2010

VB.NET Programming

By mkaatr

This ebook is a quick introduction to VB.NET programming language. It explains the very basics of the language with screenshots showing what is expected to see during development process.

mkaatr
http://www.mkasoft.com
10/13/2010

VB.NET Programming

By mkaatr

Table of Contents

Introduction

This ebook is the result of combining a number of tutorials available on the site. It gives basic information about using the language, however it does not cover every aspect of the language. There are lots to learn. I tried to put screen shots in most of the tutorials along with the source files embedded in the PDF file itself. If you still need to see how to do some stuff, check the website, there you can watch video tutorials which cover the same exact topics in this book.

Feedback is highly appreciated. If you have any comments, notes, or recommendations, send them to **notes@mkasoft.com**. Finally if you find this book useful, you might want to support this work and further tutorials by sending a small donation to **donation@mkasoft.com**.

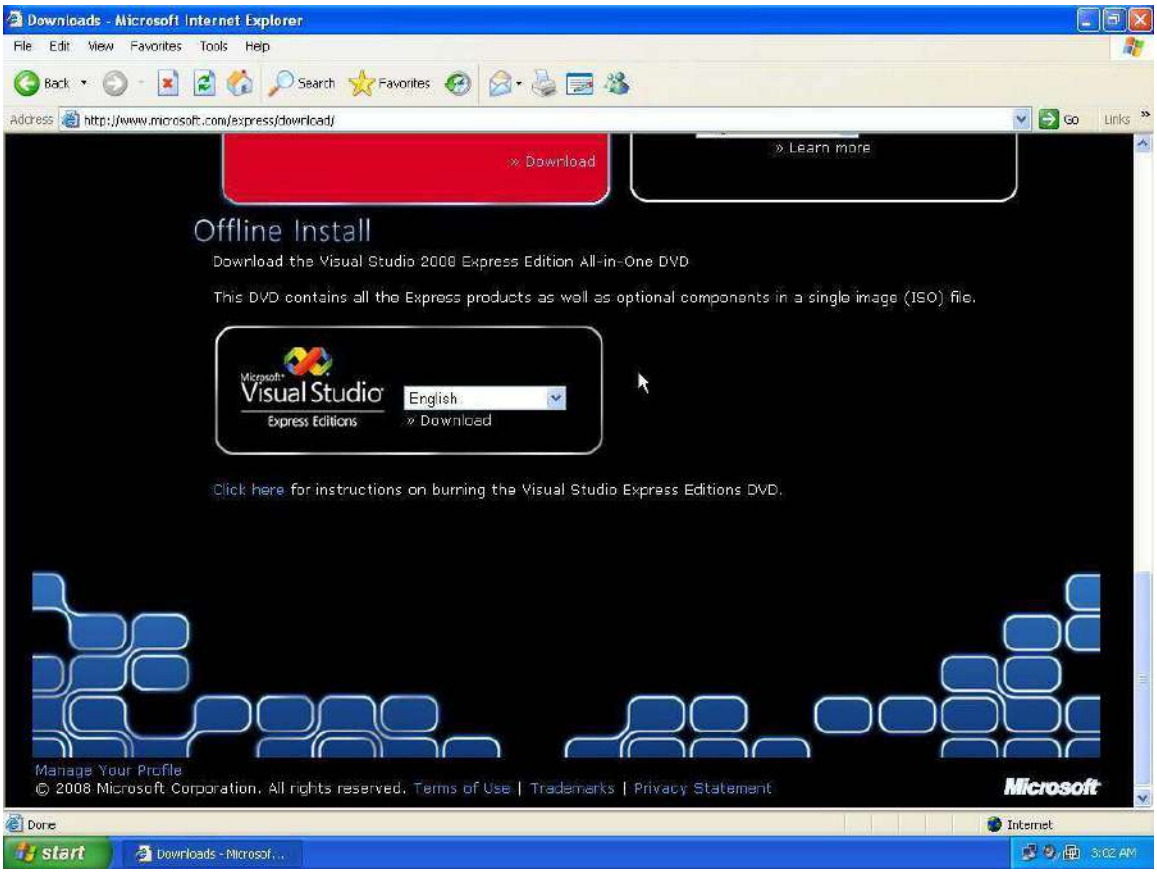
Thank you.

mkaatr

Chapter 1: Installation

Installation

Installation of VB.NET should not be hard to do. The installation might be a little different since Microsoft updates the site from time to time. All you have to do is follow the steps below:



Scrool down and select offine install. You need to burn this ISO image into a disk before using it… or you can select web install for Visual Basic .net…

Select Visual Basic 2008 from the list…

The installation appears after a few seconds

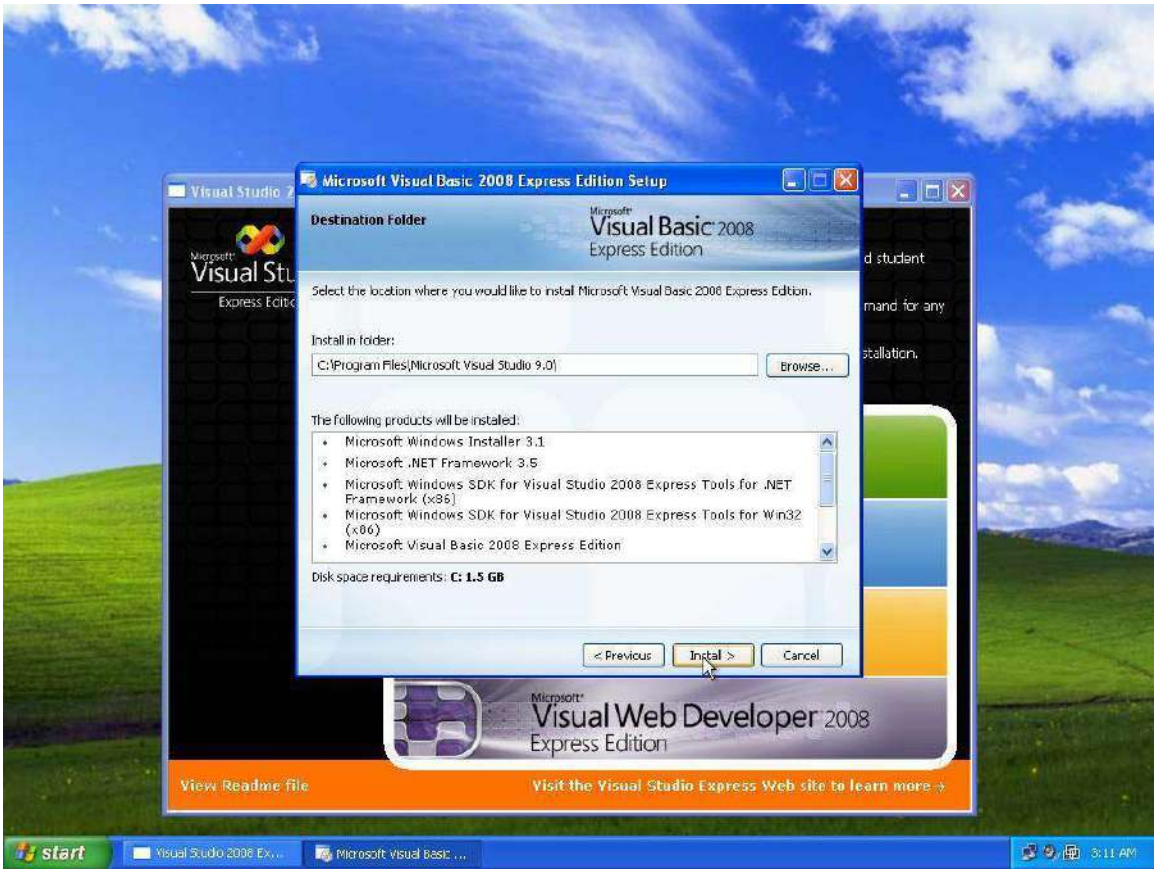
Wait for a while


Microsoft Visual Basic 2008 Express Edition Setup
Welcome to Setup
Visual Basic 2008
Express Edition
Welcome to the Microsoft Visual Basic 2008 Express Edition installation wizard. Microsoft Visual Basic 2008 Express is a fun, simple and easy-to-learn development tool for Visual Basic programmers interested in creating Windows Forms, Windows Presentation Foundation (WPF) as well as class libraries and console-based applications. This wizard will guide you through the installation process. If this product requires any prerequisites that are not currently installed on this computer, you will be able to install those prerequisites as well.
Help Improve Setup
You can submit anonymous information about your setup experiences to Microsoft. To participate, check the box below.
Yes, send information about my setup experiences to Microsoft Corporation.
For more information, click Data Collection Policy
< Previous
Next >
Cancel
Visual Web Developer 2008
Express Edition
View Readme file
Visit the Visual Studio Express Web site to learn more
start
3:10 AM

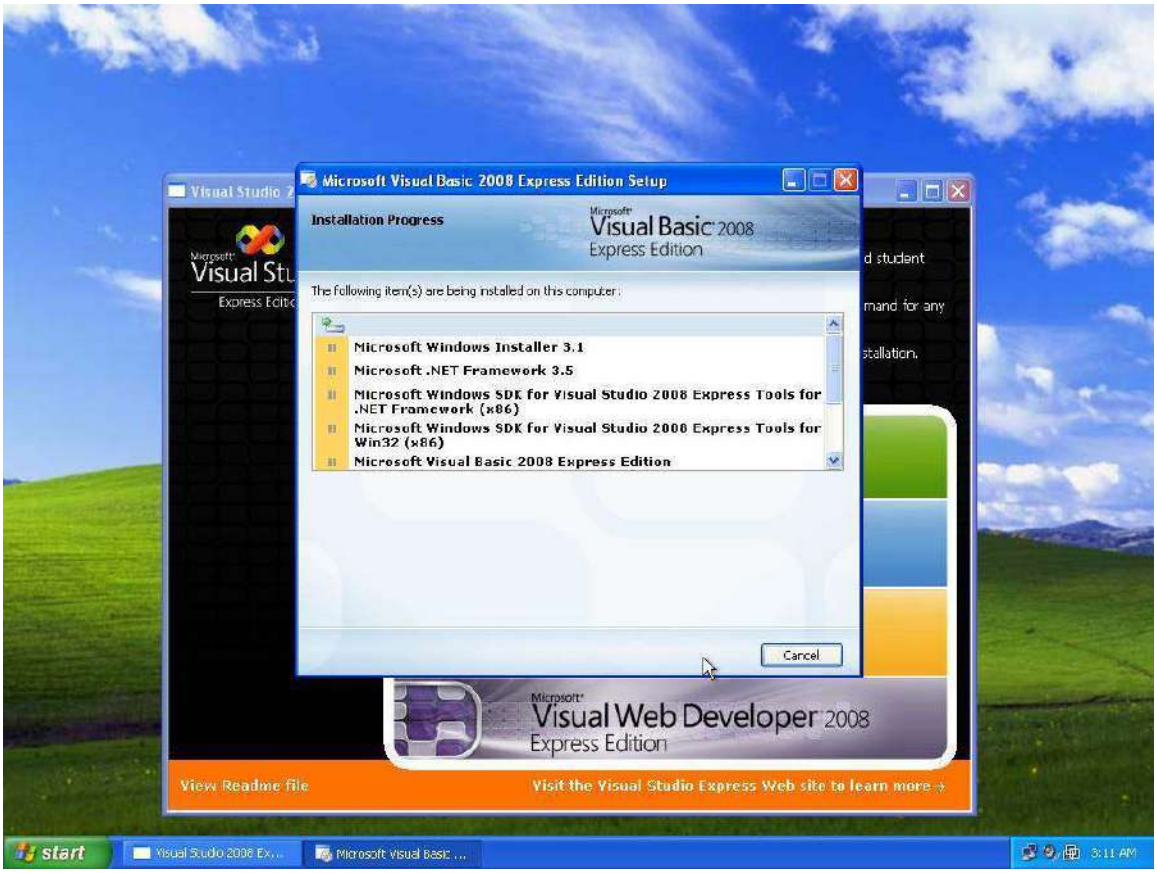
Click next…

Read the agreement and agree about it… then click next…

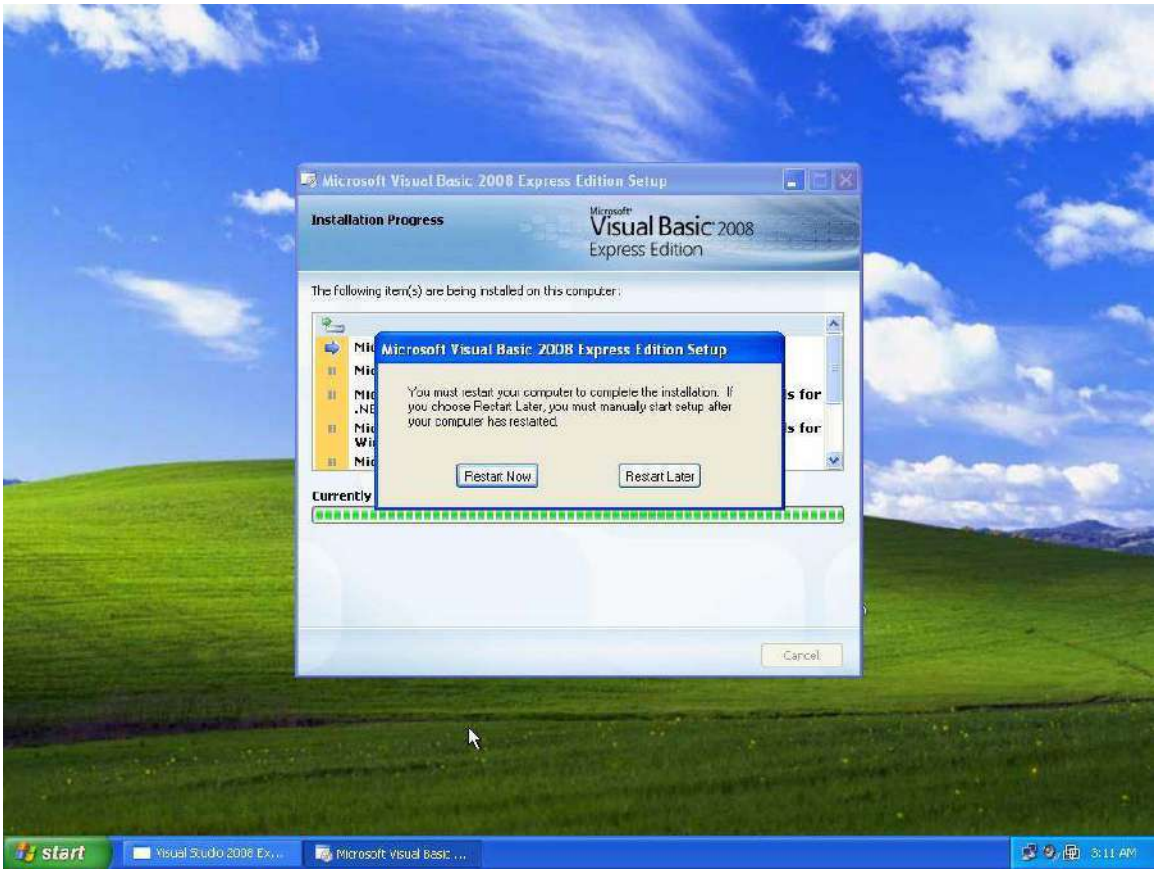
Select the MSDN (the help library) and Silver Light then click next…


Microsoft Visual Basic 2008 Express Edition Setup
Destination Folder
Visual Basic 2008
Express Edition
Select the location where you would like to install Microsoft Visual Basic 2008 Express Edition.
Install in folder:
C:\Program Files\Microsoft Visual Studio 9.0\
Browse...
The following products will be installed:
Microsoft Windows Installer 3.1
Microsoft .NET Framework 3.5
Microsoft Windows SDK for Visual Studio 2008 Express Tools for .NET Framework (x86)
Microsoft Windows SDK for Visual Studio 2008 Express Tools for Win32 (x06)
Microsoft Visual Basic 2008 Express Edition
Disk space requirements: C: 1.5 GB
< Previous
Install >
Cancel
Visual Web Developer 2008
Express Edition
View Readme file
Visit the Visual Studio Express Web site to learn more
start

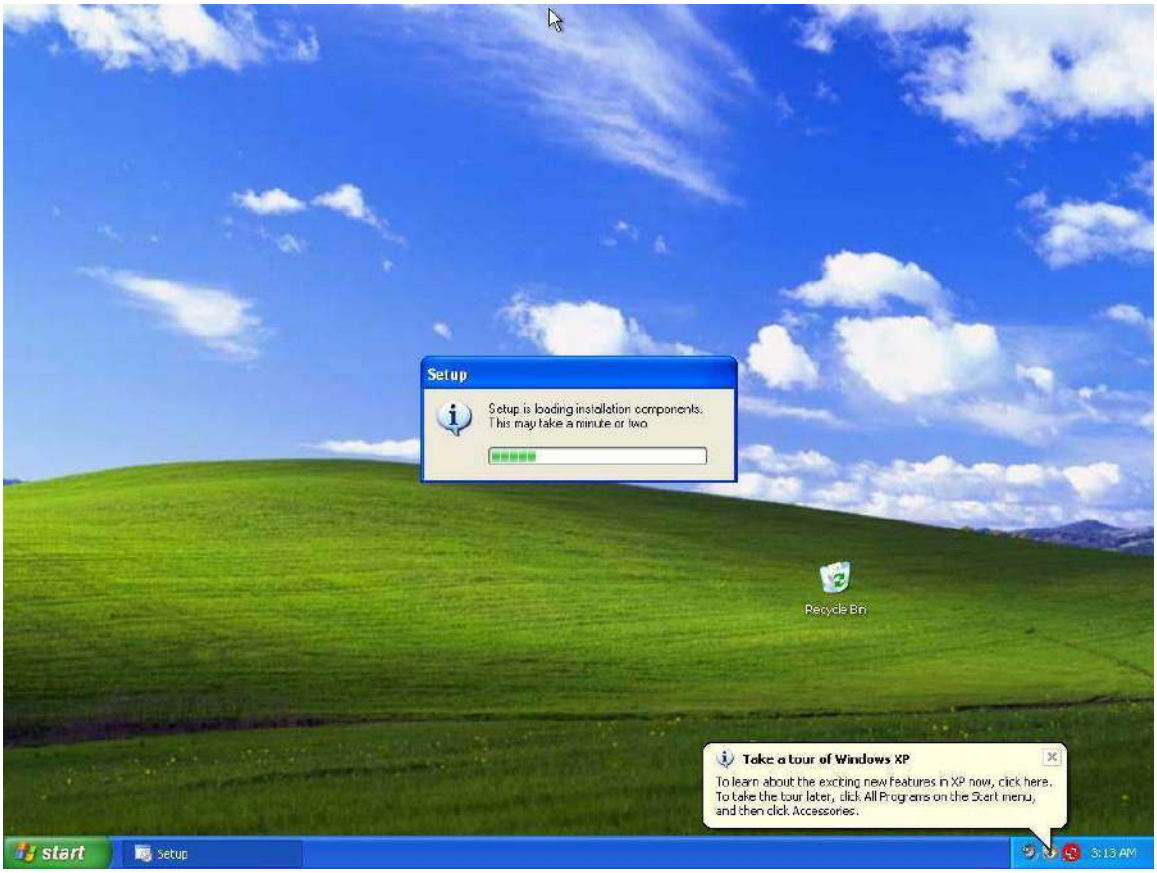
Click next…



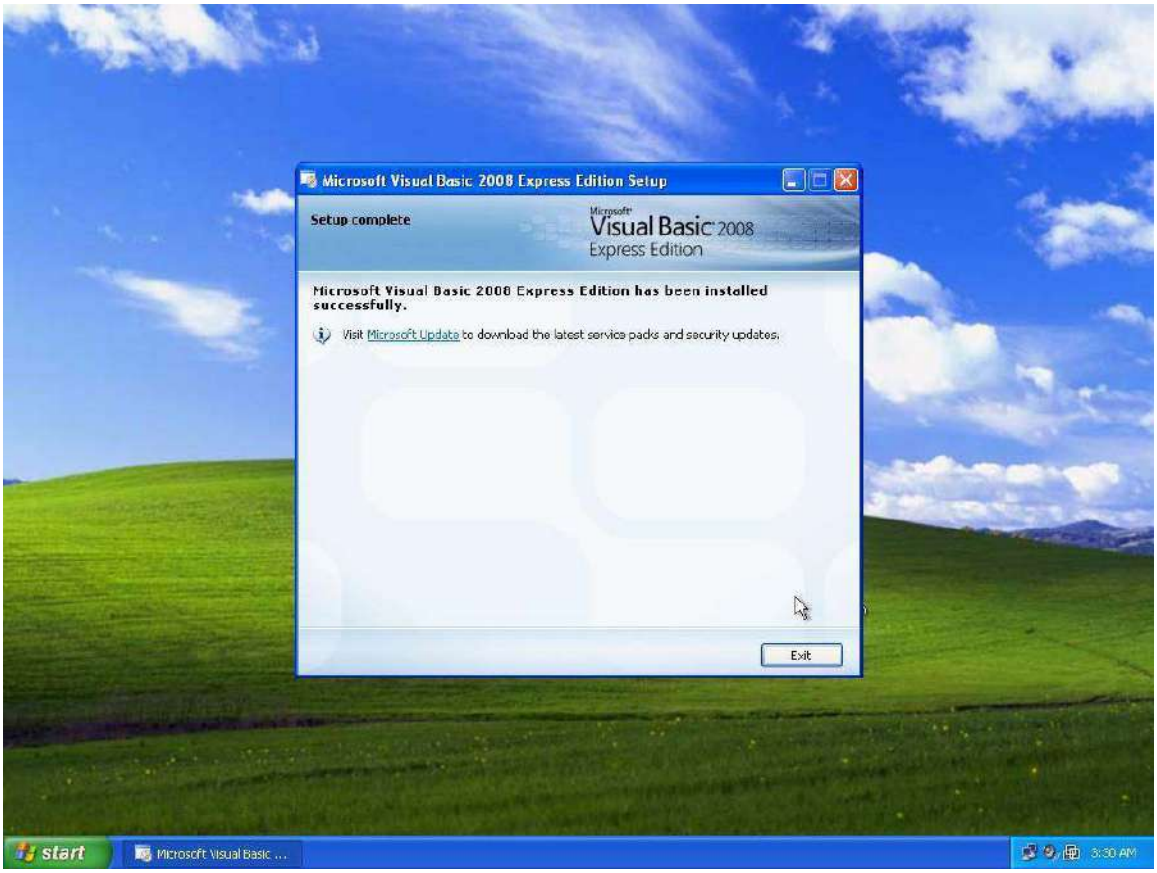
The installation starts…



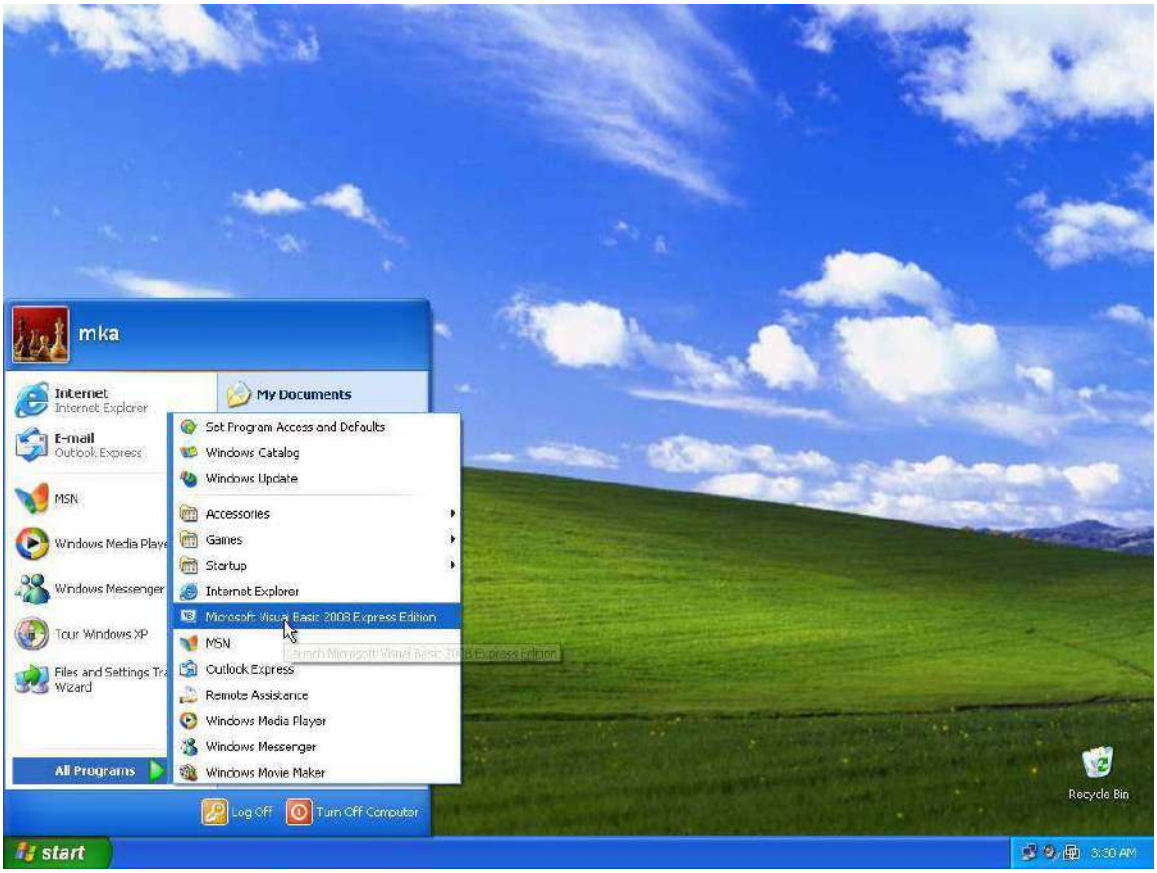
After a while a restart will be required… click restart now…



Do not press or do anything… the installation continues automatically…

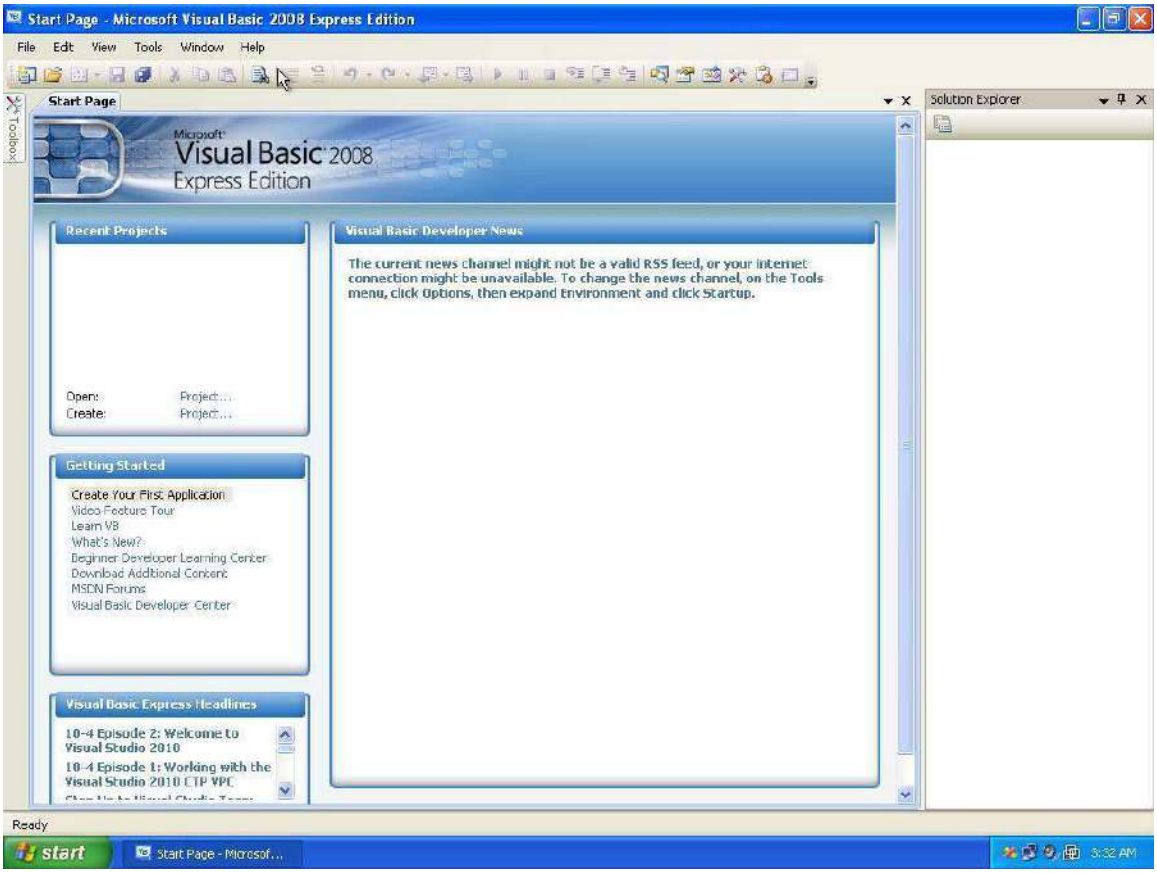


After the installation finishes, press exit… now let us test the VB.Net and write a simple application.

