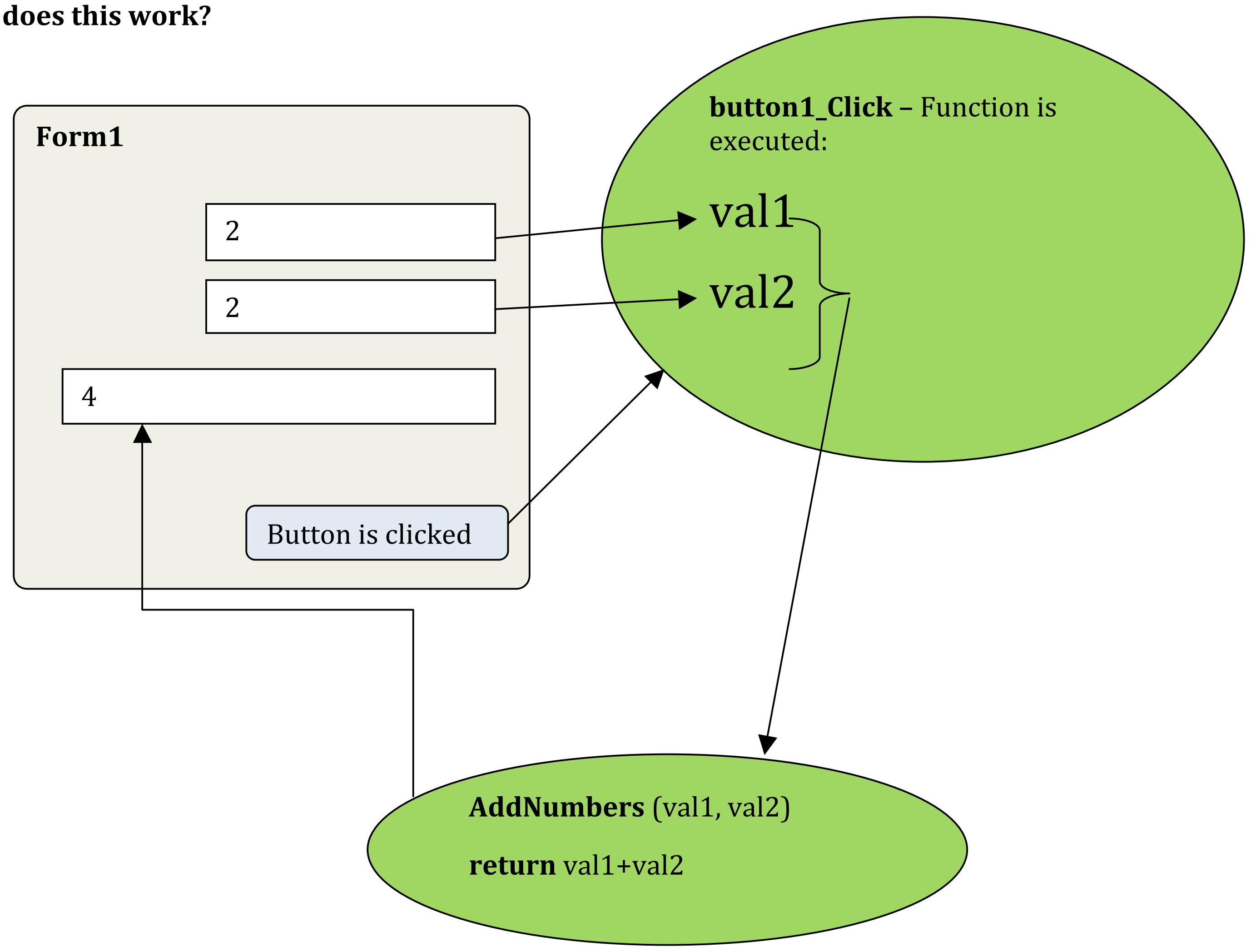
```
            return val1 + val2;
```

"`return`" is used to specify what value the function will return when it is called. To call this function, delete the code in the **button1_Click** function and type:

```
        private void button1_Click(object sender, EventArgs e)
        {
            double val1 = double.Parse(textBox1.Text);
            double val2 = double.Parse(textBox2.Text);
            textBox3.Text = AddNumbers(val1, val2).ToString();
        }
```

As you can see, we can assign a value to a variable when we declare it. We used the **AddNumbers** function to add the two numbers.