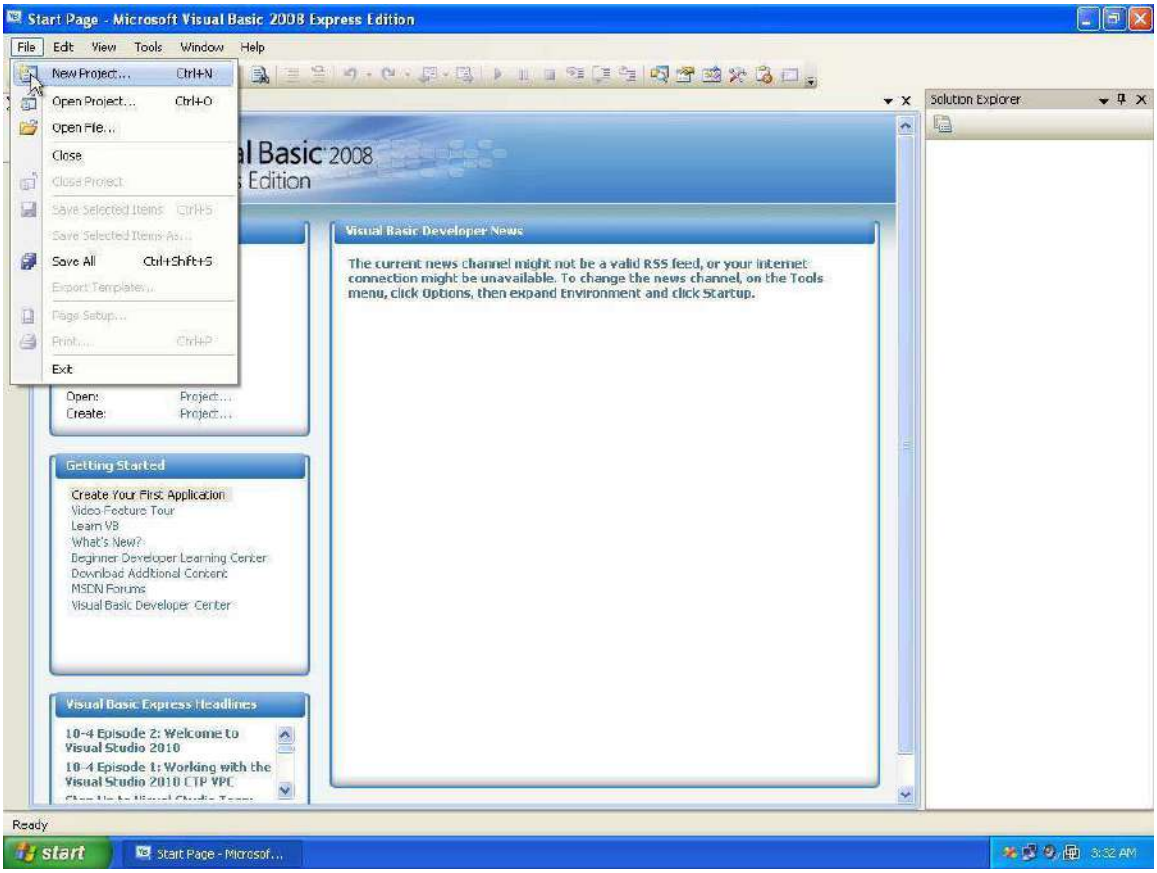


Go to All Programs->Microsoft Visual Basic 2008…

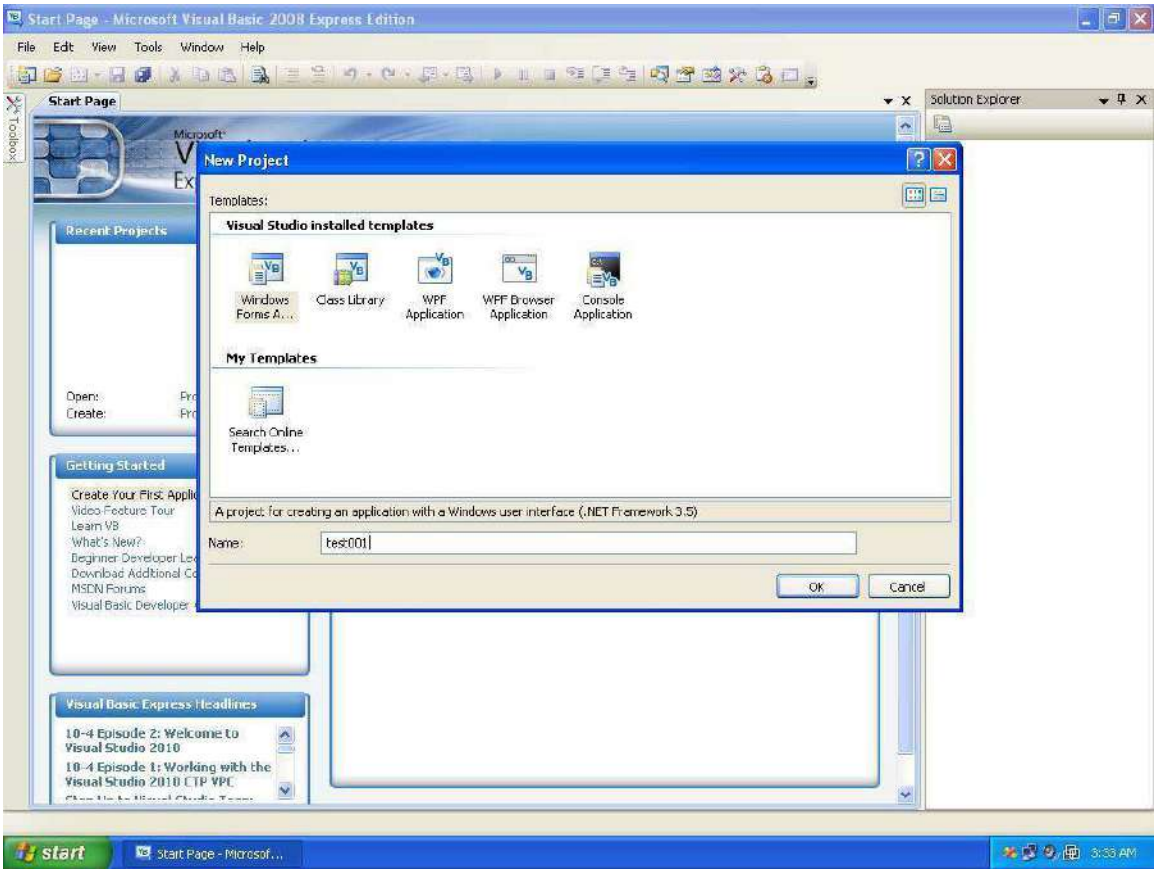
VB will start…



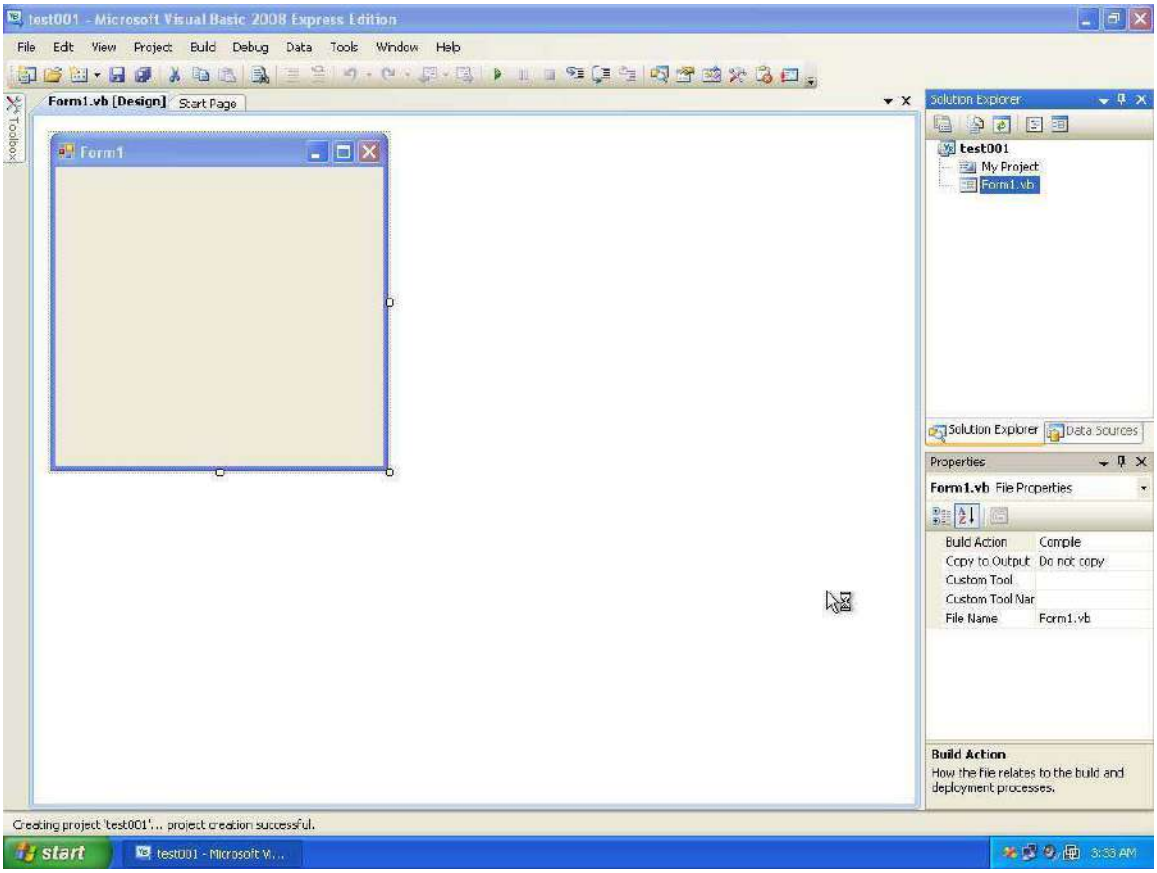
This is only a quick test… the next lessons explains what happens in detail… so for now this is the IDE (Integrated Development Environment) that you build your applications with…



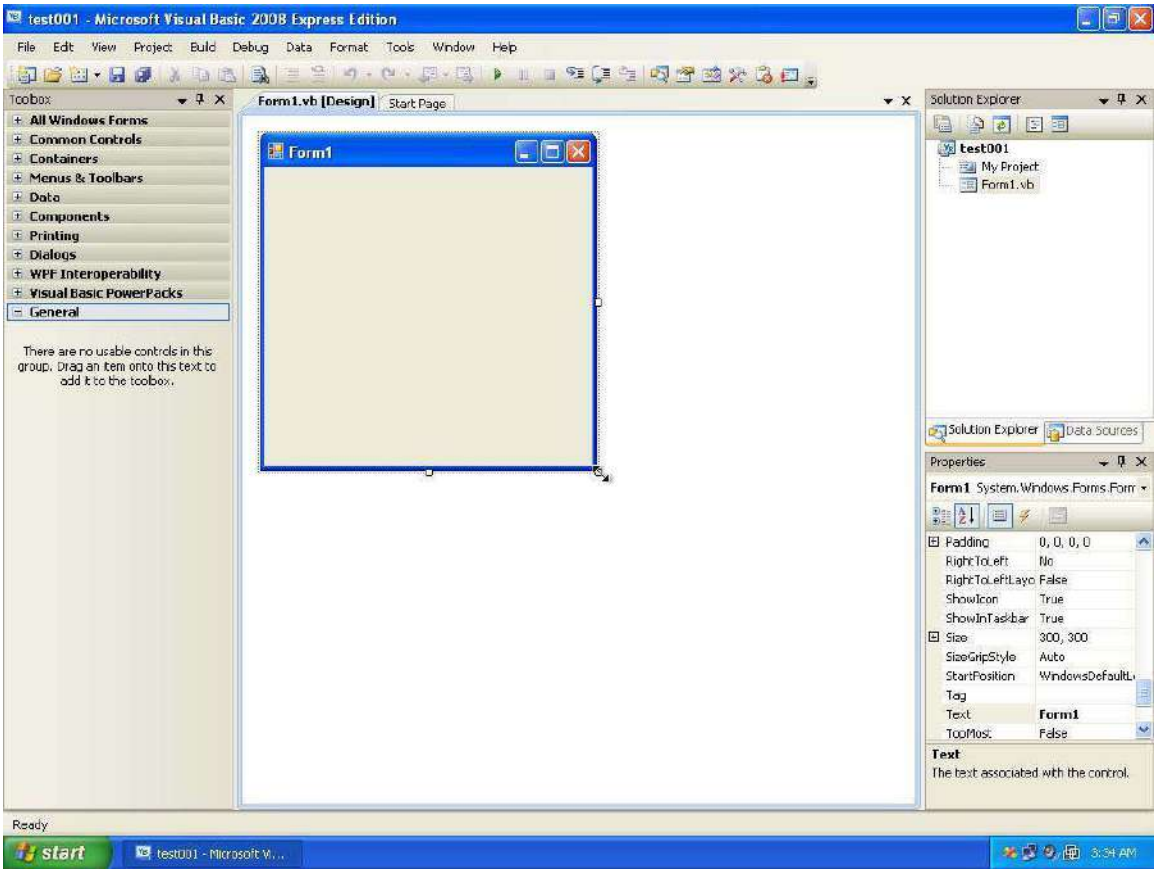
Select File->New Project



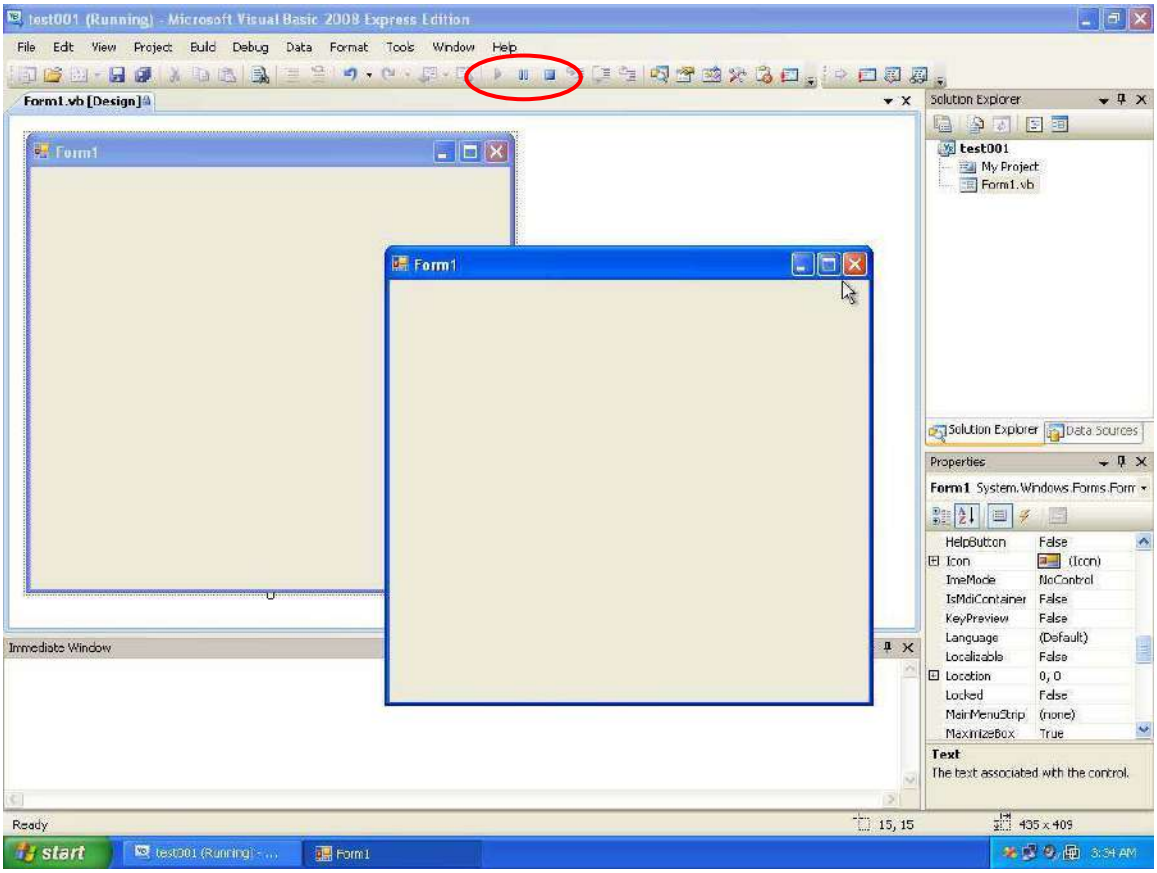
Select Windows Forms, and give a name to your first application, then press OK.



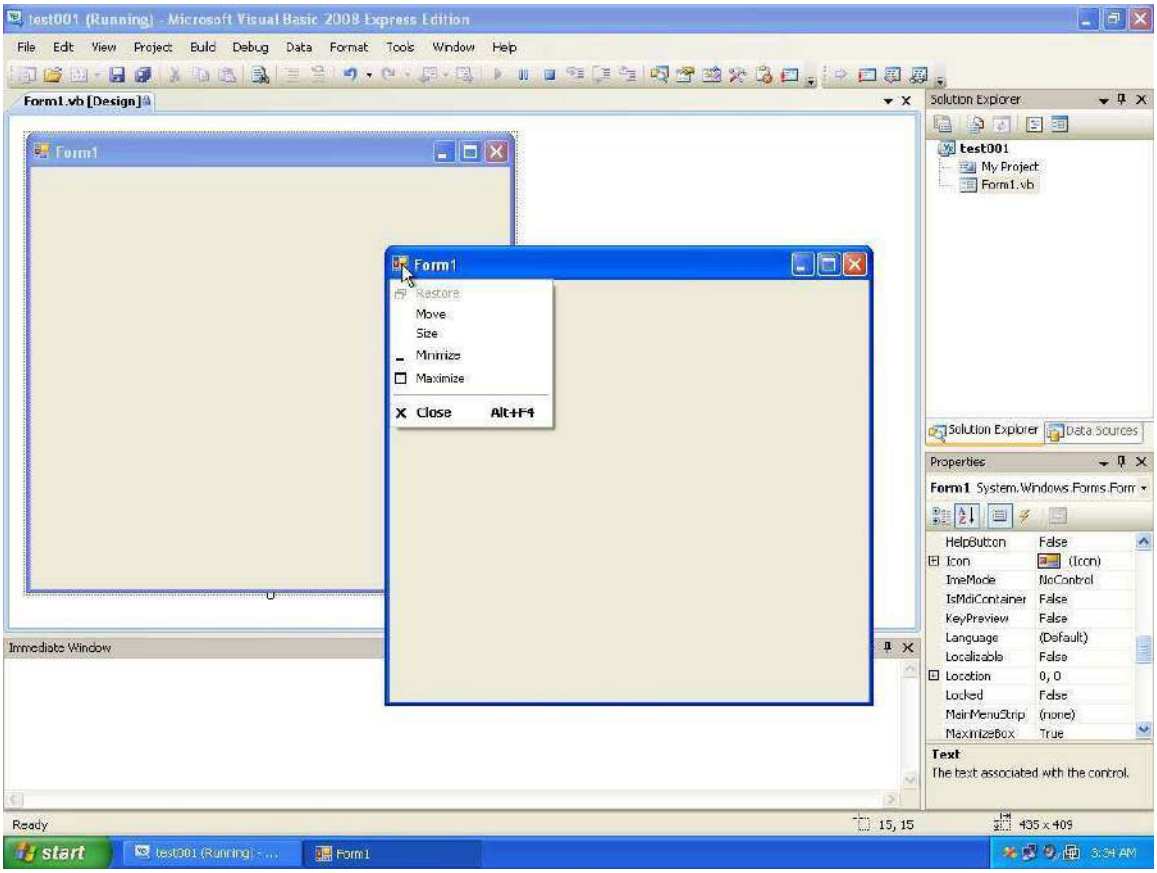
Your application contains one window… you can modify it visually without doing any programming…



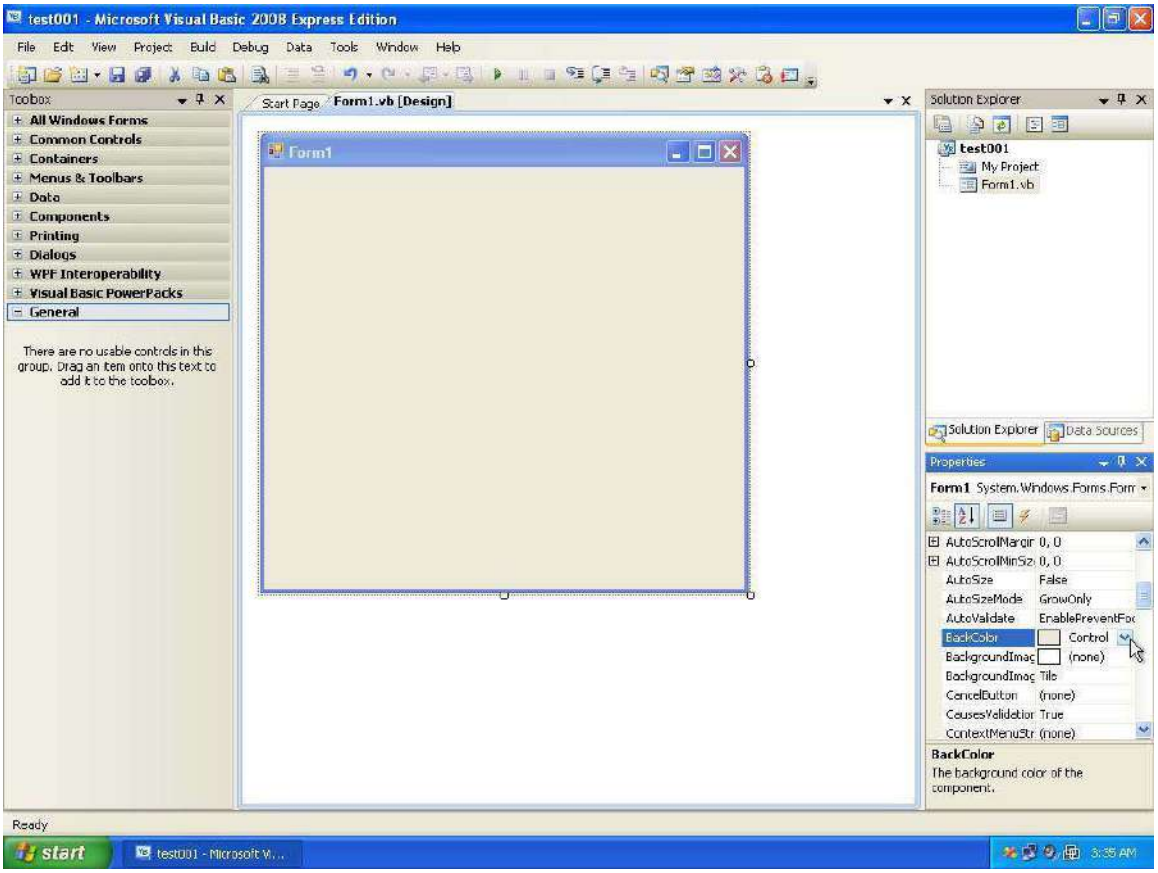
Put the mouse pointer over the small square on the corner, and drag the window's corner and see what happens … you are resizing the window very easily…