How does this work?

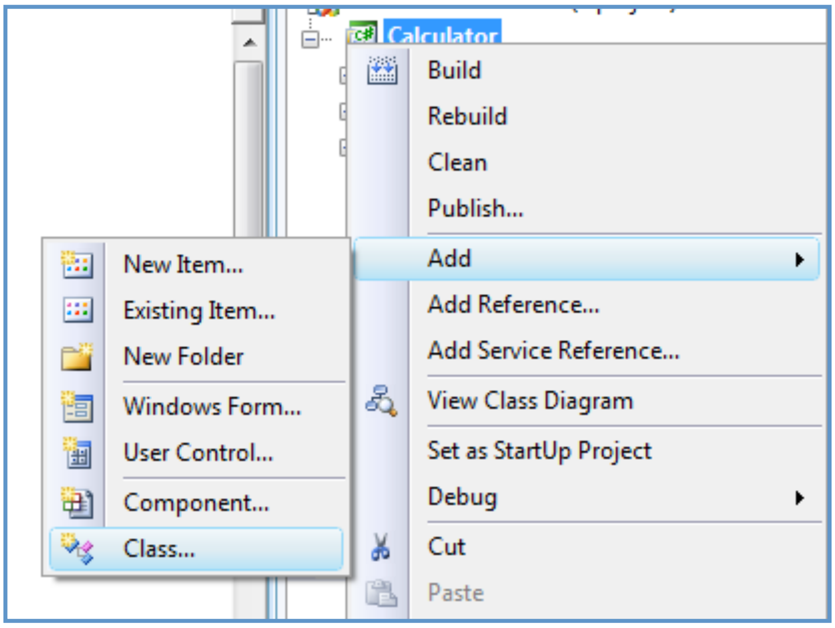


Note how the **button1_Click** function passes the values to the **AddNumbers** and then assigns them to the third text box.

Debug your application and you will notice that it will work. The best part in using function to do your job is that you can use them multiple times without you having to write the same code all over again.

If you have too many functions, you source code might get really crowded. In this case you might want to create a class to hold all of them in one place.

To create a class in your project, right click on your project icon in the “Solution Explorer” and add a new class:



Name it “Calculator.cs”. Once you create it, it will appear in the solution explorer, it will automatically open, and you will have this:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Calculator
{
    class Calculator
    {

    }
}
```

Inside the class, cut/paste the **AddNumbers** function, and change the following line:

```
        public double AddNumbers(double val1, double val2)
```

We typed “`public`” before declaring the function so we can use it outside this class.

Now, go back to “Form1.cs” and declare the class right on top of the **Main** function:

```
        Calculator Calc = new Calculator();
```

The correct syntax is:

```
        ClassName WhatNameYouWantToUse = new ClassName();
```

Now that we declared the class, we need to use the function from it named "**AddNumbers**". To do that, change the following line of the **button1_Click** function:

```
            textBox3.Text = Calc.AddNumbers(val1, val2).ToString();
```

This way, you can use the class on multiple forms.

Homework: Make a new button and a new function named "MultiplyNumbers", and make it multiply the two numbers when the button is pressed.

Note: to multiply two numbers in C#, you can use the "" operator. ("- to subtract, and \ to divide").*

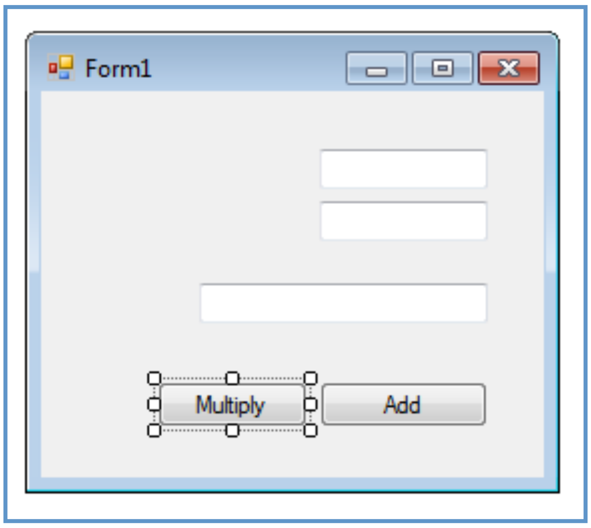
Try doing it by yourself before reading the code.

Code:

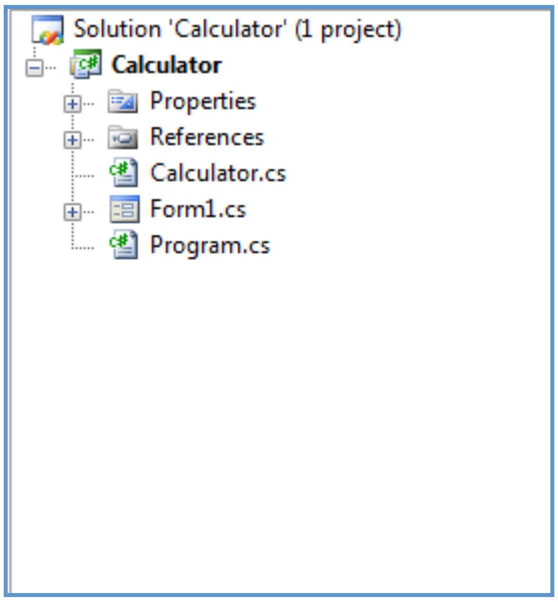
Calculator.cs

```
        public double MultiplyNumbers(double val1, double val2)
        {
            return val1 * val2;
        }
```

Form1.cs



```
        private void button2_Click(object sender, EventArgs e)
        {
            double val1 = double.Parse(textBox1.Text);
            double val2 = double.Parse(textBox2.Text);
            textBox3.Text = Calc.MultiplyNumbers(val1, val2).ToString();
        }
```



Solutions and Projects

It is time to learn how the solution explorer works.

File types

All you have to care about in the solution explorer are the forms, classes and references.

The forms have the following icon: and they are actually a collection of two classes, one handles the design but you don't write in that file, instead you arrange the control using the designer that writes the code by itself in the designer class, and the other one is the code class. Right-clicking on the form icon will give you the option to open either the code or the designer.

The classes have the following icon: and they are just independent files with code. "Program's" is the main class of your project, and you must not modify it.

The references are all inside this folder: , and they usually represent the pieces of the .NET Framework that you are using. Also, if you are using any other class library it will appear there.

Resources usually represent the images that you imported into your project. They can be found all in this folder: .

File system

You might ask yourself after you work at an application "Where exactly is my application?". To find the executable created as a result of you building your project, go to your documents folders, in "Visual Studio 2008" (Depending on the version you have) and go to:

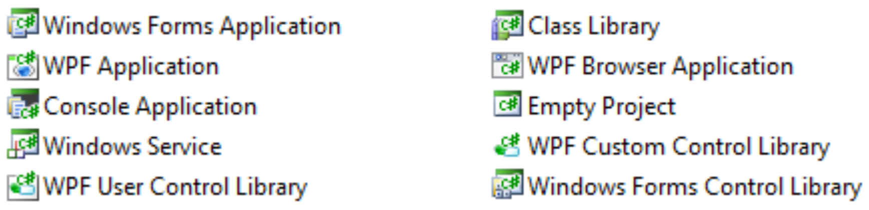
Projects/<Project Name>/<Project Name>/bin/Debug/<Project Name>.exe

Other project types

This chapter is not that important and you can skip it if you are not interested in other types of projects.

Windows

As you have noticed when you created a new project, there are many other types of projects. This is what we have in the Windows category:



The "Windows Forms Application" project is what we previously made. It is just a simple application with a user interface. On the other hand, the "Console Application" is a simple C# application project without the user interface, it works inside the console.

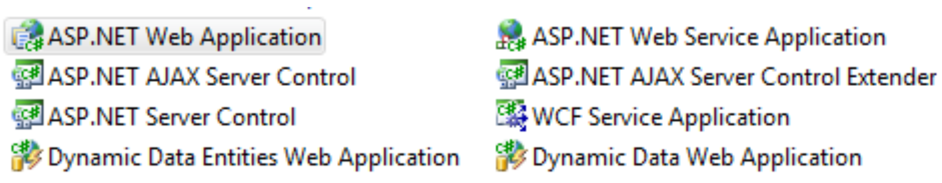
The “Class Library” is actually a project full of classes. Once you compile the project, you will end up with a “.DLL” file (Dynamic Link Library). You can add that file as a reference to your project so you can use the classes inside the library.

The other project types are advanced and you will learn what they are good for at the right time.

Web

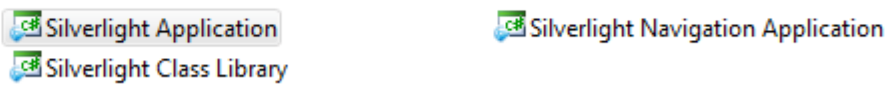
There are also other categories of Visual C# project types:

Web projects – These are for ASP.NET Web Developers:


ASP.NET Web Application
ASP.NET Web Service Application
ASP.NET AJAX Server Control
ASP.NET AJAX Server Control Extender
ASP.NET Server Control
WCF Service Application
Dynamic Data Entities Web Application
Dynamic Data Web Application

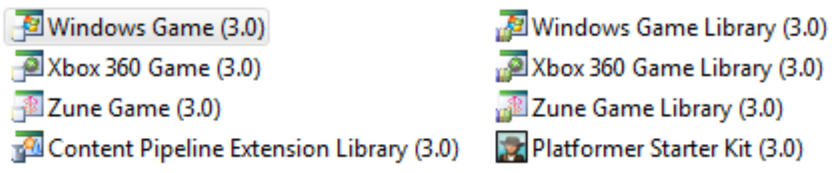
Silverlight

Silverlight development – In case you want to program Silverlight applications:


Silverlight Application
Silverlight Navigation Application
Silverlight Class Library

XNA Games

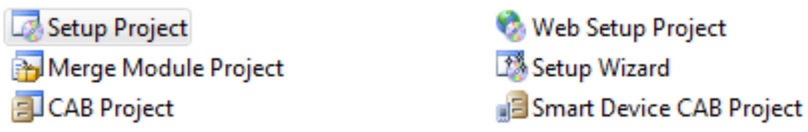
Also, last but not least, XNA projects for game-making using C#:


Windows Game (3.0)
Windows Game Library (3.0)
Xbox 360 Game (3.0)
Xbox 360 Game Library (3.0)
Zune Game (3.0)
Zune Game Library (3.0)
Content Pipeline Extension Library (3.0)
Platformer Starter Kit (3.0)

For the last two project categories you must install plug-ins. Don’t worry, they are free to download from the internet, how you will learn to use them is the biggest problem.

Setup Projects

In case you have Visual Studio Professional installed, under “Other Project Types” in “Setup and Deployment” you have this:


Setup Project
Web Setup Project
Merge Module Project
Setup Wizard
CAB Project
Smart Device CAB Project

The setup project is the easiest way to create a decent and customizable setup for your project.

Visual C# Syntax

This chapter will show you some basic C# code for doing different operations.

Variables & Operations