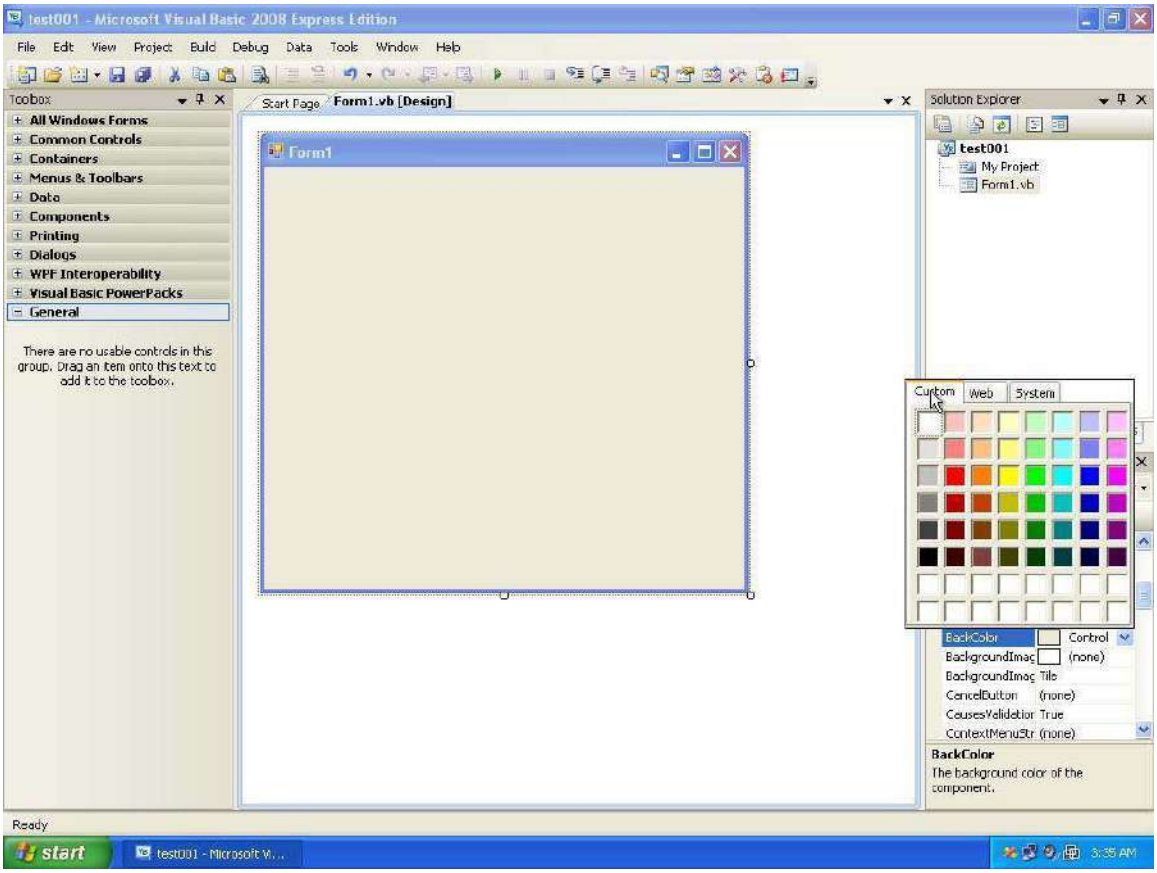
To test this window, you hit the play button… the play button runs your application, to stop the application hit the stop button or close the application window…



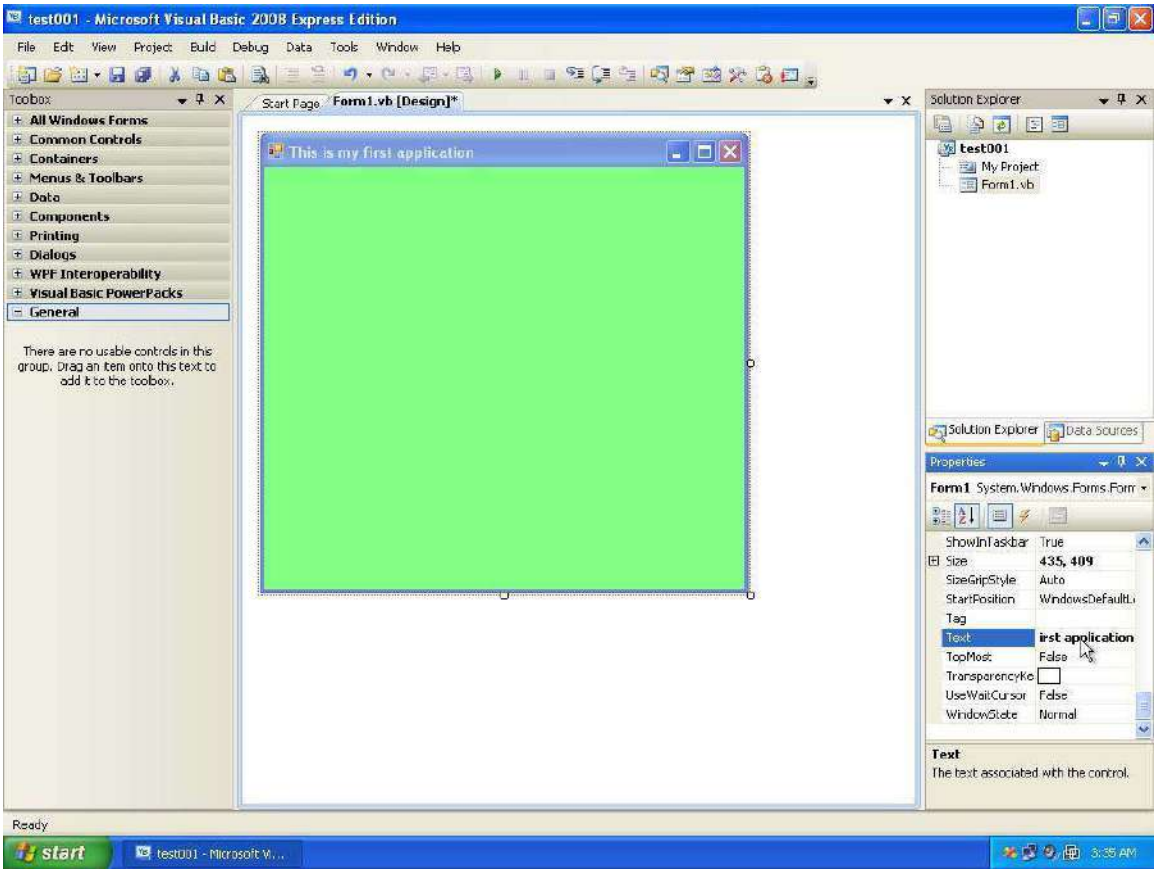
You can test the window and see it behaves like many XP forms or Vista forms…



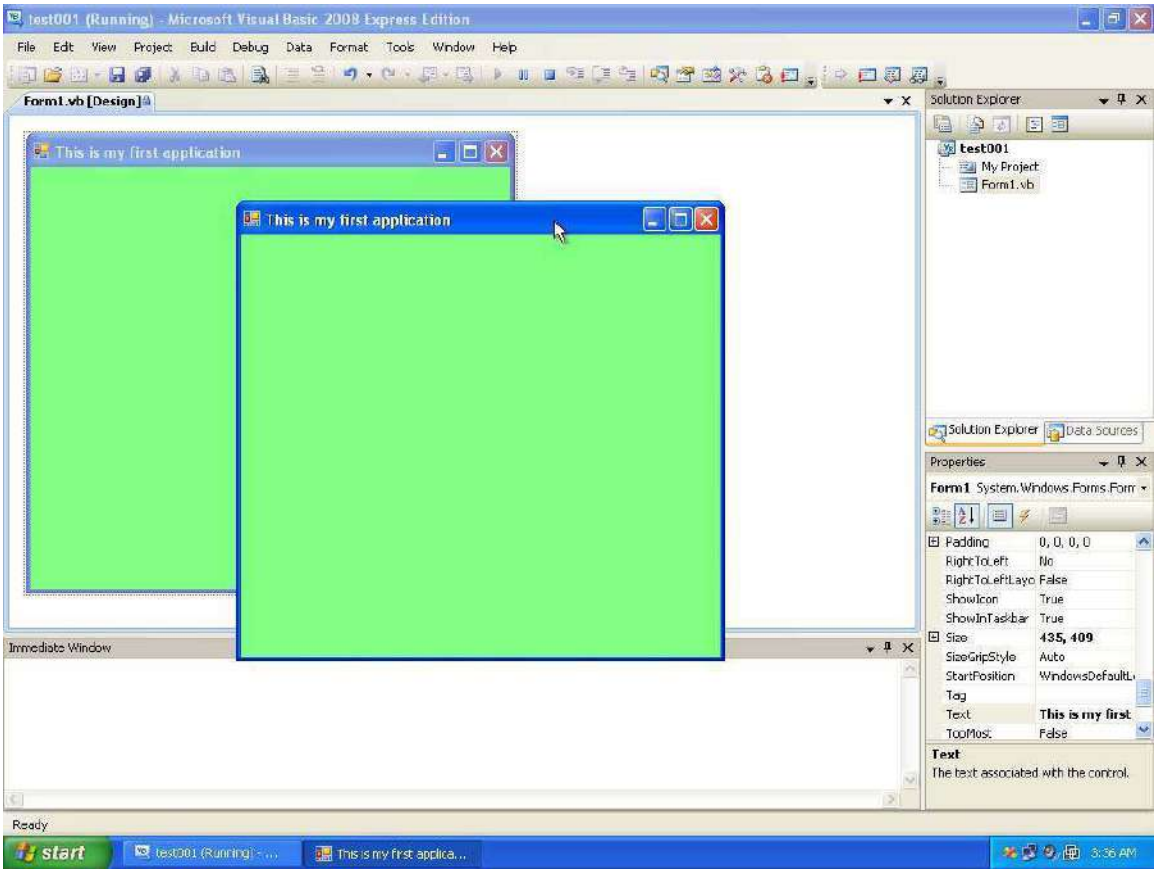
Now stop the application and search for Back color property from the list on the right…



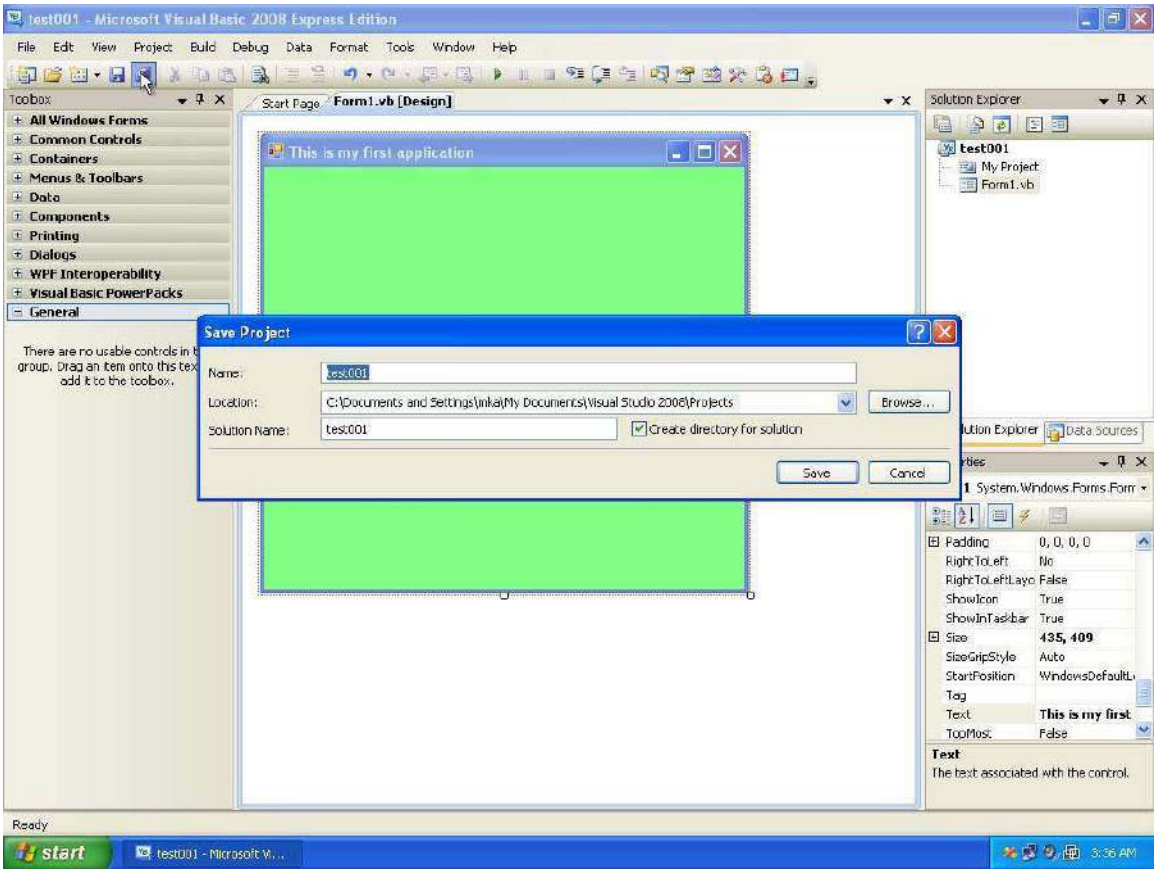
Press the small arrow that appears, and select General as shown In the image above… then select a color that you like.



Now search for the Text property and replace its value with anything you like …



Run the application…

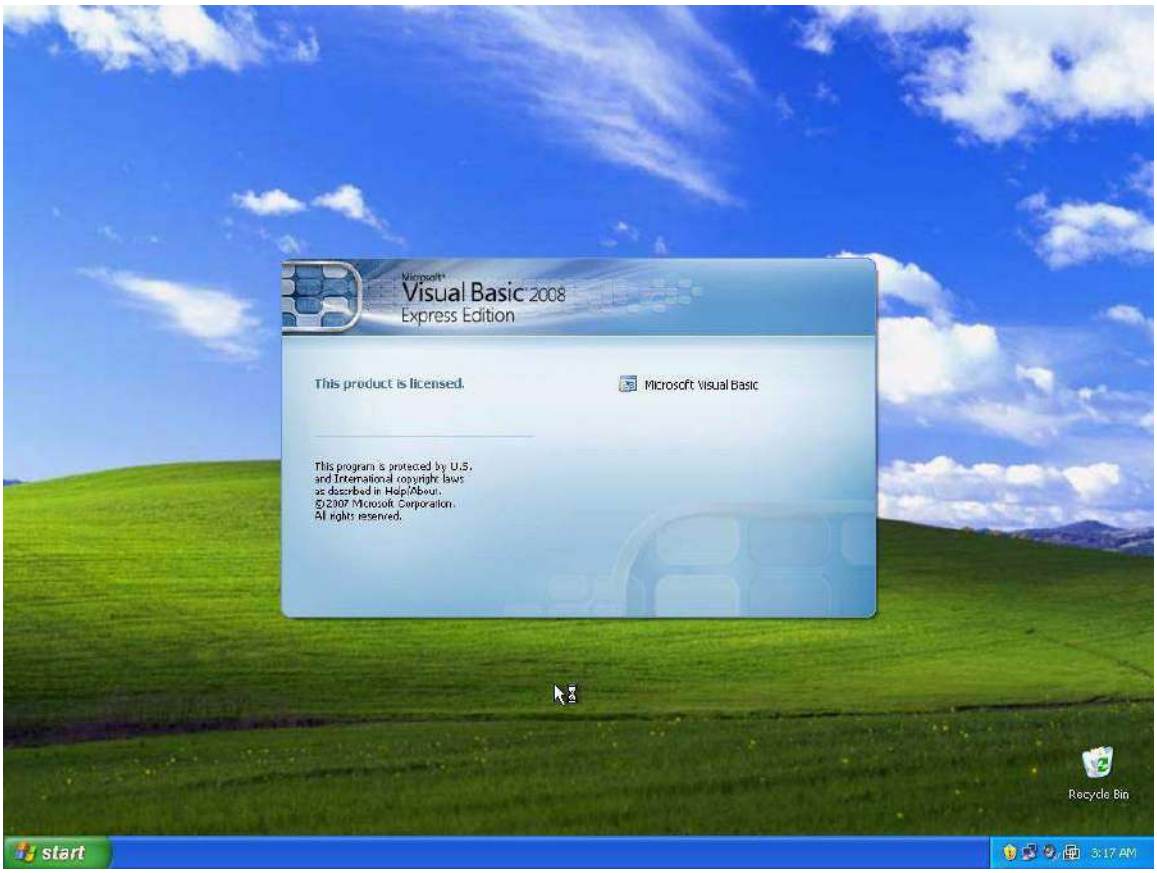


Very easy… still one last thing… you should save the application. Just Hit the Save all icon from the tool bar…and select save…

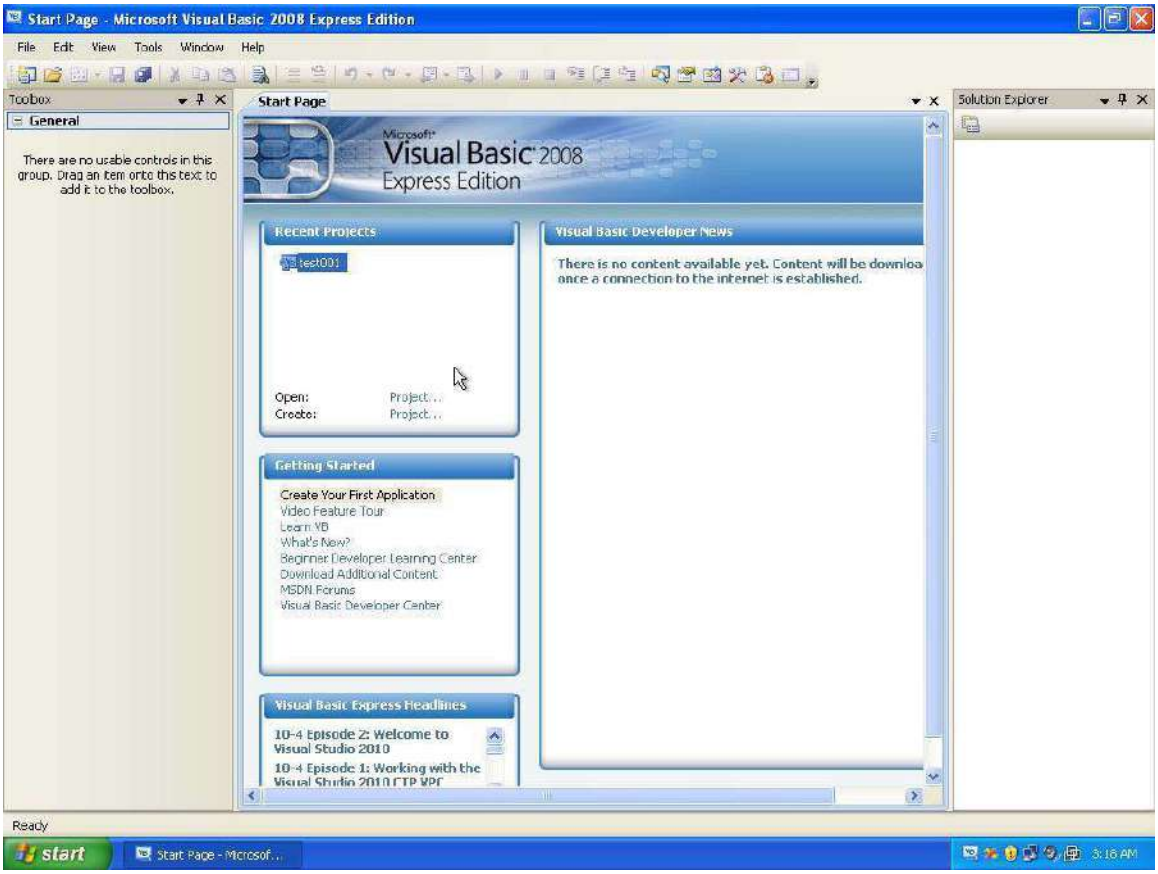
Chapter 2: Understanding the IDE

Understanding the IDE

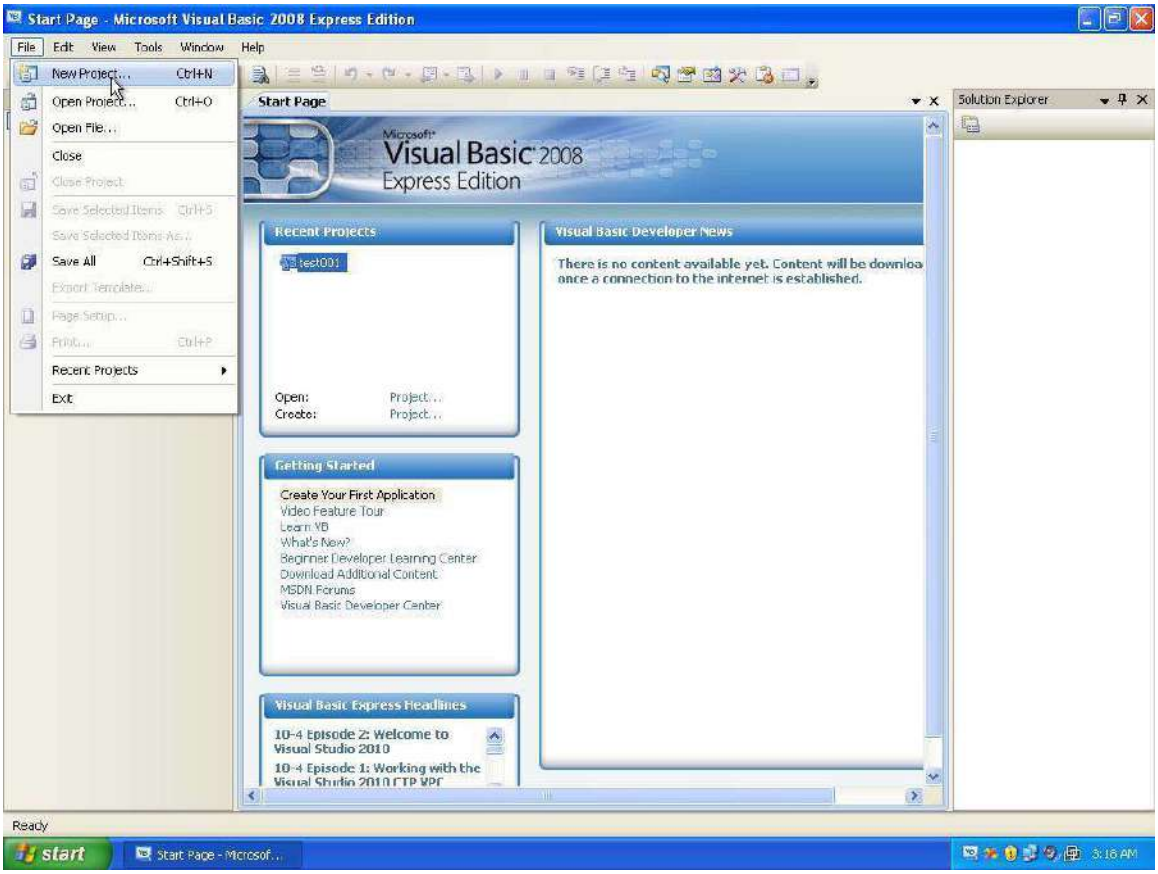
In this chapter, we will understand a little bit about the IDE, and what does it provide, so later on it will be easier for us to work on applications,



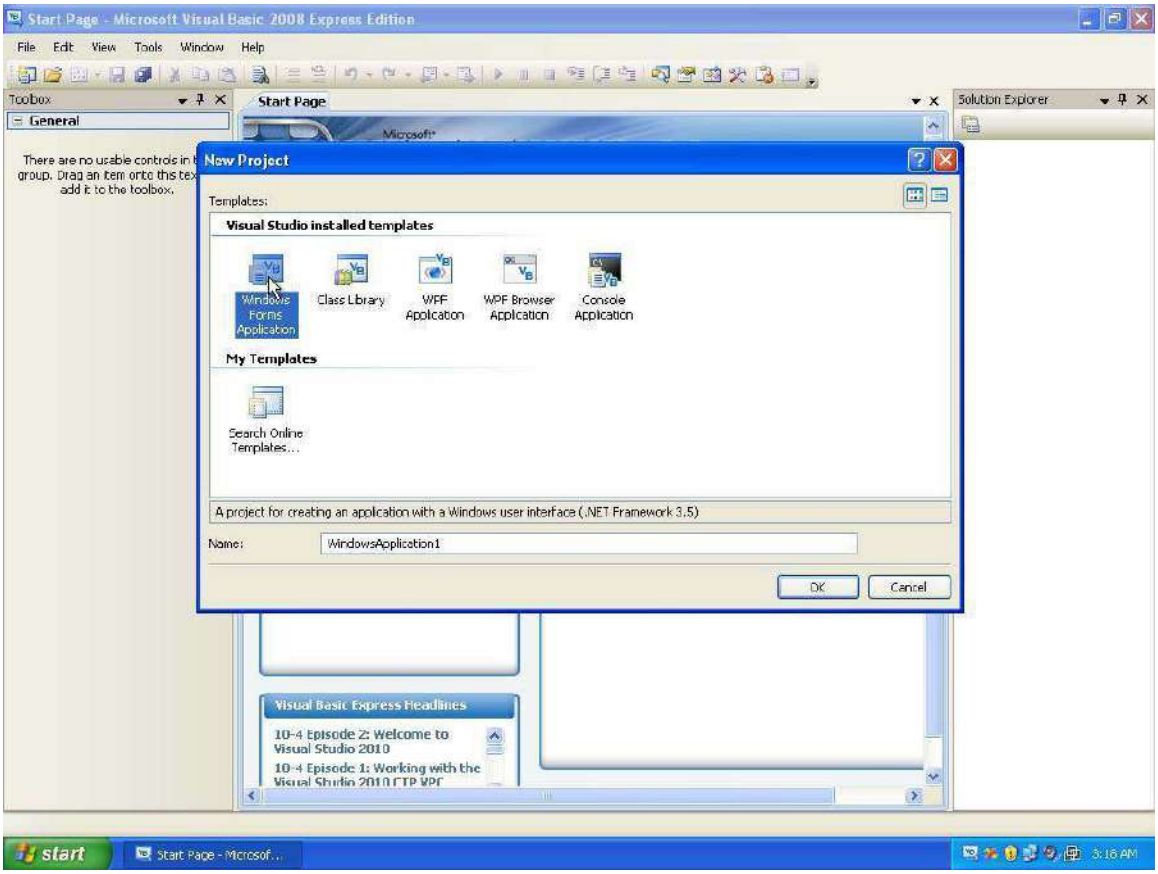
Open VB.NET



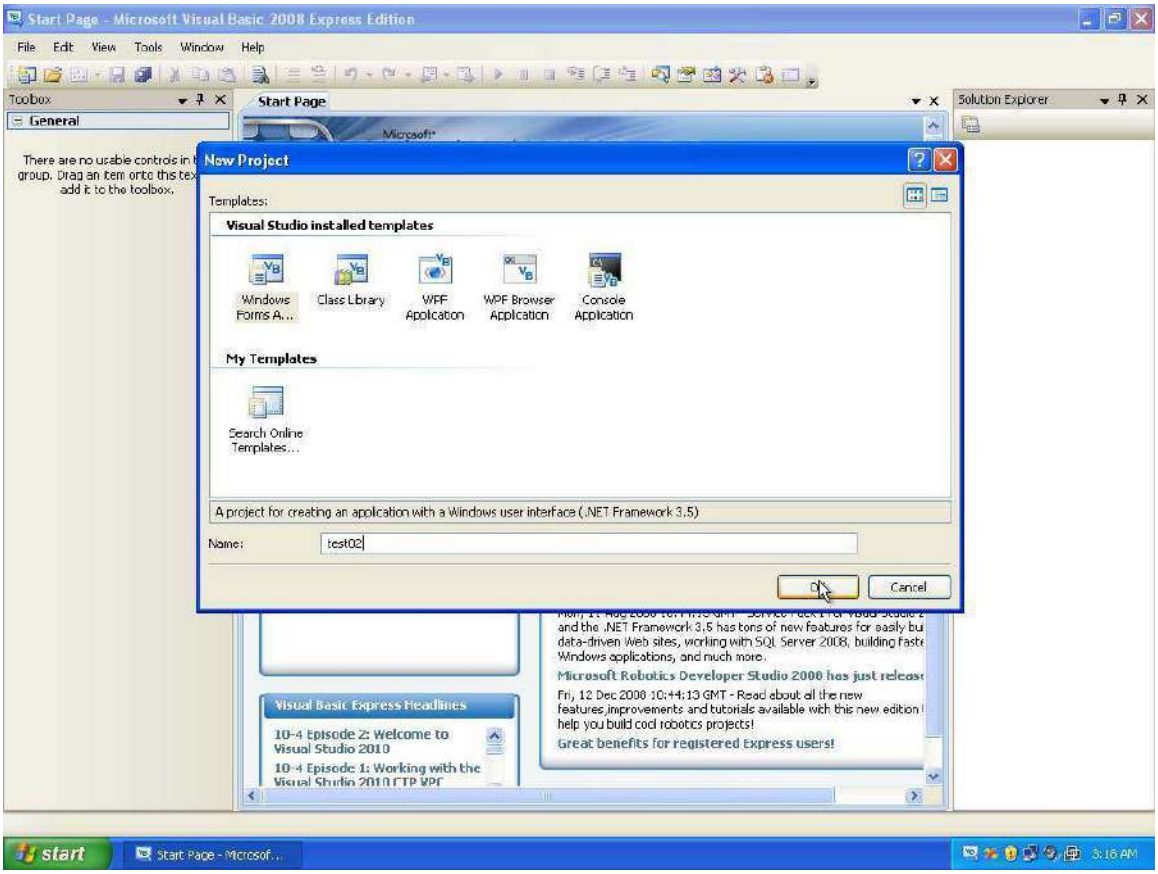
Now this is the IDE, in order to work with it, there should be an application, so we will create a new application in order to discover the IDE, and see what does it provide and how does it help us.



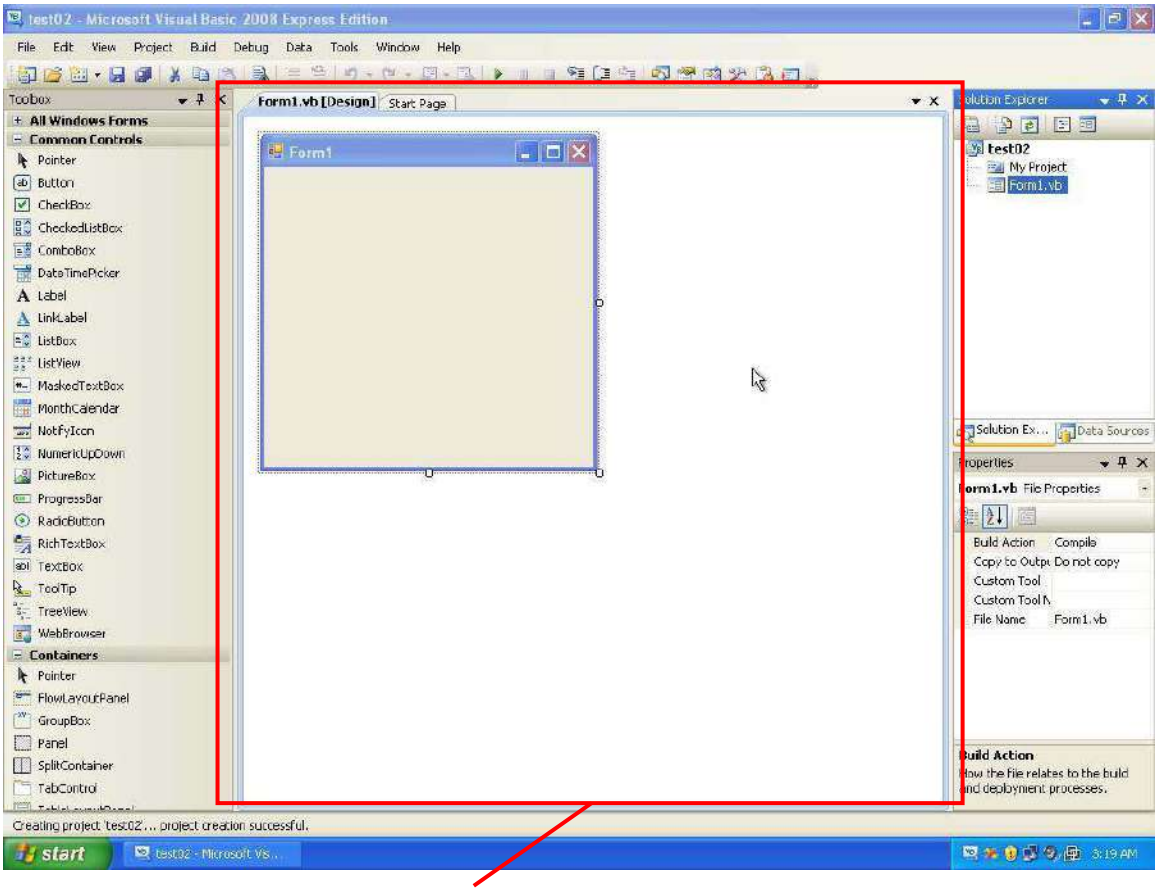
Open a new project



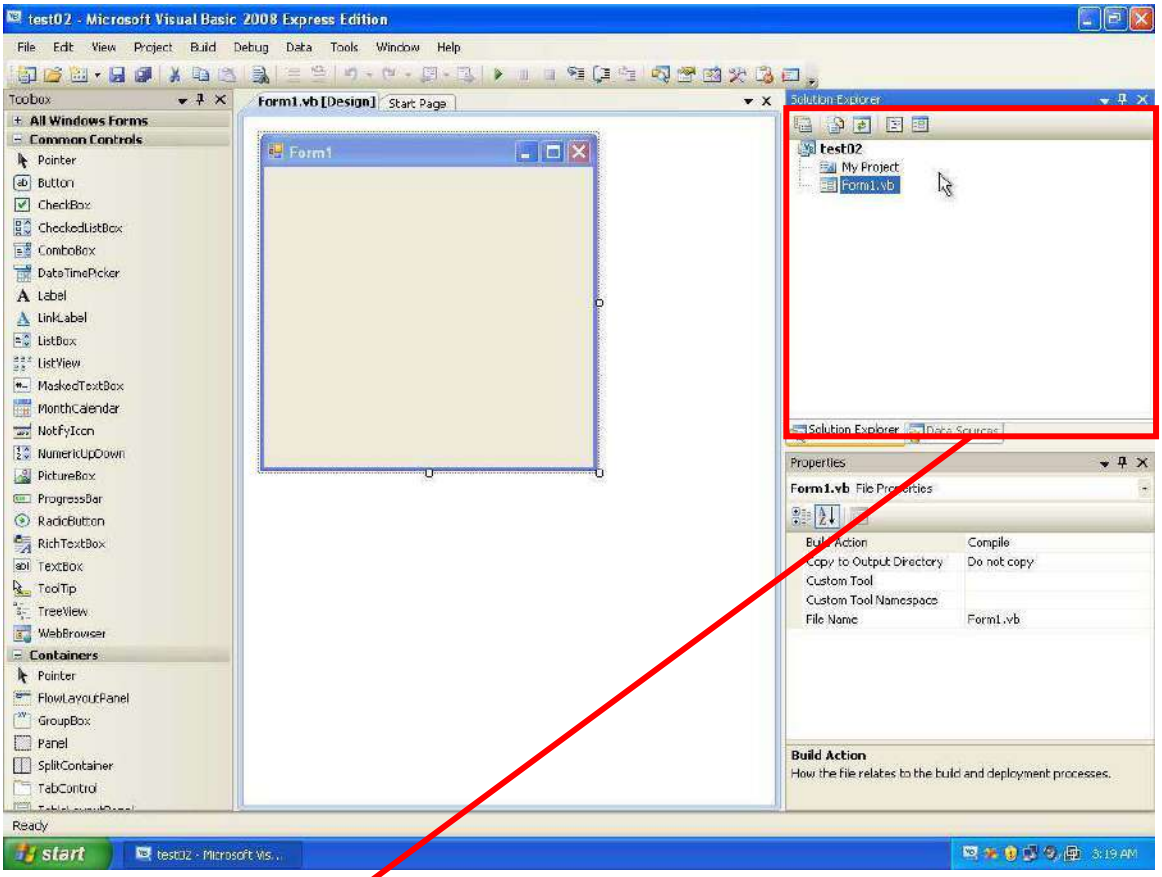
The new project dialog box appears, select windows forms application, and supply a name for your project


Start Page - Microsoft Visual Basic 2008 Express Edition
File Edit View Tools Window Help
Toolbox
General
There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.
Start Page
Solution Explorer
New Project
Templates:
Visual Studio installed templates
Windows Forms A...
Class Library
WPF Application
WPF Browser Application
Console Application
My Templates
Search Online Templates...
A project for creating an application with a Windows user interface (.NET Framework 3.5)
Name:
test02
OK
Cancel
Visual Basic Express Headlines
10-4 Episode 2: Welcome to Visual Studio 2010
10-4 Episode 1: Working with the Visual Studio 2010 CTP VPC
Microsoft Robotics Developer Studio 2008 has just release
Great benefits for registered Express users!
start
Start Page - Microsof...
3:16 AM

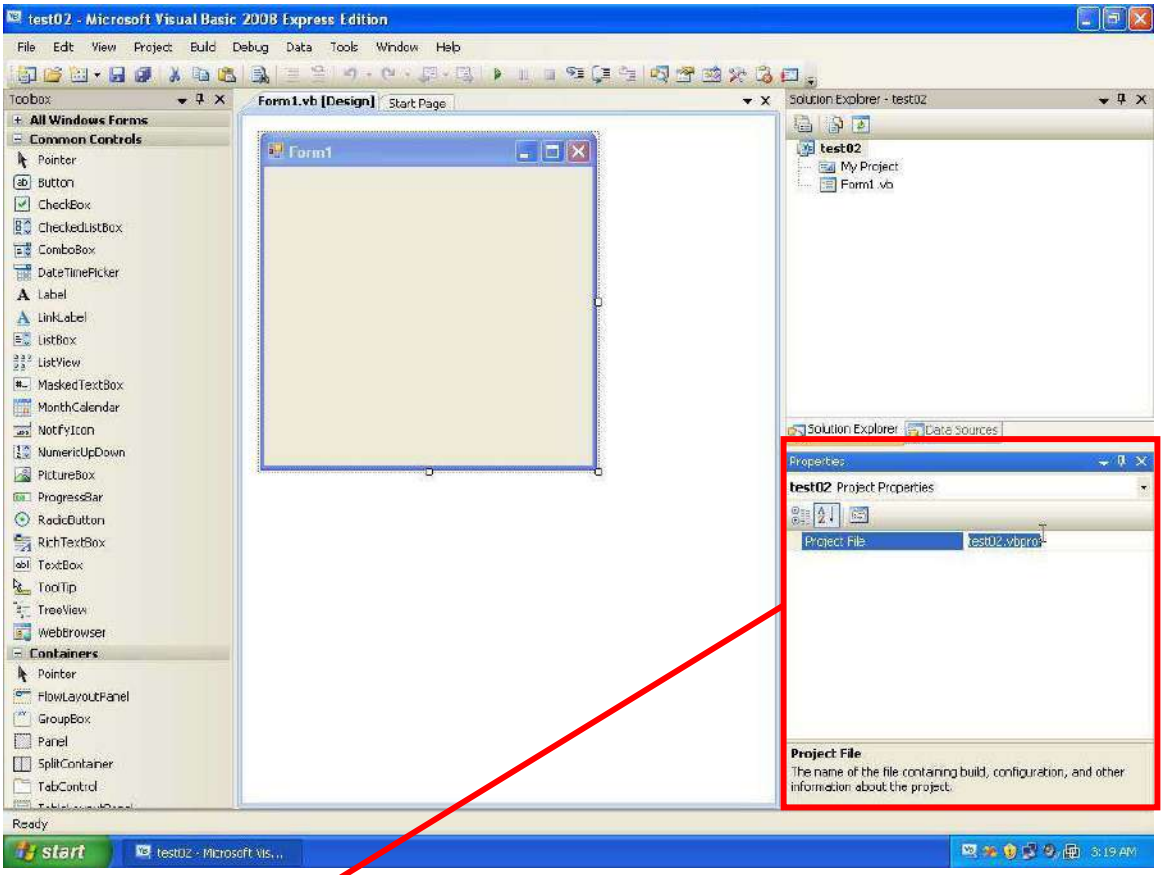
Press OK



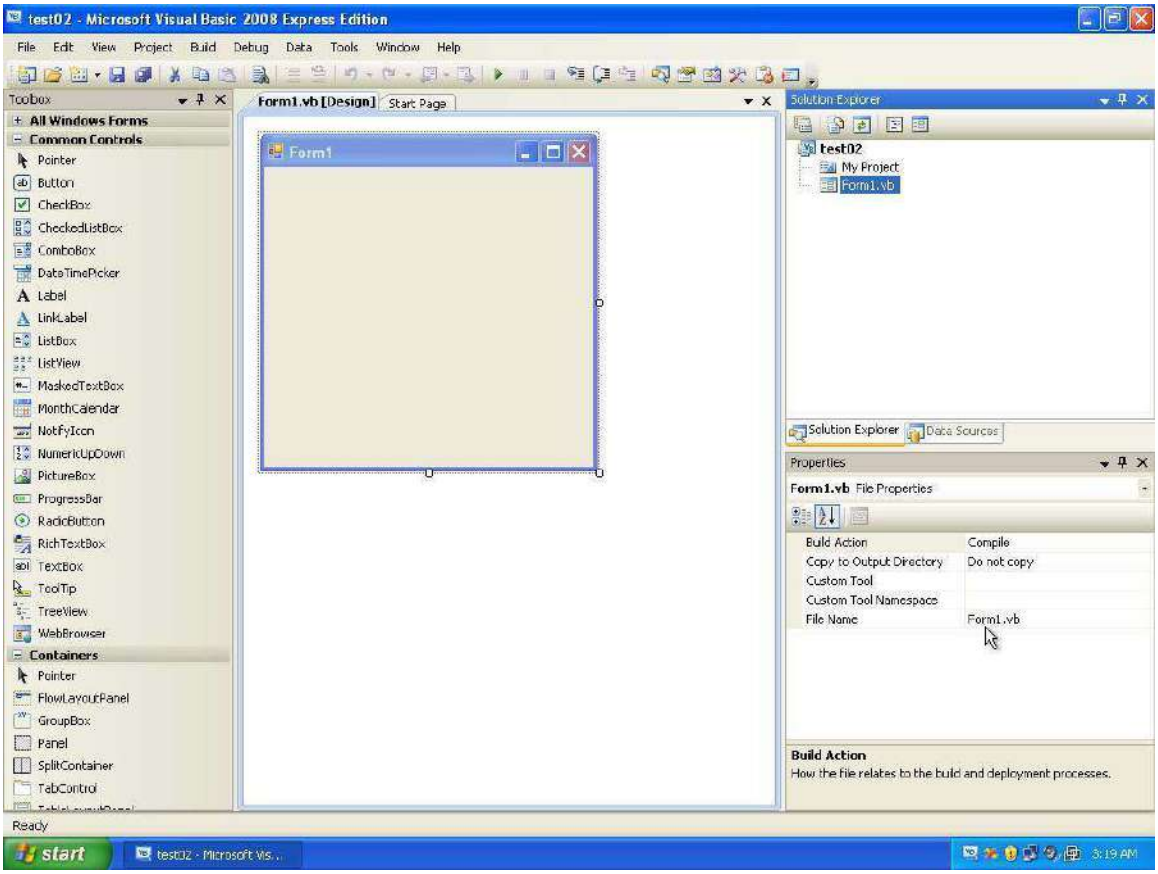
Now you see the IDE. The central part is your working area, there you write code, design the user interface, and do many other things. What you are seeing now is the how the user interface looks like.



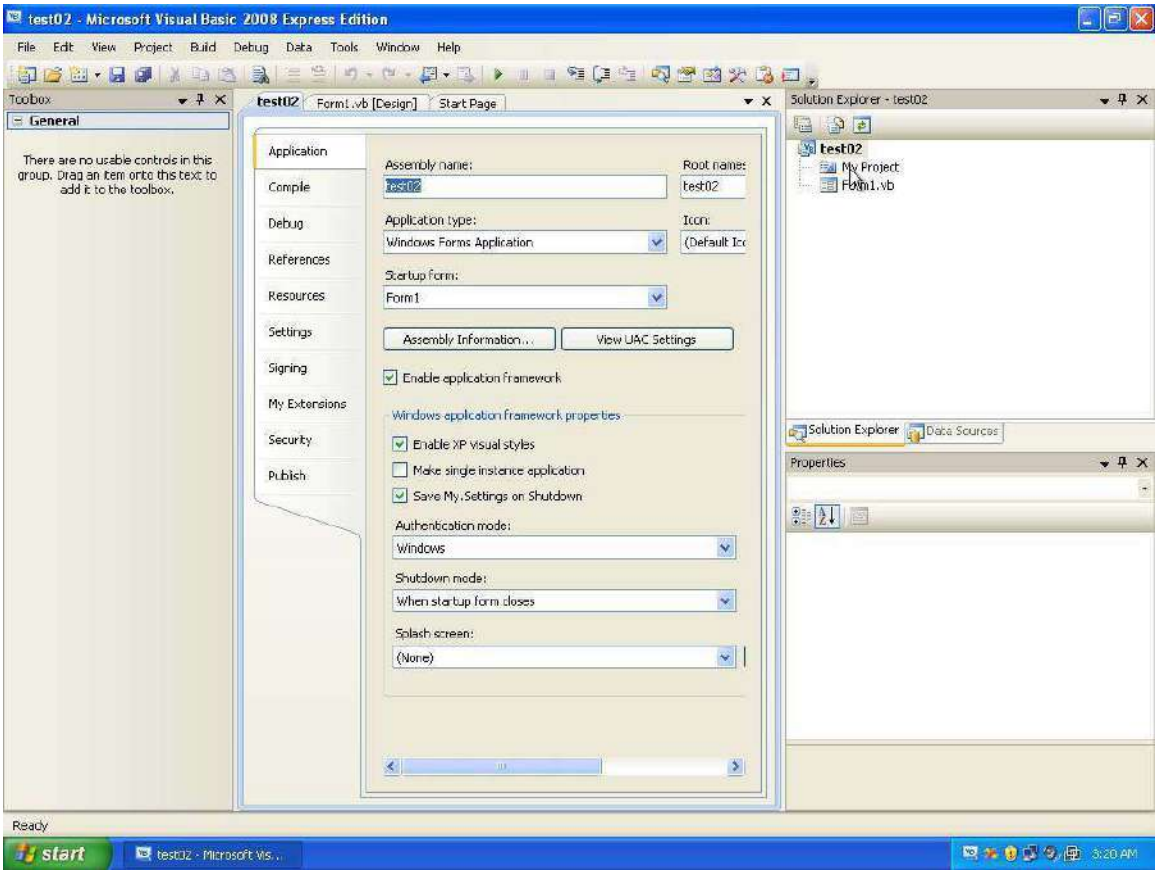
Next part is the solution explorer. This part you can see the main files that your application is consisting of. You use it to quickly move from one part of the application to the other.



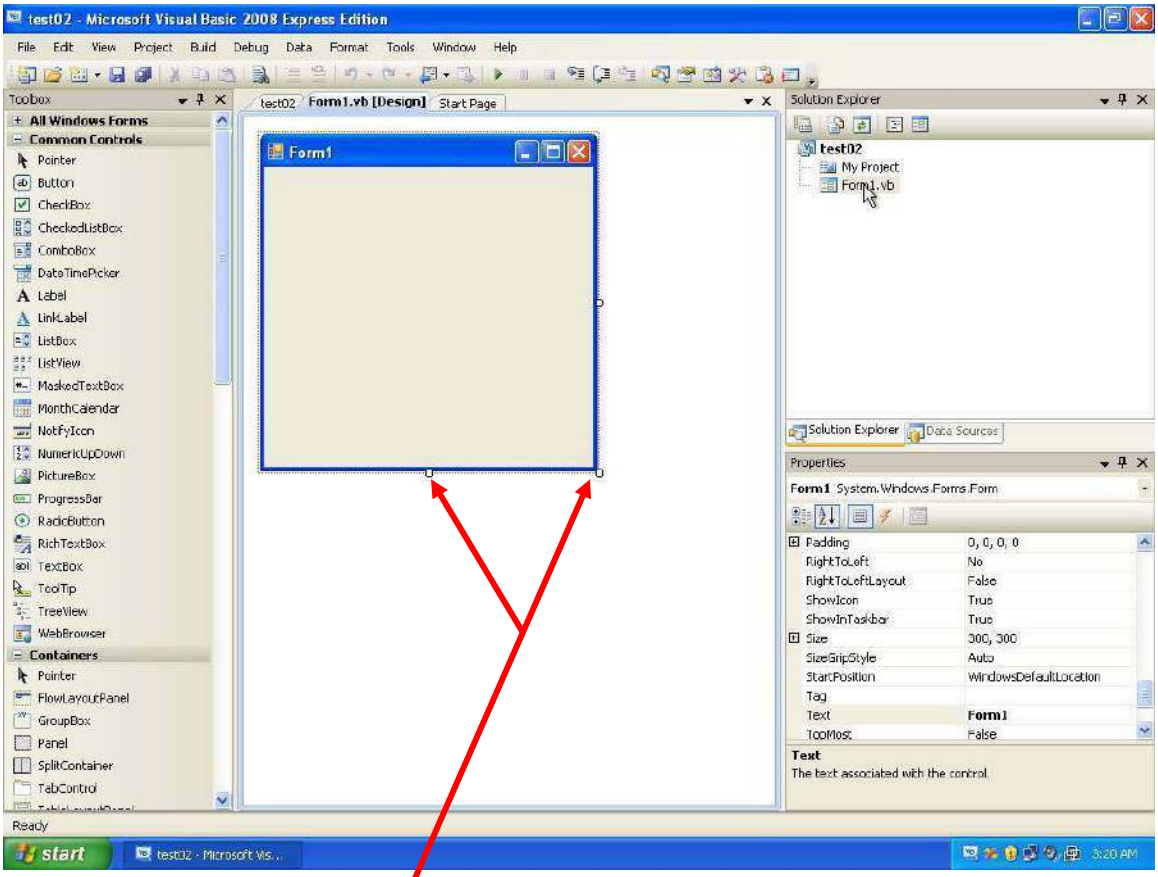
Next is the properties window. This window changes according to what you are doing right now. For example, whenever you select a file from the Solution Explorer window, it changes itself to show you only the properties related to that specific file. It works the same way with the graphical user interface in the central area (the GUI).