```
string MyString = "123456";
int MyInt = 24;
double MyDouble = 15.6;

MyString = "dav"; //Simple assigning
MyString = MyDouble.ToString(); //Double to string conversion;

//Int to string to double conversion
MyDouble = double.Parse(MyInt.ToString());
//This is because you need a sting between thos brackets.

MyInt = Int32.Parse(MyDouble.ToString());
//Same here;

MyInt += 1234;
//This is the += operation, that means you
//assign to "MyInt" MyInt + 1234;
//Equivalent: MyInt = MyInt + 1234;

MyDouble = double.Parse(MyInt.ToString()) + 15;
```

Loops

```
bool ok=true;
// boolean can hold true or false values
int i = 0;
//while... do loop
while (ok==true)
{
    i++; //This adds one unit to i
    if (i==1000) ok=false;
}

//for loop
for (i=0; i<=1000; i++)
{
    if (i == 5000) break; //break is used to end the loop
}
```

Decisions

```
//if ... then condition
int i = 4;
bool ok=false;
if (i / 2 == 2)
{
    ok = true;
}
else
{
    i++;
}

string String = "1234"; //Notice that C# is case-sensitive
```

```
            //This is the switch-case command
            //it works like multiple if's
            switch (String)
            {
                //This happens when no other value works
                default: break;