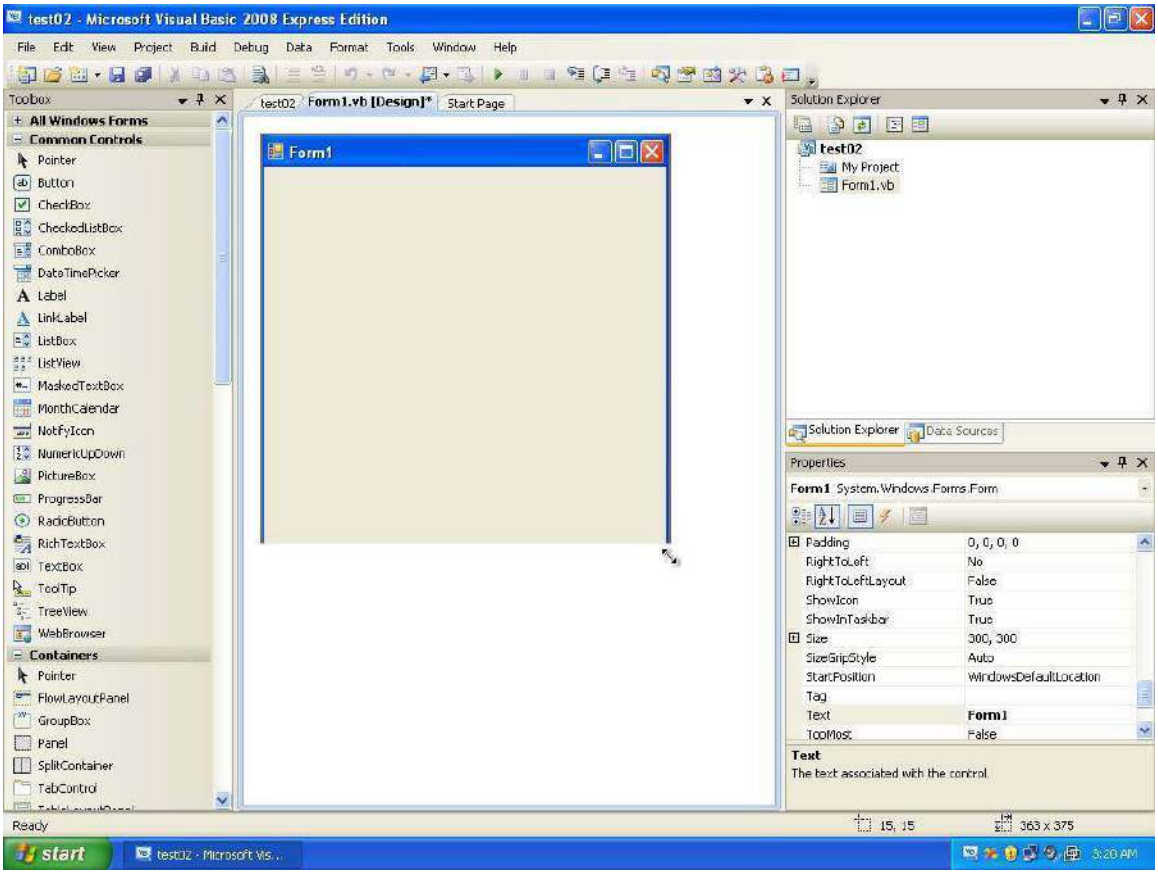
Select Form1.vb from within the solution explorer window by clicking it once and see how the properties change to view the relevant information. Try to select My project again by clicking it one and see how the properties changes again.



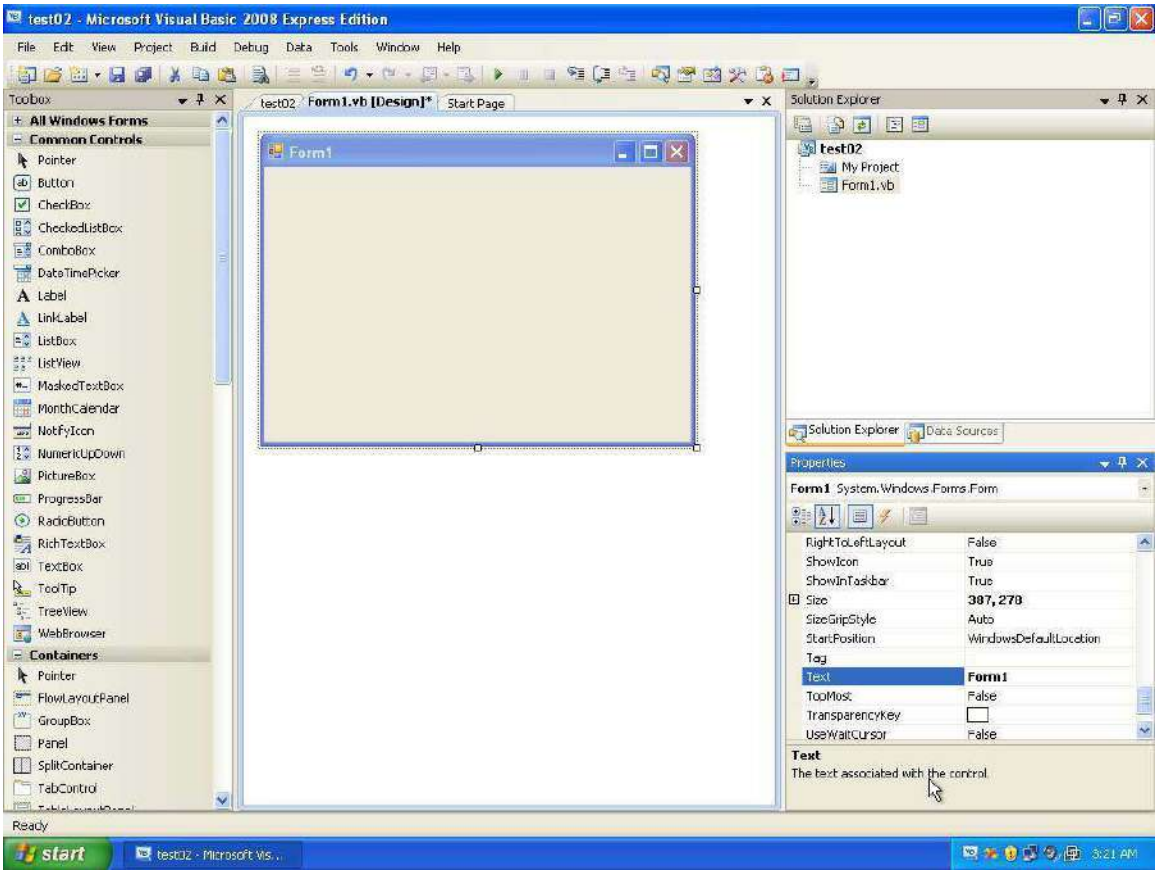
Now as I said before, the Solution explorer helps you navigate your application quickly. So now try double clicking My Project to see something similar to the above. Now you can modify your application. This is just an example, so don't worry about the details of all these options we will come to this later.



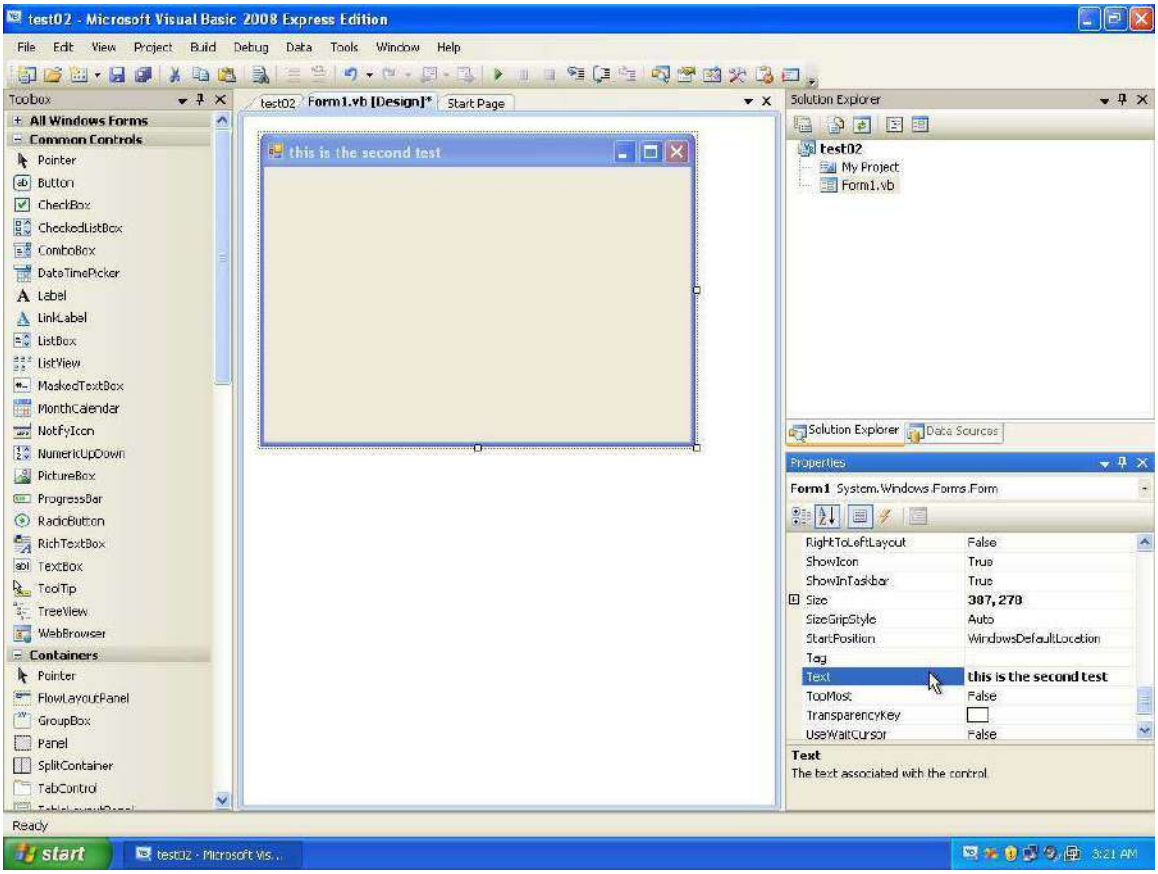
Now double click on Form1.vb you see the GUI again. Now let us work a little with these toys. Right now, our application has one window. We want to say, change its size. To do so, you drag one of the white boxes.



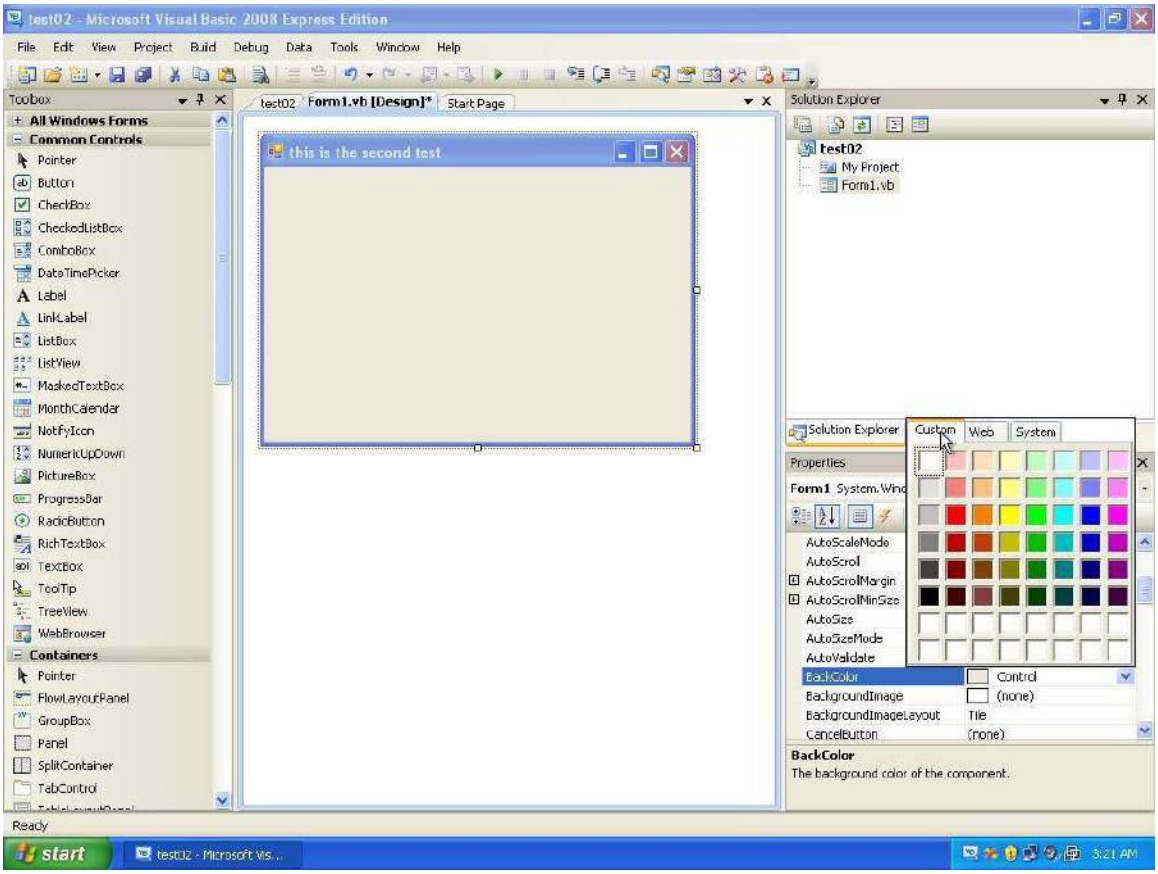
Notice also how does the properties window change to help you work on what you have just selected (here the window is selected so the properties window is showing its properties).



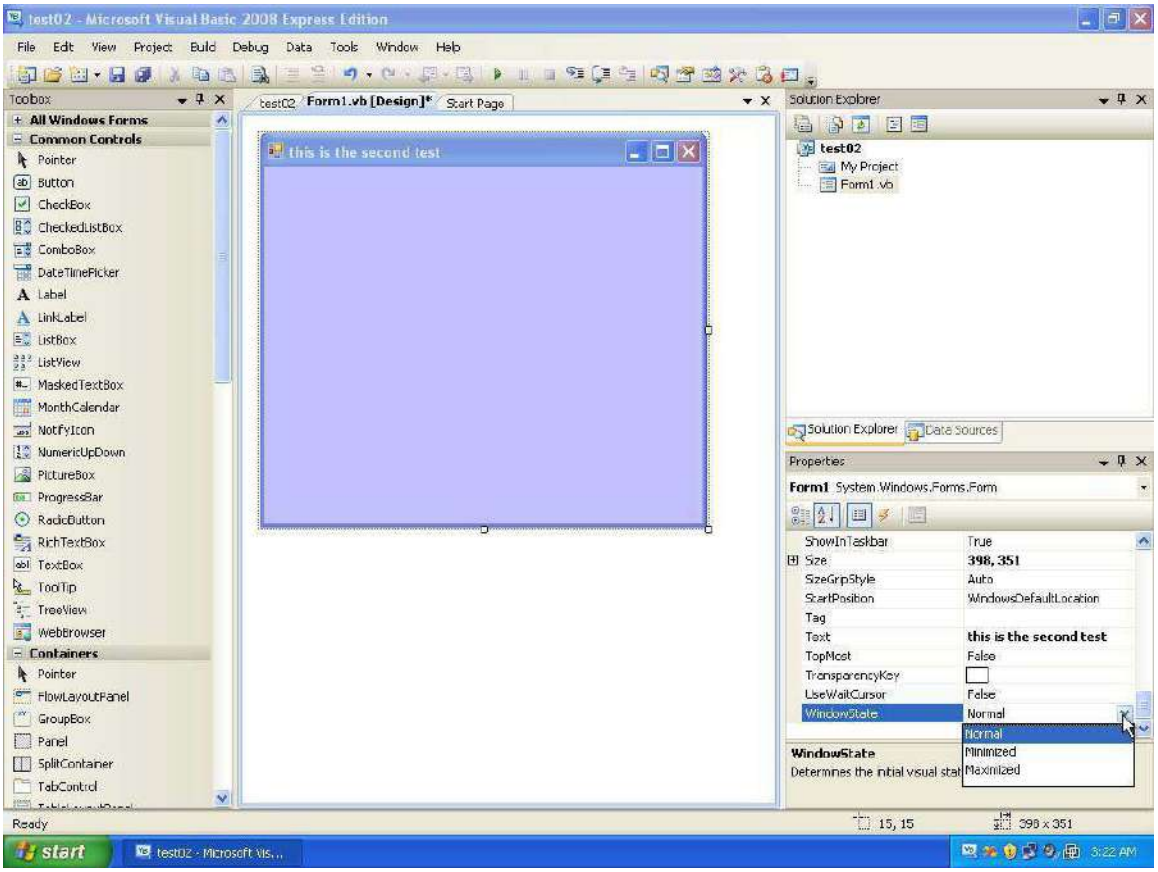
Now let us work with the properties window. Each object (window, or file or control) within the IDE has a number of properties that affects its behavior, and/or appearance. For example, if you search for a property called Text and change it, you can modify the title that appears on the window.



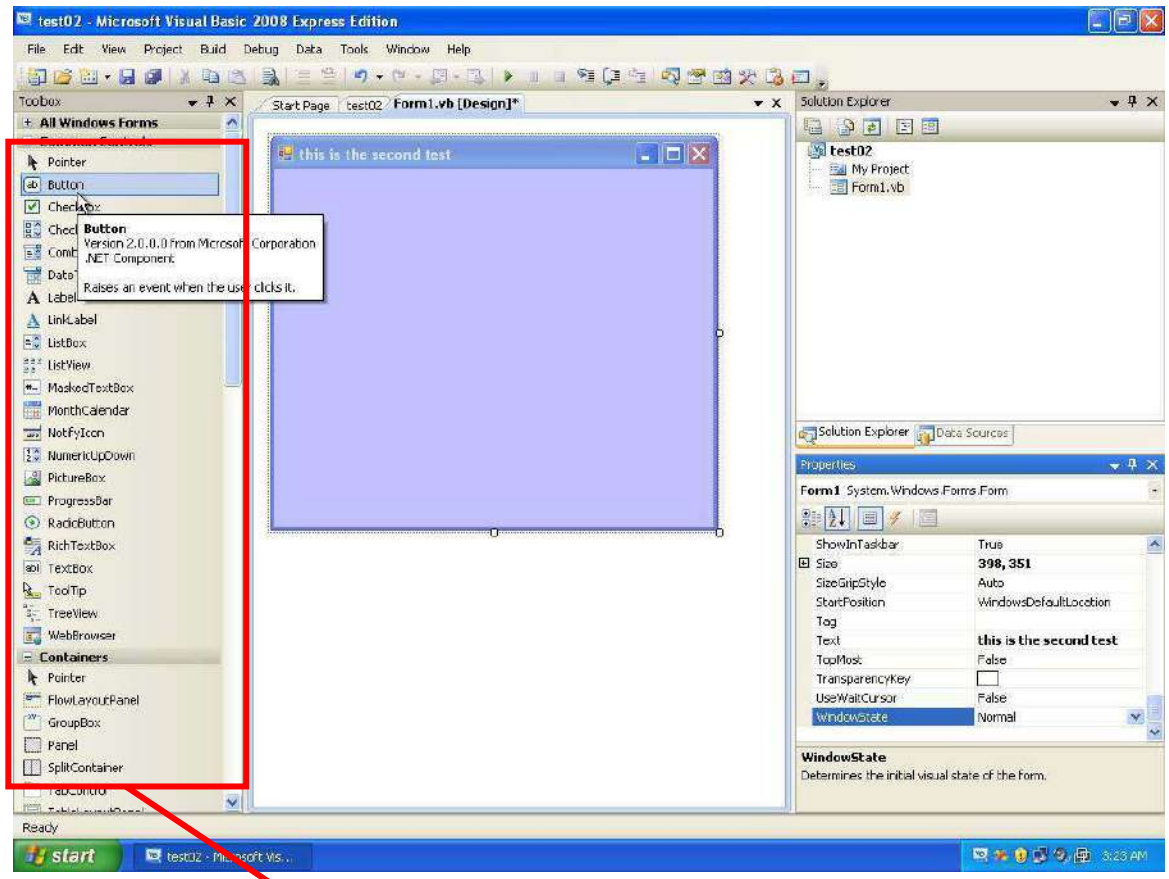
Try entering this is the second test and then press enter. You can see how does that affects the window you are designing.



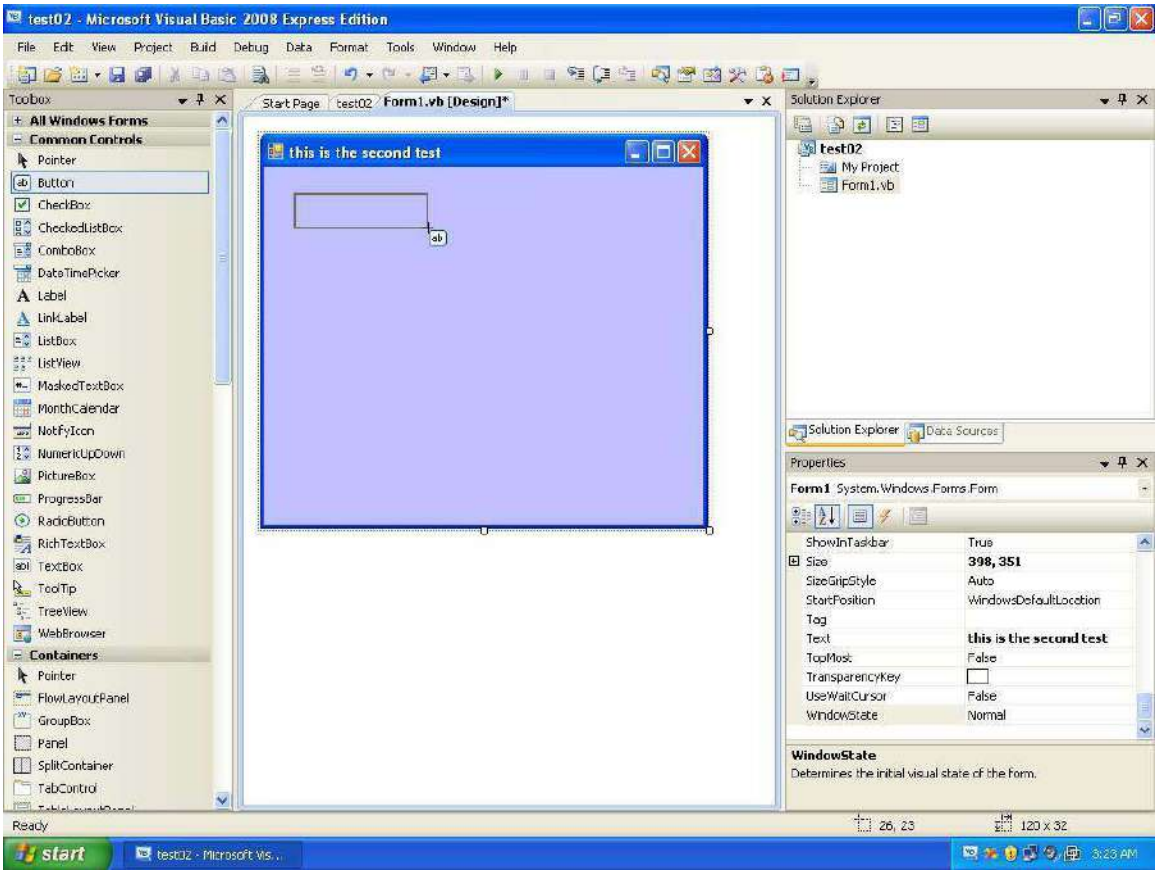
Now let us work with another property, which is BackColor, try to choose a color and see how does that affects the window's color.



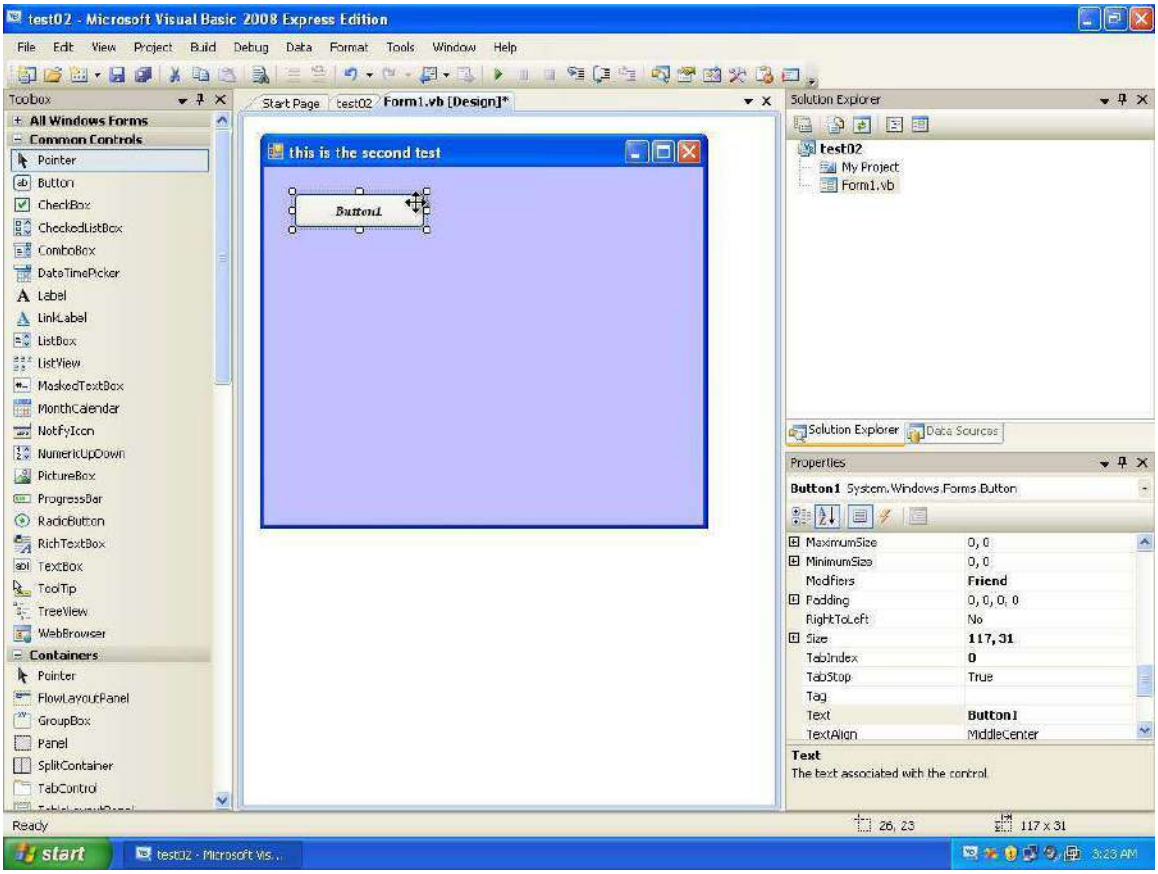
And so on, there are many other properties, we will learn just some of these that are commonly needed.



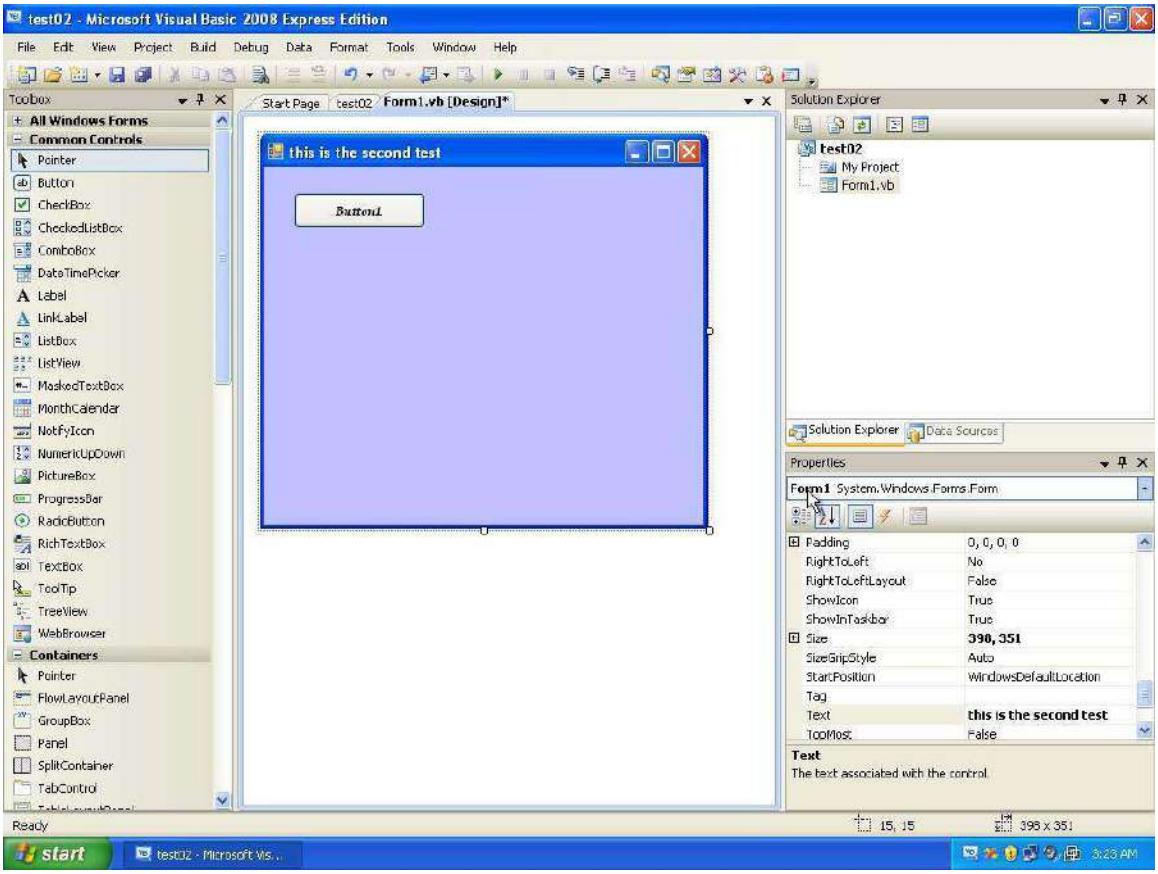
Now we come to the left side which we forget to tell you about. This one helps you adding controls to your window. For now click on the command button , and then draw it on the window (the command button is used to trigger some kind of actions or processing, more details on that later).



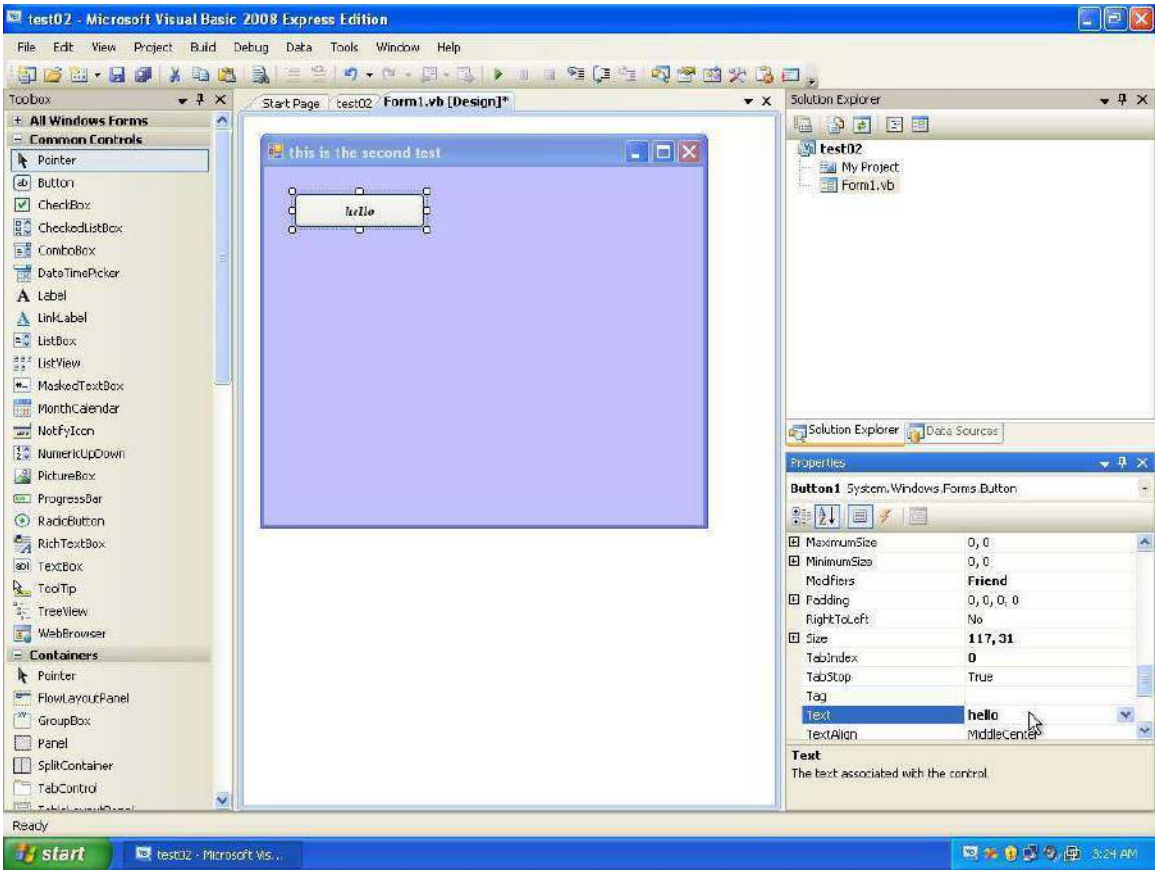
By pressing the mouse button continuously, you can specify the dimensions of that control. After that release the mouse button.



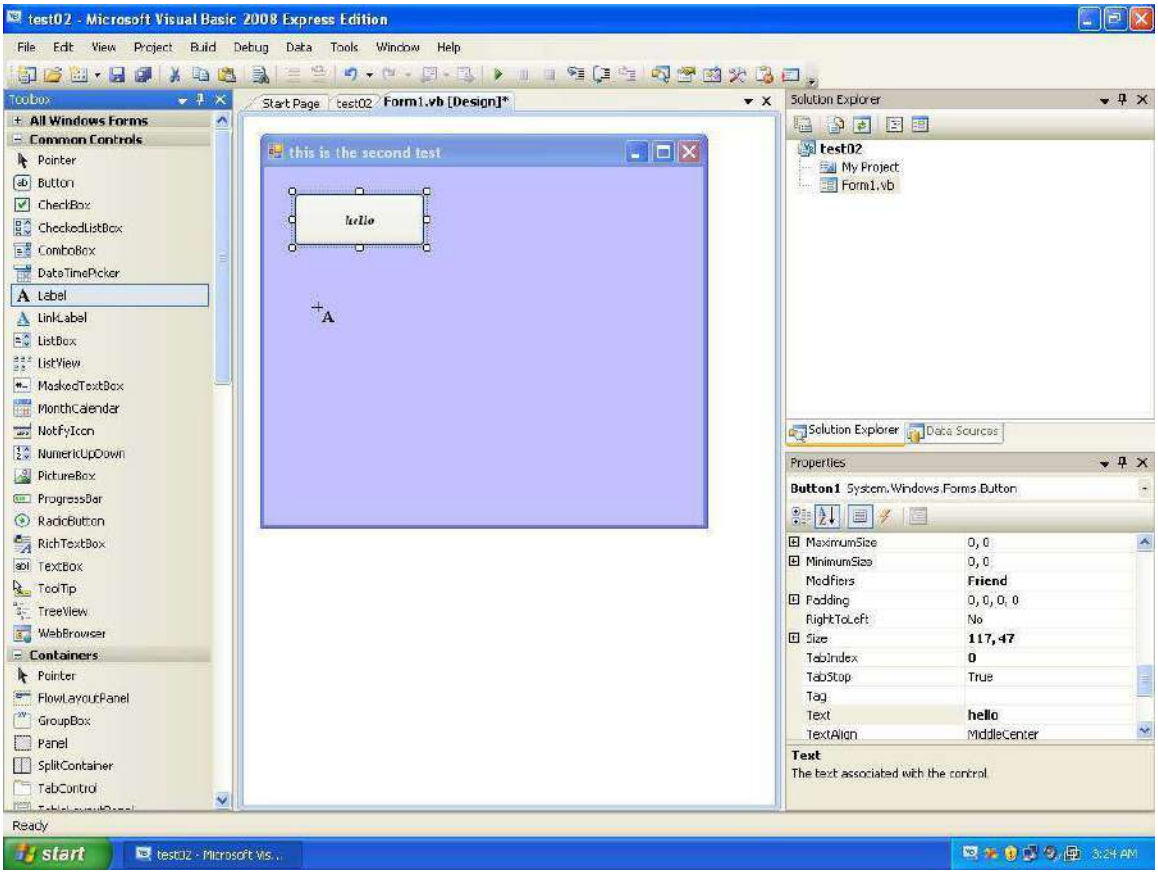
You can see how does your window look like. You can move the button by dragging it, or you can resize it using the white boxes. Also check out the properties window that shows you the properties of this button.



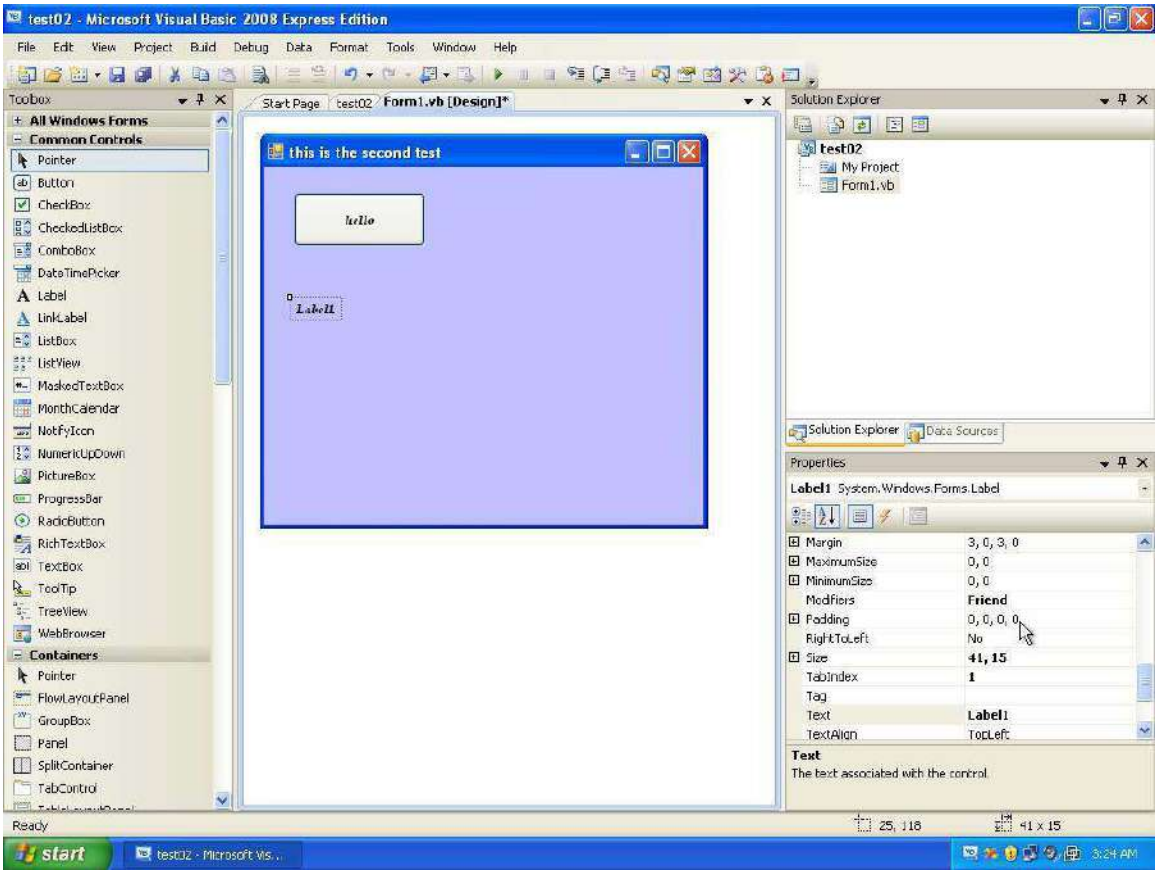
Now click on any empty space on your form to select it, and see how the properties window shows details about the selected object.

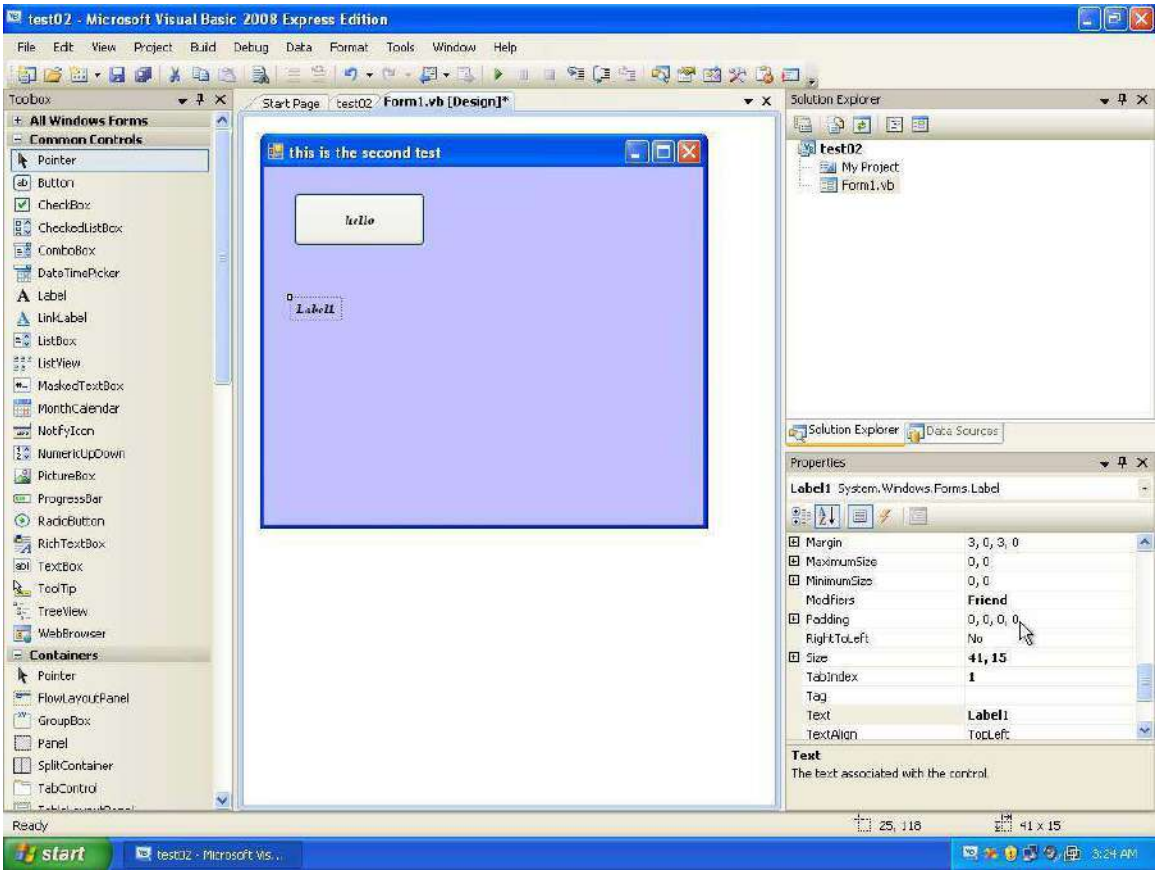


Now select the button again. You remember that when we wanted to change window title we modified the Text value for our window. The same is true for the button, and many other controls, so change the button's Text property to hello and see how does the GUI changes as well.

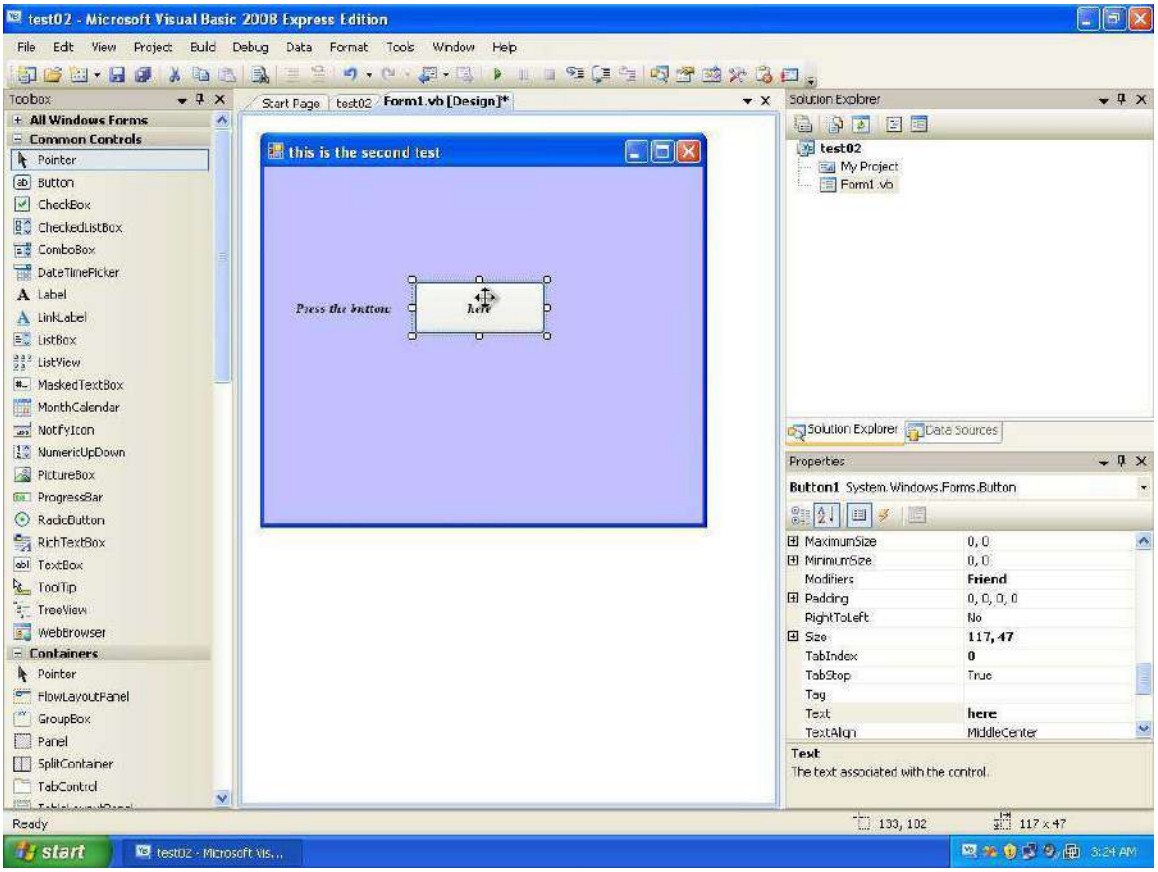


Next place a LABEL on the form. The Labels are used to display text information

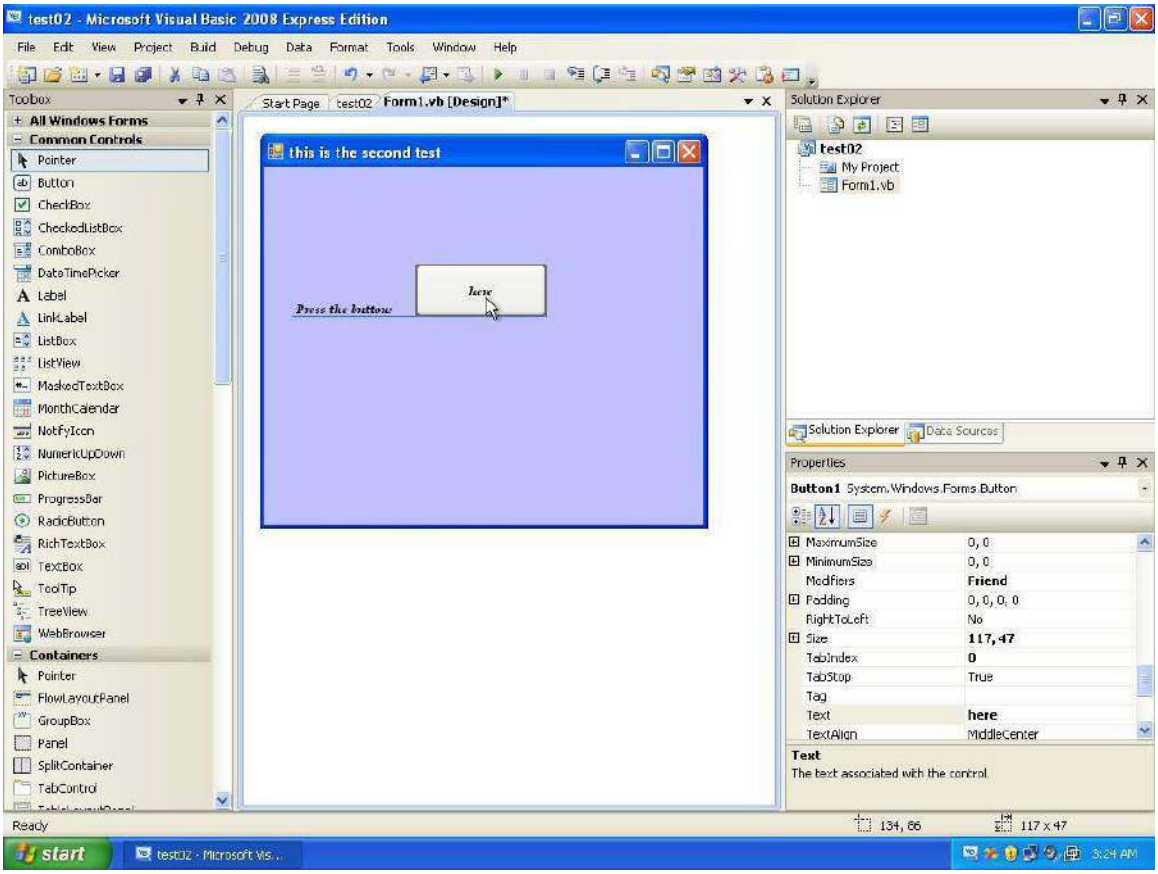

test02 - Microsoft Visual Basic 2008 Express Edition
File Edit View Project Build Debug Data Format Tools Window Help
Toolbox
All Windows Forms
Common Controls
Pointer
Button
CheckBox
CheckedListBox
ComboBox
DateTimePicker
Label
LinkLabel
ListBox
ListView
MaskedTextBox
MonthCalendar
NotifyIcon
NumericUpDown
PictureBox
ProgressBar
RadioButton
RichTextBox
TextBox
ToolTip
TreeView
WebBrowser
Containers
Pointer
FlowLayoutPanel
GroupBox
Panel
SplitContainer
TabControl
Start Page
test02
Form1.vb [Design]*
this is the second test
hello
Label1
Solution Explorer
test02
My Project
Form1.vb
Solution Explorer
Data Sources
Properties
Label1 System.Windows.Forms.Label
Margin 3, 0, 3, 0
MaximumSize 0, 0
MinimumSize 0, 0
Modifiers Friend
Padding 0, 0, 0, 0
RightToLeft No
Size 41, 15
TabIndex 1
Tag
Text Label1
TextAlign TopLeft
Text
The text associated with the control.
Ready
25, 118
41 x 15
start
test02 - Microsoft Vis...
3:24 AM