                //In CASE String is "12345", this is what happens
                case "12345":
                {
                    i=1;
                    break; //Always remember to break
                }

                case "412":
                {
                    i = 4;
                    break;
                }
            }
```

File Operations

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO; //Notice how we are using the system IO for file operations

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

        private void button1_Click(object sender, EventArgs e)
        {
            string FilePath="C:\\Users\\Bubby\\Documents\\1234.txt";
            //Remember to use \\ instead of \ because it will throw you an error

            //Writing to files
            StreamWriter Name = new StreamWriter(FilePath);
            Name.WriteLine("First line");
            Name.Write("Second line, but not ended");
            Name.Close(); //NEVER forget to close

            //Reading from files
            StreamReader OtherName = File.OpenText(FilePath);
            string Something = OtherName.ReadLine(); //Reading a line
            string AllContent = OtherName.ReadToEnd(); //Reading everything to end
            OtherName.Close();
        }
```

Windows Forms Controls

In this chapter you will learn how to use most of the controls that you can find in your toolbox. We are still going to work on the calculator project until we reach some control that we can't use in such a project. Also, keep in mind that you will learn to use some easy control in this chapter, for more difficult controls, see the "Advanced Visual C# Programming" tutorial.

Forms

This is a must-read item because you will need to know how to make your form look better. For start, our main form in calculator named "Form1" is resizable, even if it has only 4 controls in it. Once you resize it, it looks ugly so go to its properties in **FormBorderStyle** and change it to "Fixed Single". This way you can't resize it anymore.

Even though you can't resize it, you can still maximize it (annoying, isn't it?) so go to its **MaximizeBox** property and change that too false.

In case you want to make and application with multiple forms, go to your project, right click and add a new "Windows Form".

Still, the first form that will open is Form1 so, if you want to open the other form, make a new button, and on its **Click** event write:

```
        private void button2_Click(object sender, EventArgs e)
        {
            Form2 NewForm = new Form2();
            NewForm.Show();
        }
```

Of course, this works in case your form is named "`Form2`" and you can name the variable after it the way you want. This is just like declaring a class.

To close the form you're in, type "`Close();`", but this will also close your application if you are in your main form. If you just want to hide it, use "`Hide();`".

Labels

Labels are just pieces of text that you put on a form to guide the user. In your "Calculator" project, drag 3 labels on your form. From the property windows you can change their color (**ForeColor**) their text (**Text**), their alignment (**TextAlign**), font (**Font**), and many more…

Check Box

A check box is usually used to check if an option is enabled or not. In this case we can use it to enable/disable the "Add" button.

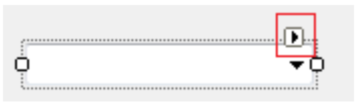
Drag a check box on your form. Then double-click on it to open the function that occurs when it's check is changed and type:

```
        private void checkBox1_CheckedChanged(object sender, EventArgs e)
        {
            if (checkBox1.Checked == true)
            {
                button1.Enabled = true;
            }
            else
            {
                button1.Enabled = false;
            }
        }
```

This will disable the button if it is unchecked and enable it if it is checked. As you can see, the ".checked" property is a Boolean type variable.

Combo Boxes

These are usually used to select one item or a certain option. In this case we will use it to choose whether we add or multiply the numbers. Drag and drop a new text box on your form, and click on the arrow in the upper right corner (that is inside a red square in the picture) and click "Edit Items":



Then add the items you would like in the windows that opens.

We can use it by checking if its value (**Text** property) is "Add" or "Multiply" when clicking the button:

```
        private void button1_Click(object sender, EventArgs e)
        {
            double val1 = double.Parse(textBox1.Text);
            double val2 = double.Parse(textBox2.Text);
            if (comboBox1.Text == "Add")
            {

                textBox3.Text = Calc.AddNumbers(val1, val2).ToString();
            }
            else
            {
                textBox3.Text = Calc.MultiplyNumbers(val1, val2).ToString();
            }
        }
```

Link Label

The link label is basically a simple label that can be clicked just like a button. The main difference is that the cursor changes when you hover on it so the user knows that it can be clicked.

Picture Box

To change the picture of a picture box, go to its properties in "Image". I recommend changing it's background image instead of the real image that it's holding since it can be stretched and tiled, but in this case you could use a panel for this job.

Radio Buttons

Radio buttons work just like check boxes, but it doesn't allow you to have more than one option selected. Fortunately, if you add more to the form, only one of them can be checked at once.

If you want more radio buttons but for something different, just make a new panel and put them in there so that they won't un-check when you click the other buttons.

Contents