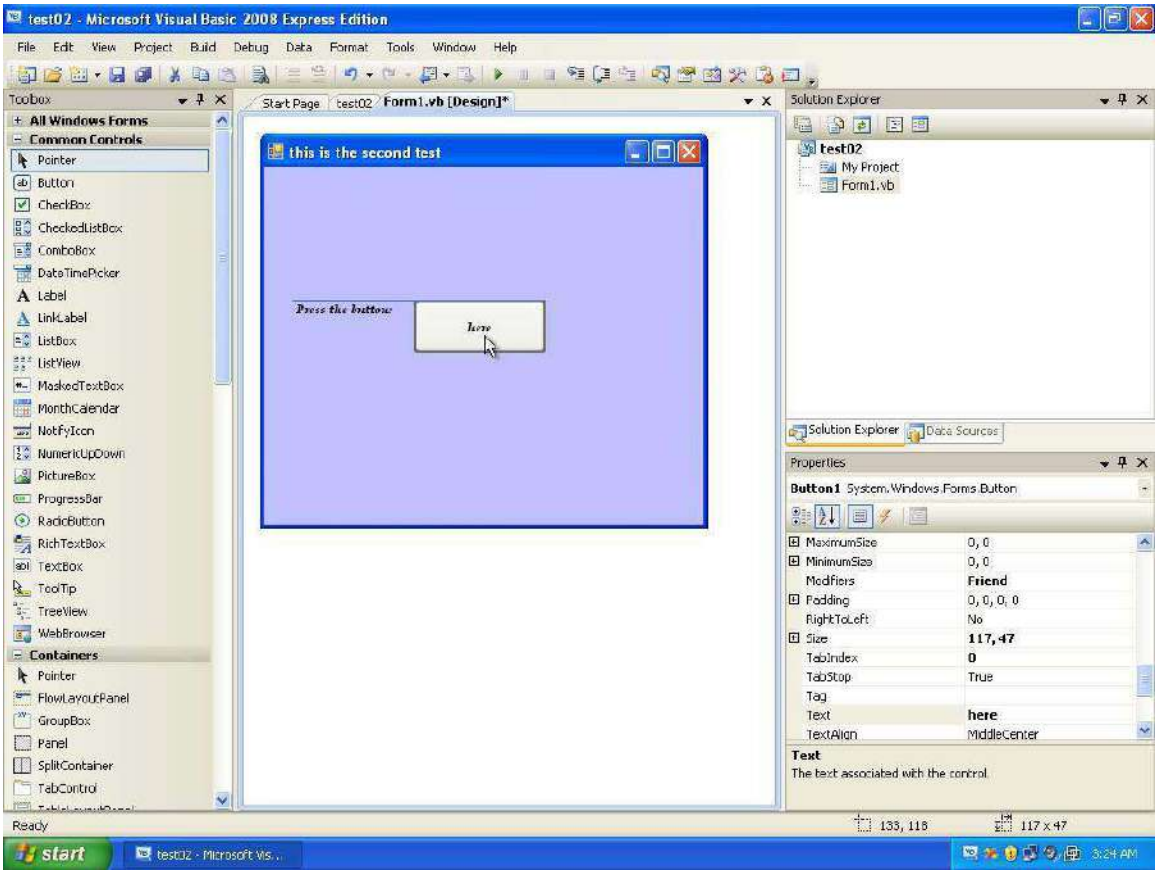


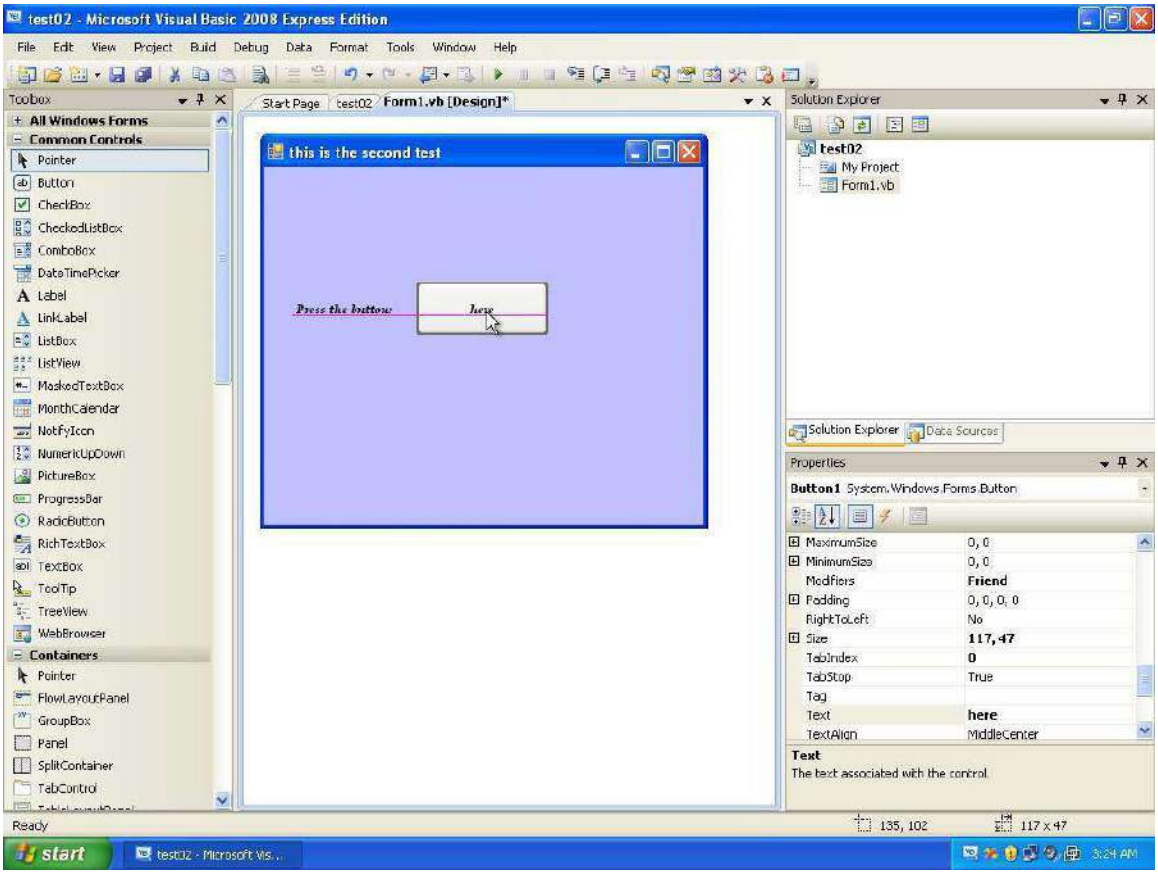
Now change the label to view the message: press the button

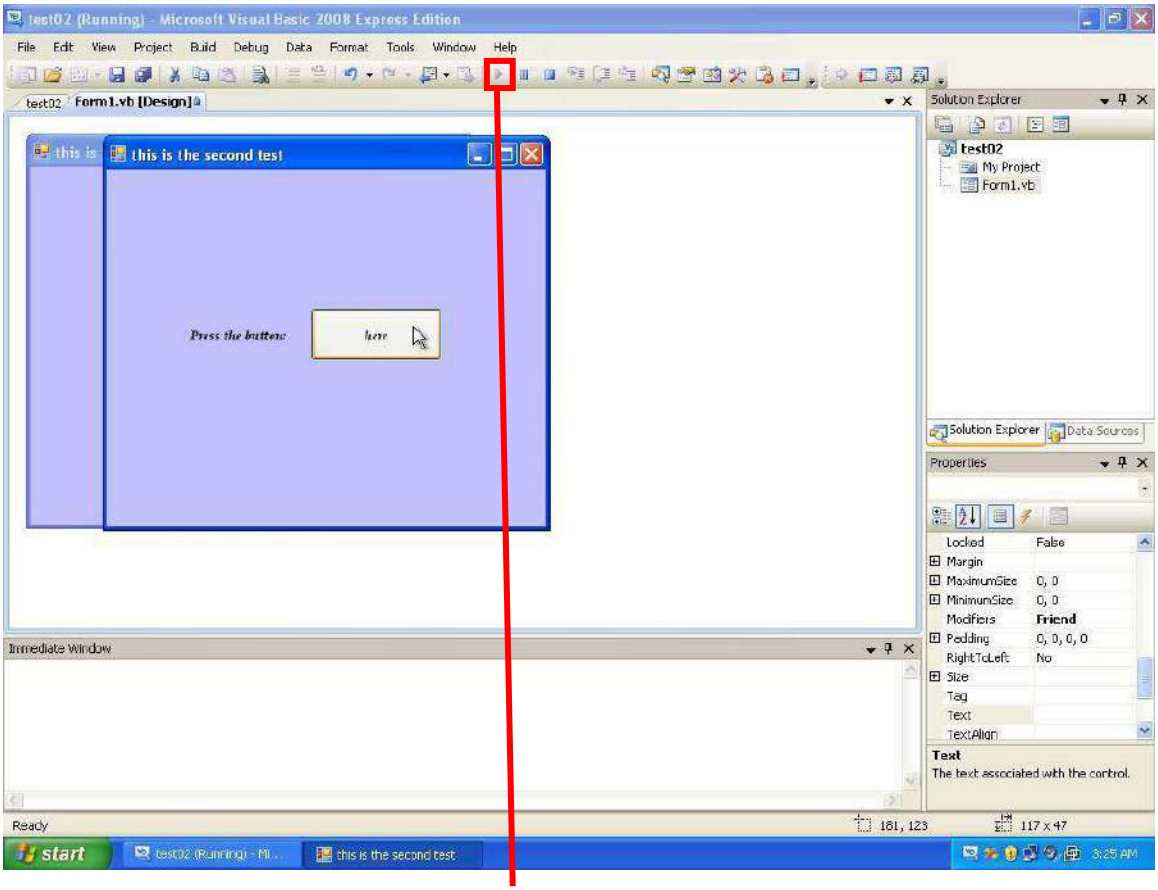


Now try to move the control and see how the IDE will help you place it relative to the label by showing you imaginary lines for placing the control on the form

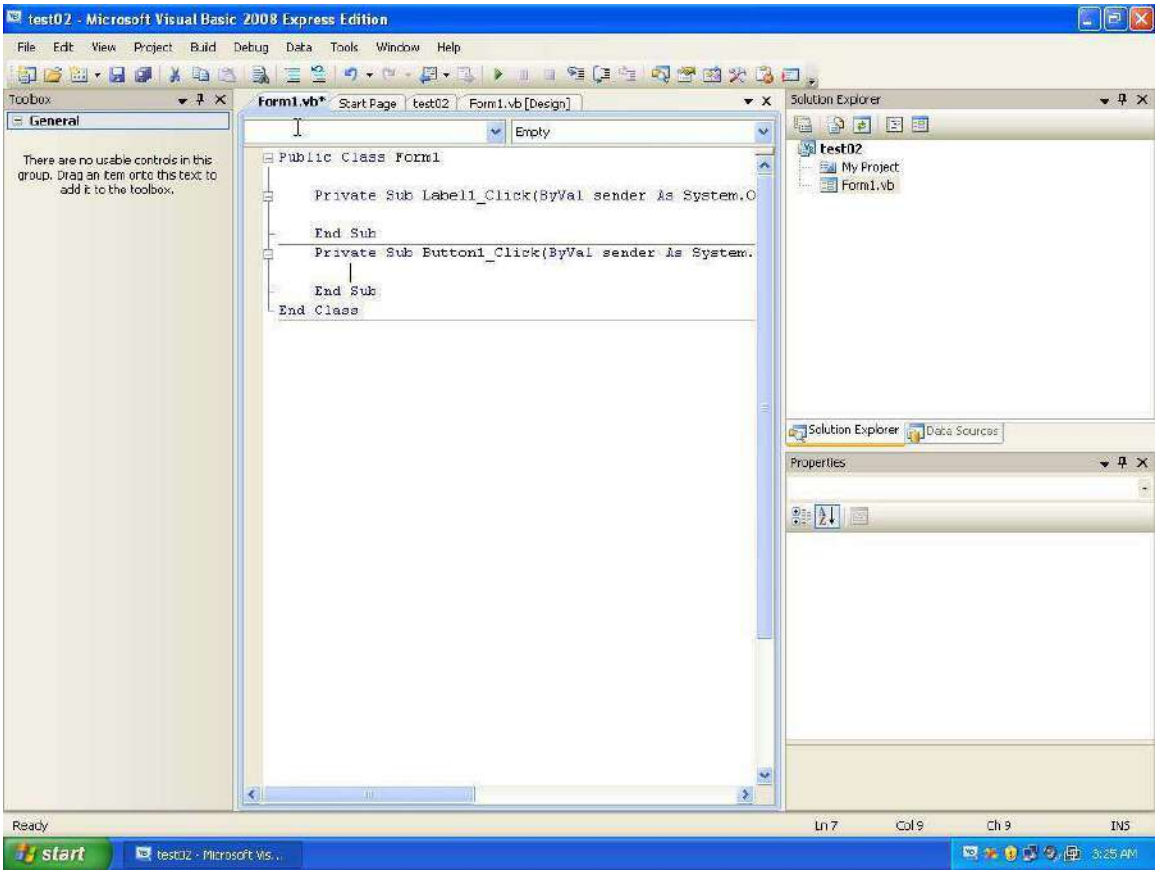

test02 - Microsoft Visual Basic 2008 Express Edition
File Edit View Project Build Debug Data Format Tools Window Help
Toolbox
All Windows Forms
Common Controls
Pointer
Button
CheckBox
CheckedListBox
ComboBox
DateTimePicker
Label
LinkLabel
ListBox
ListView
MaskedTextBox
MonthCalendar
NotifyIcon
NumericUpDown
PictureBox
ProgressBar
RadioButton
RichTextBox
TextBox
ToolTip
TreeView
WebBrowser
Containers
Pointer
FlowLayoutPanel
GroupBox
Panel
SplitContainer
TabControl
Start Page
test02
Form1.vb [Design]*
this is the second test
here
Press the button
Solution Explorer
test02
My Project
Form1.vb
Solution Explorer
Data Sources
Properties
Button1 System.Windows.Forms.Button
MaximumSize 0, 0
MinimumSize 0, 0
Modifiers Friend
Padding 0, 0, 0, 0
RightToLeft No
Size 117, 47
TabIndex 0
TabStop True
Tag
Text here
TextAlign MiddleCenter
Text
The text associated with the control.
Ready
134, 86
117 x 47
start
test02 - Microsoft Vis...
3:24 AM


test02 - Microsoft Visual Basic 2008 Express Edition
File Edit View Project Build Debug Data Format Tools Window Help
Toolbox
All Windows Forms
Common Controls
Pointer
Button
CheckBox
CheckedListBox
ComboBox
DateTimePicker
Label
LinkLabel
ListBox
ListView
MaskedTextBox
MonthCalendar
NotifyIcon
NumericUpDown
PictureBox
ProgressBar
RadioButton
RichTextBox
TextBox
ToolTip
TreeView
WebBrowser
Containers
Pointer
FlowLayoutPanel
GroupBox
Panel
SplitContainer
TabControl
Start Page
test02
Form1.vb [Design]*
this is the second test
Press the button
here
Solution Explorer
test02
My Project
Form1.vb
Solution Explorer
Data Sources
Properties
Button1 System.Windows.Forms.Button
MaximumSize 0, 0
MinimumSize 0, 0
Modifiers Friend
Padding 0, 0, 0, 0
RightToLeft No
Size 117, 47
TabIndex 0
TabStop True
Tag
Text here
TextAlign MiddleCenter
Text
The text associated with the control.
Ready
133, 118
117 x 47
start
test02 - Microsoft Vis...
3:24 AM

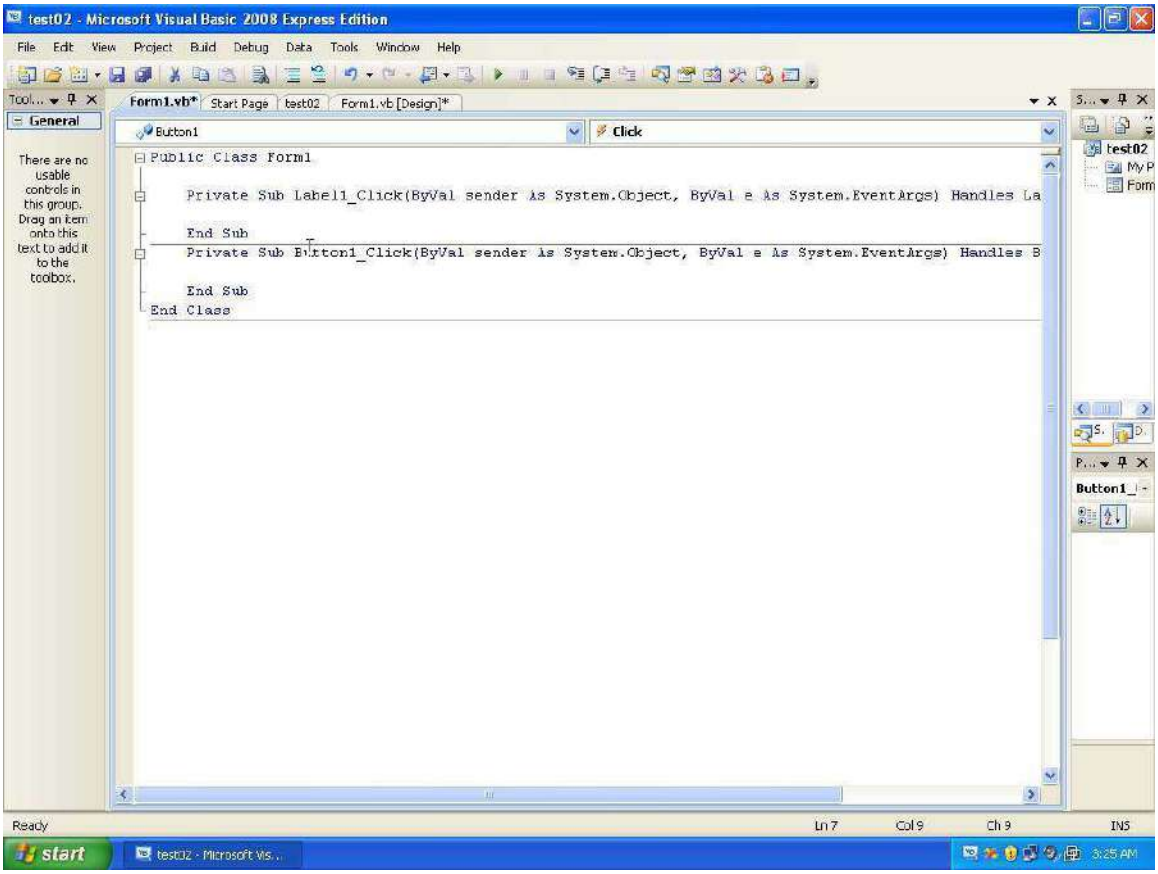

test02 - Microsoft Visual Basic 2008 Express Edition
File Edit View Project Build Debug Data Format Tools Window Help
Toolbox
All Windows Forms
Common Controls
Pointer
Button
CheckBox
CheckedListBox
ComboBox
DateTimePicker
Label
LinkLabel
ListBox
ListView
MaskedTextBox
MonthCalendar
NotifyIcon
NumericUpDown
PictureBox
ProgressBar
RadioButton
RichTextBox
TextBox
ToolTip
TreeView
WebBrowser
Containers
Pointer
FlowLayoutPanel
GroupBox
Panel
SplitContainer
TabControl
Start Page
test02
Form1.vb [Design]*
this is the second test
Press the button
here
Solution Explorer
test02
My Project
Form1.vb
Solution Explorer
Data Sources
Properties
Button1 System.Windows.Forms.Button
MaximumSize 0, 0
MinimumSize 0, 0
Modifiers Friend
Padding 0, 0, 0, 0
RightToLeft No
Size 117, 47
TabIndex 0
TabStop True
Tag
Text here
TextAlign MiddleCenter
Text
The text associated with the control.
Ready
135, 102
117 x 47
start
test02 - Microsoft Vis...
3:24 AM



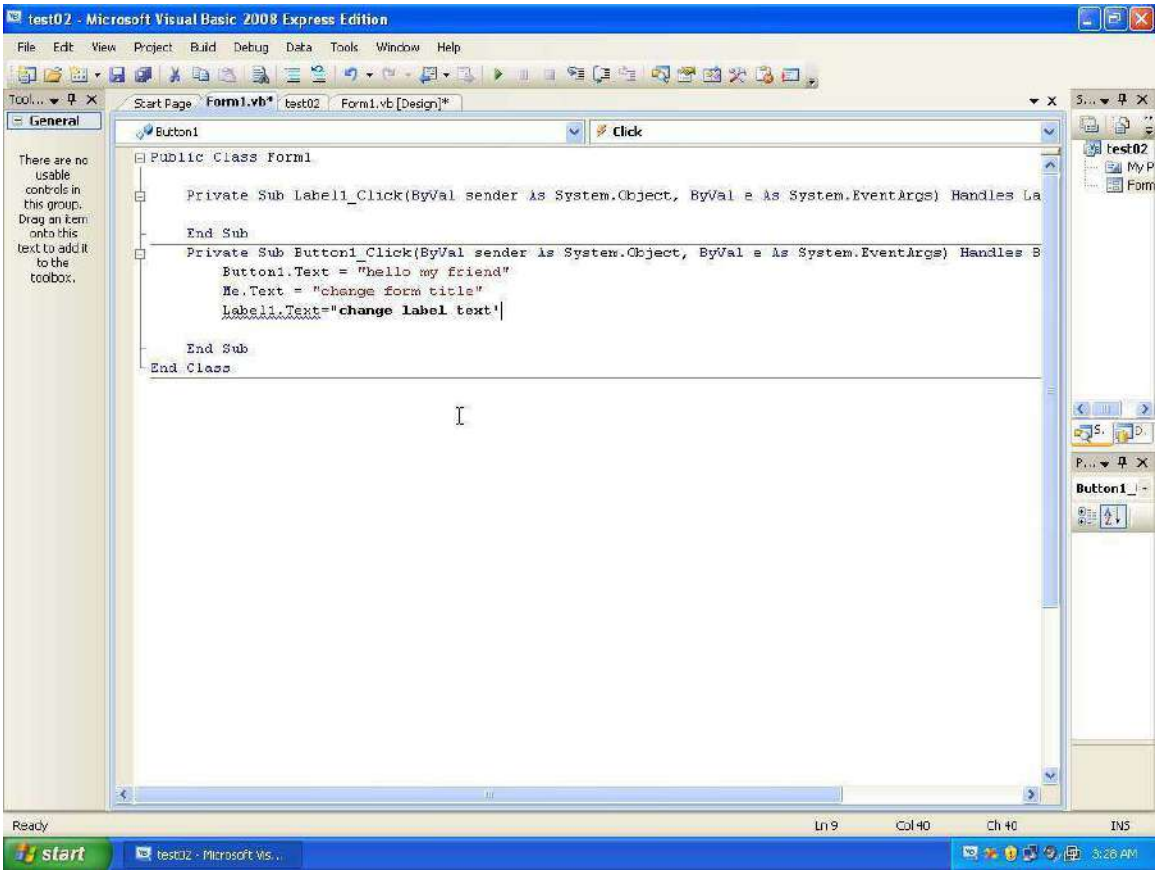
Now run the application by pressing the Play button. Now pressing the button does not do anything at all. That is because you haven't tell the computer what to do when you press the button.



Now stop the running application by closing the window, then double click on the button, you should see something like this.



This is the code editor which helps you telling application what should it does in a specific event. For our example telling it what should it does when you press the button.



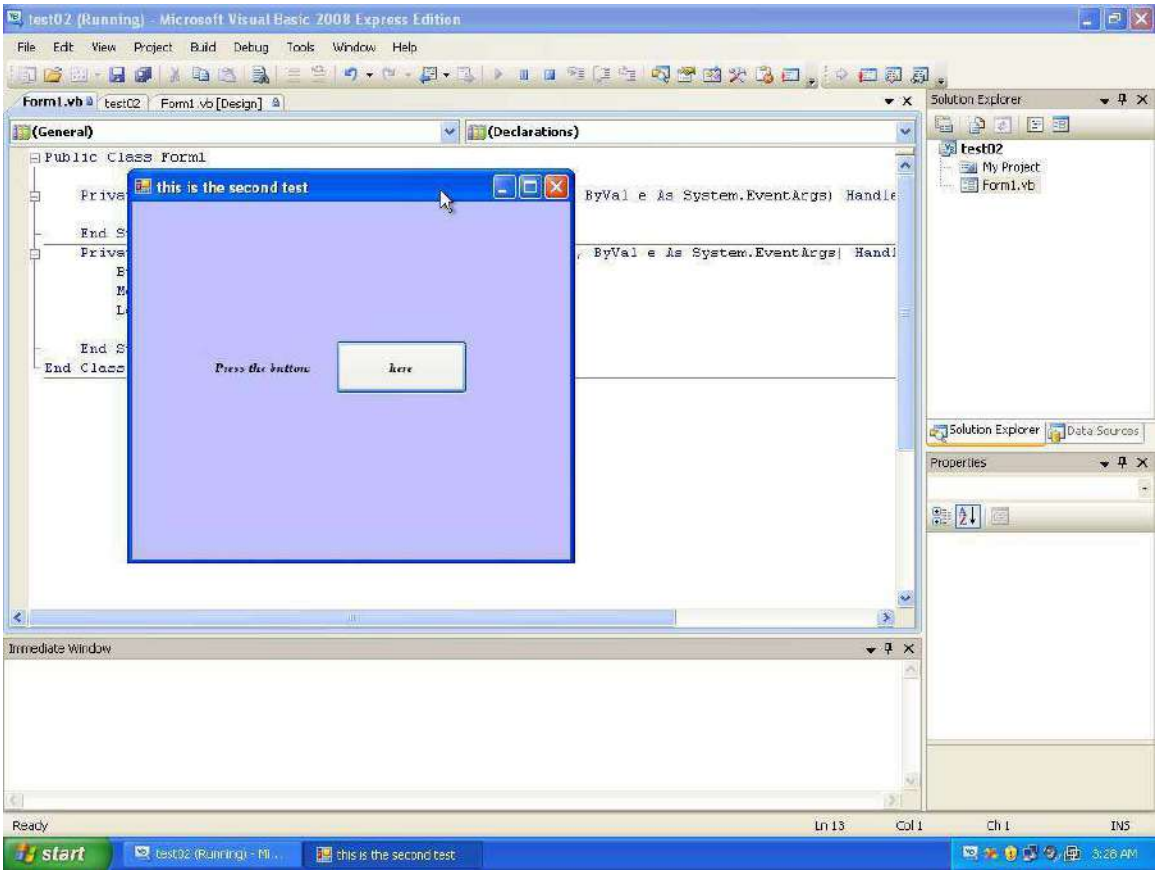
Write the lines above exactly as you see, the code means the following:

`Button1.Text = "hello my friend"` means that go to the button on the form those name is Button1, and modify its Text property to hello my friend. (more on that in the next tutorial).

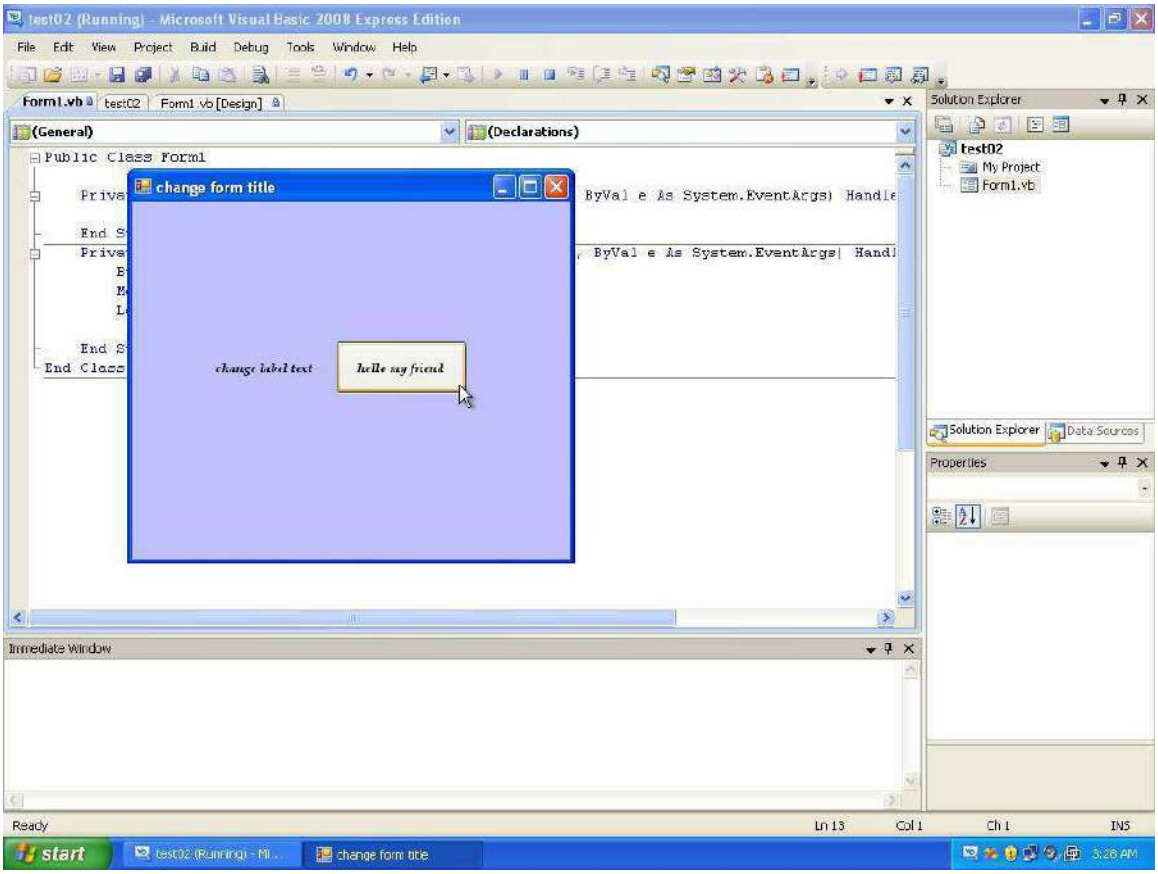
`Me.Text ="change form title"` changes the title of the form by changing its Text property, and the same for the last line which changes it for the label.

It is important to understand that the Text property here is the same one that you changed in the properties window. The properties window changes the properties while you are designing the window, hence the first change you did is a design time change. However the code you just added will not be executed until you press the button while the application is running (i.e. run time), so such changes are not visible (yet).

Now press F5 to see the application running

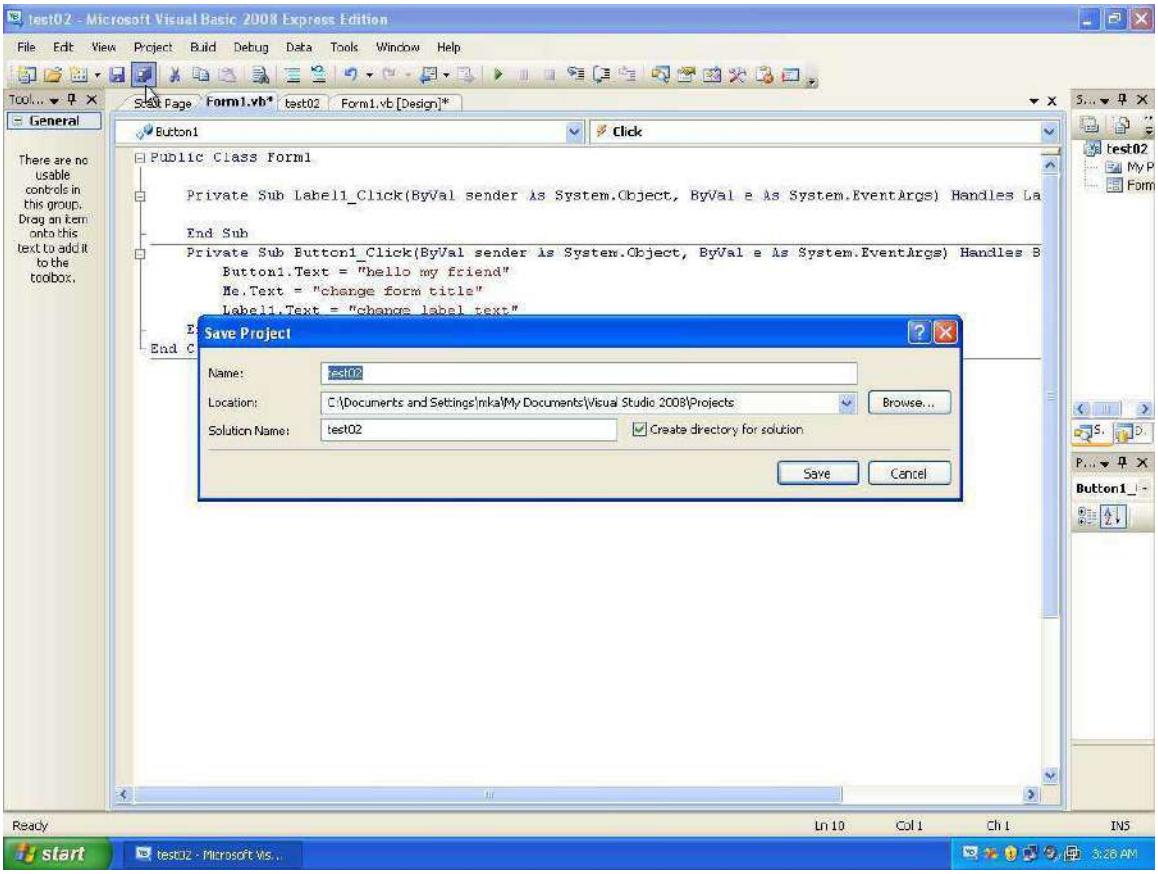


Now press the button and see what happens



Don't worry too much about the code. The idea here is to know that there are some properties that can be modified later on, and make you familiar with the GUI.

The next tutorial will explain about controls, their names, and their events. So for now you may save your project by pressing save all



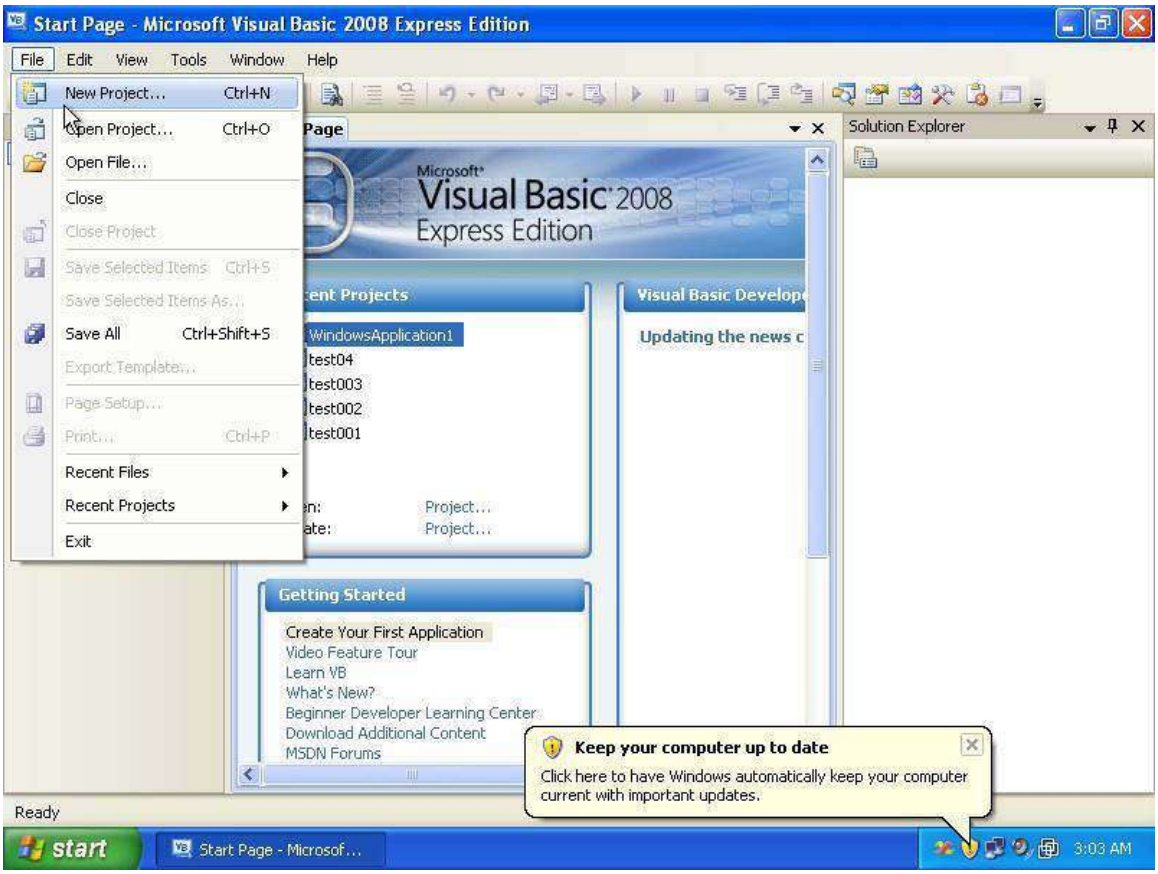
This concludes this chapter.

Chapter 3: Understanding Buttons, and Textboxes

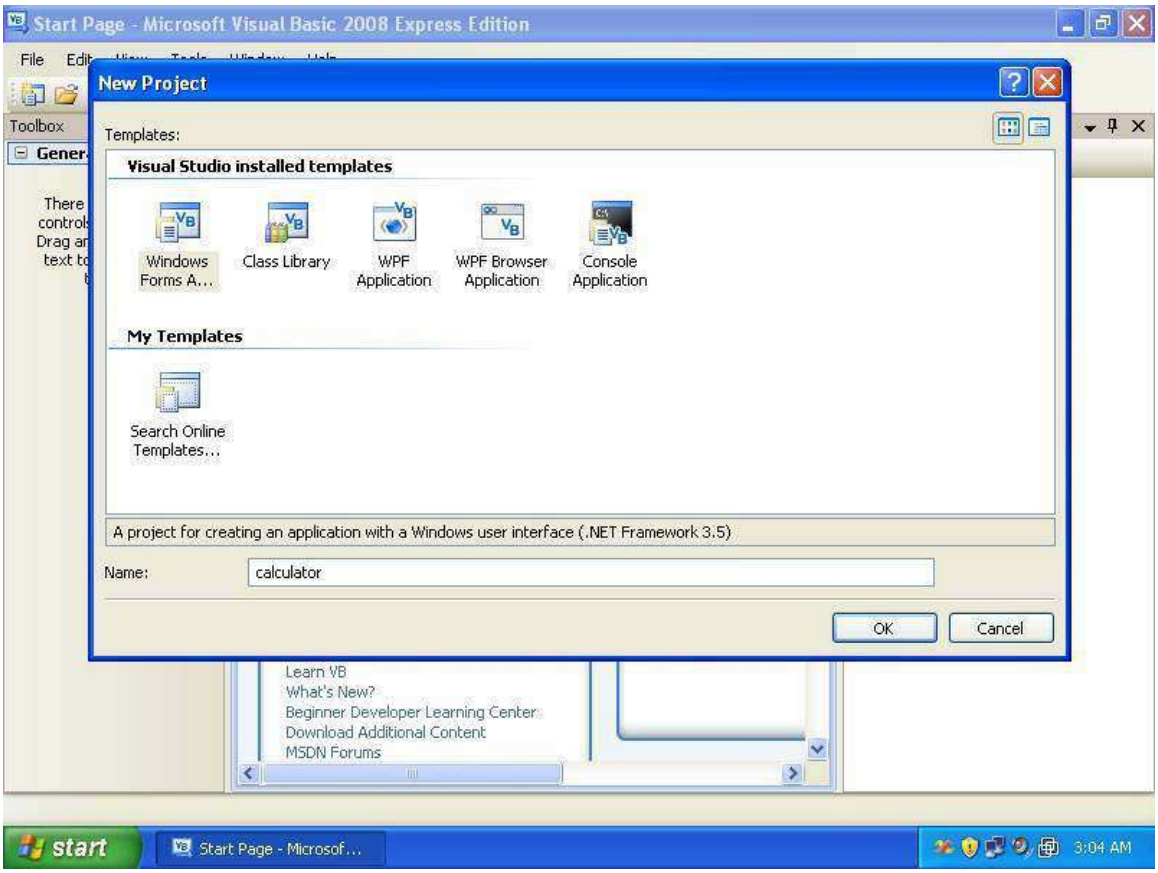
Understanding Buttons, and Textboxes

This chapter deals with buttons and textboxes. We are going to see how to develop a simple calculator application using VB.NET, and examine the controls and their properties.

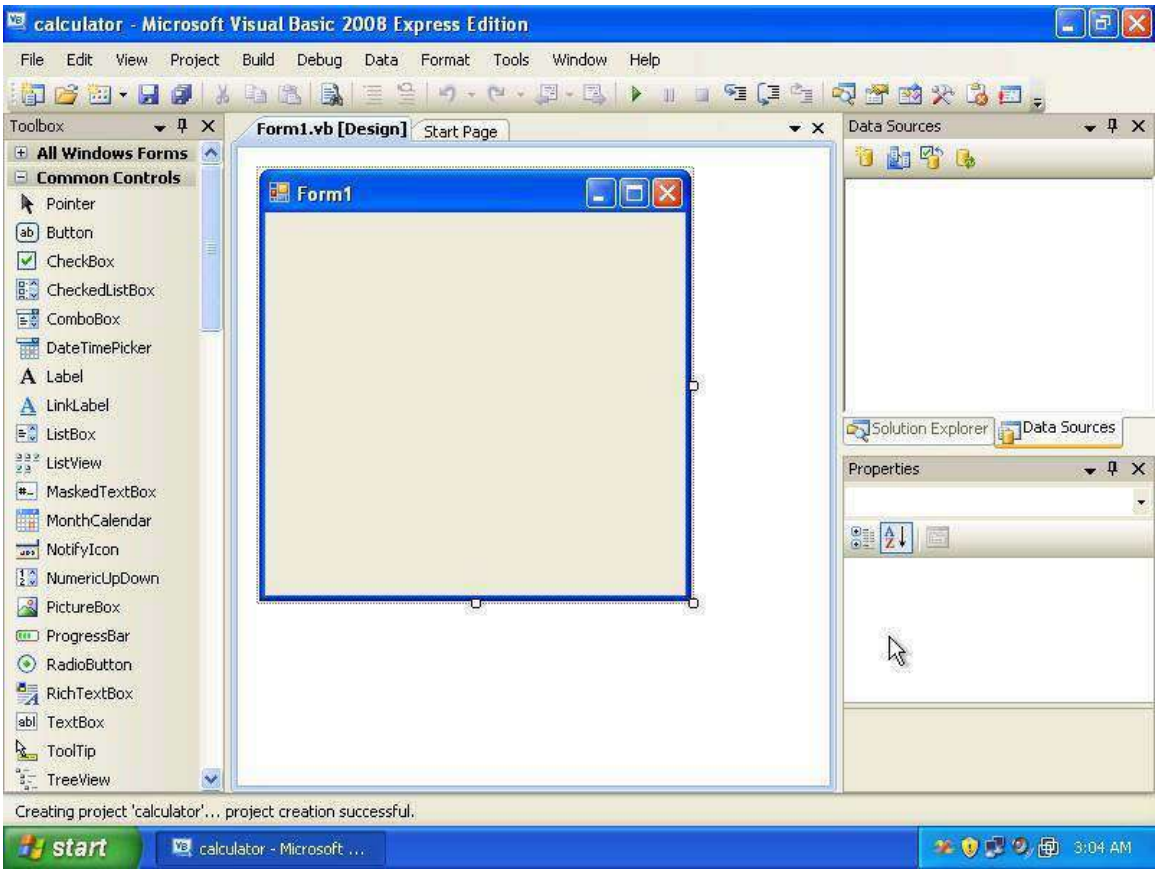
First open VB.NET



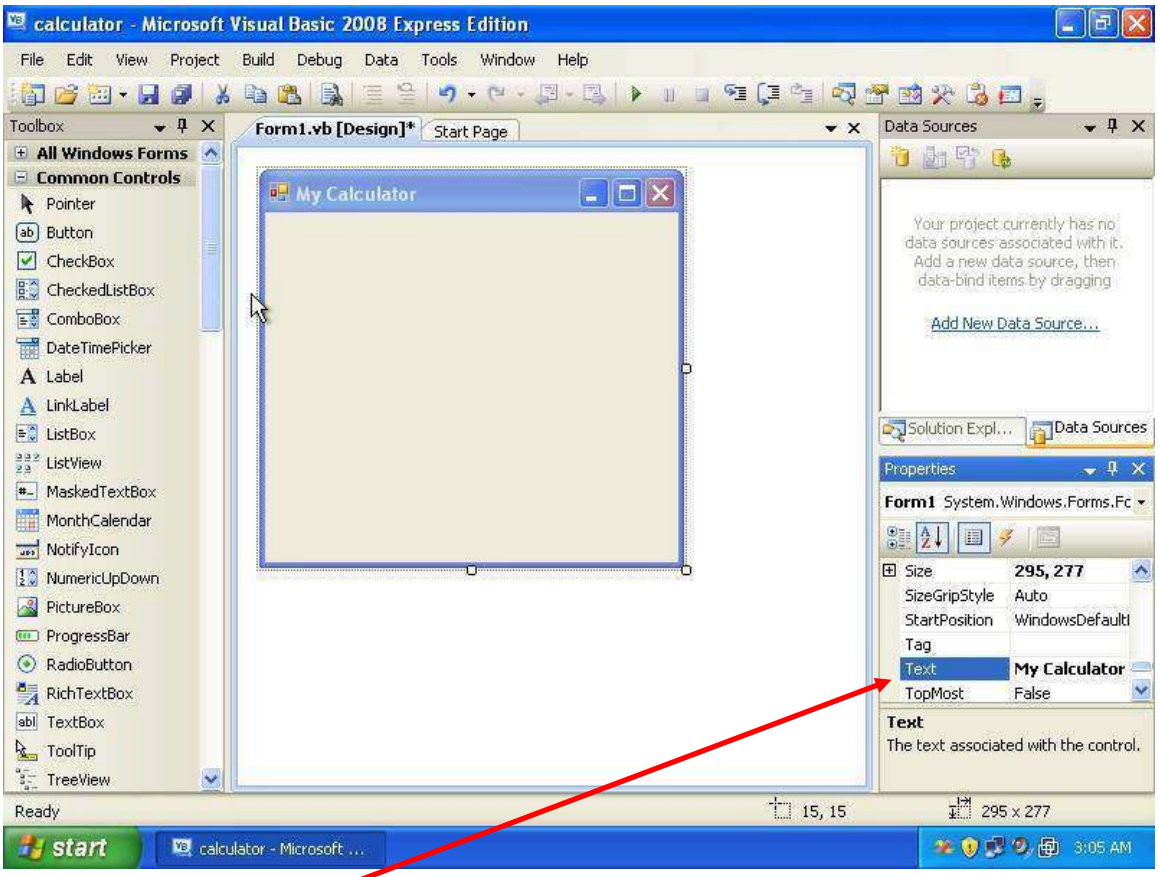
Create a new project



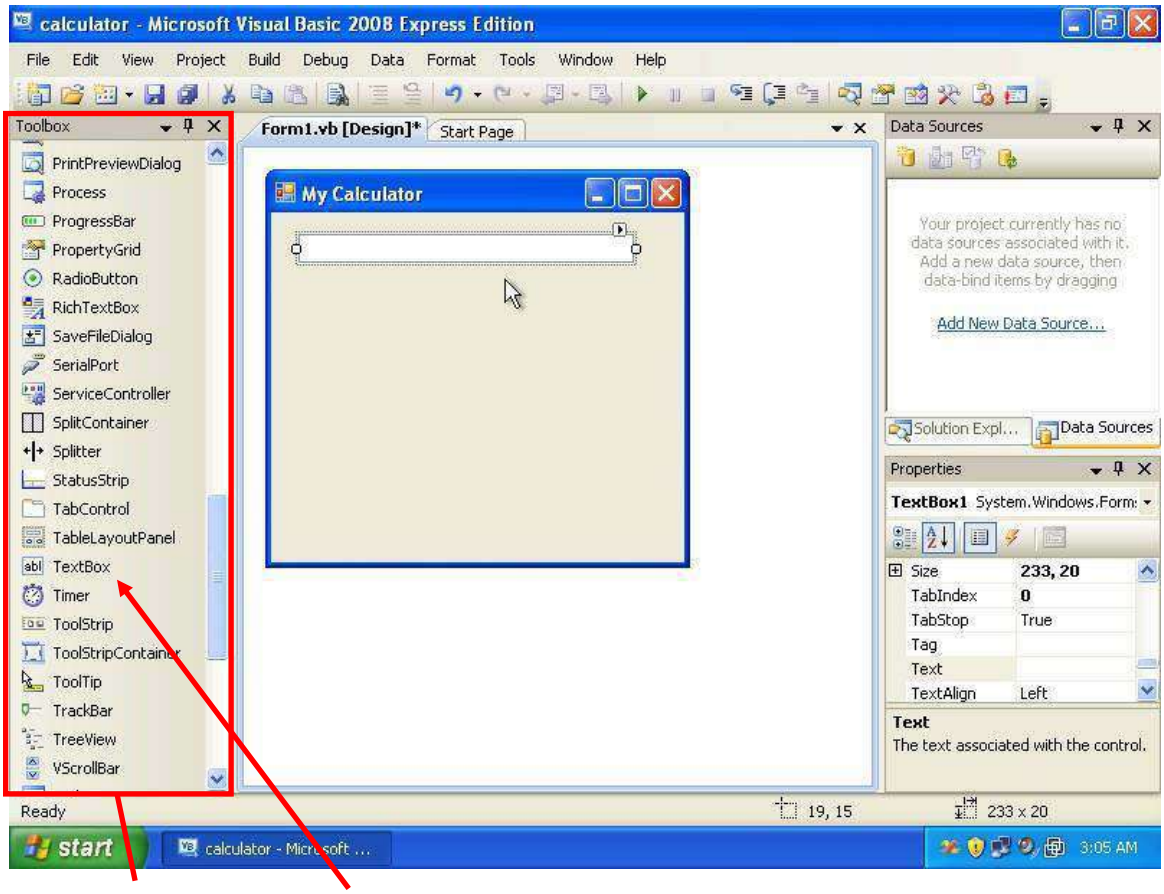
Set its type to **windows forms application** and set its name to **calculator**. Press **OK**



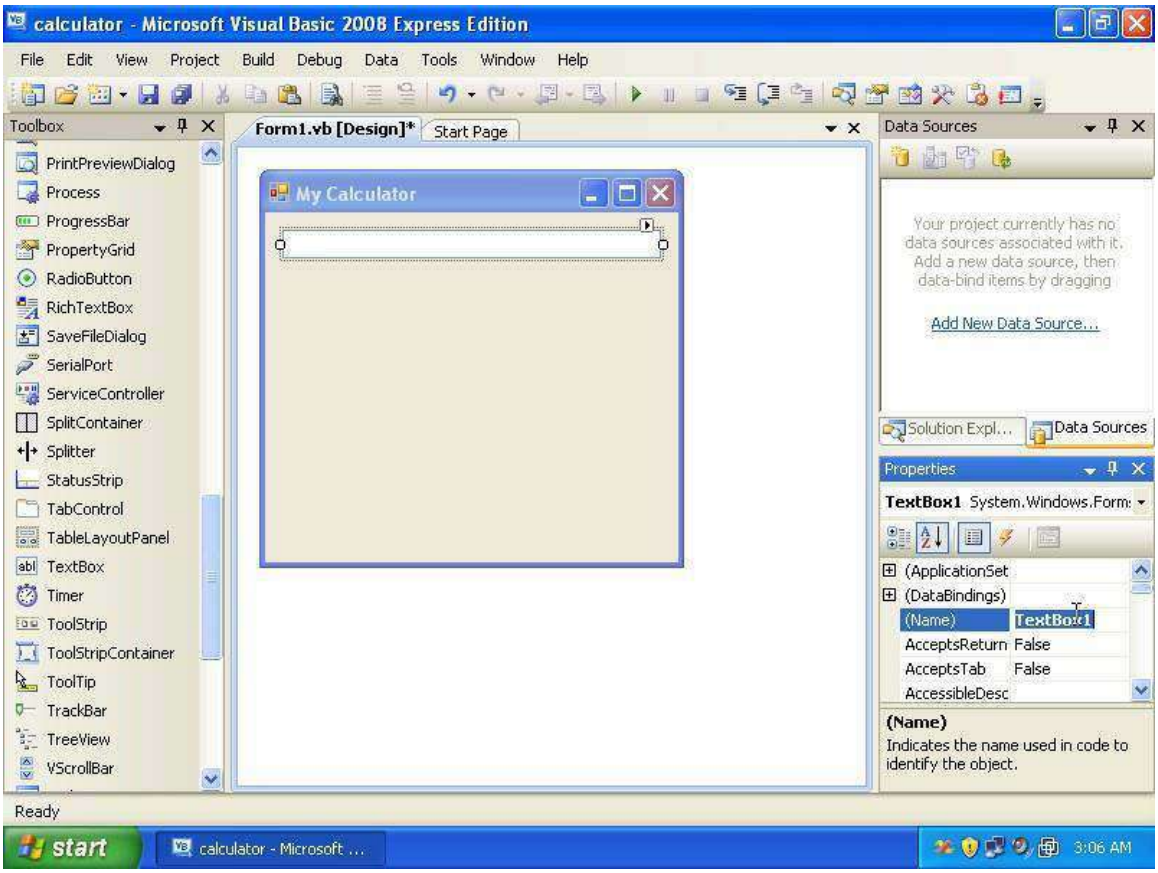
You should see the main form on the workspace



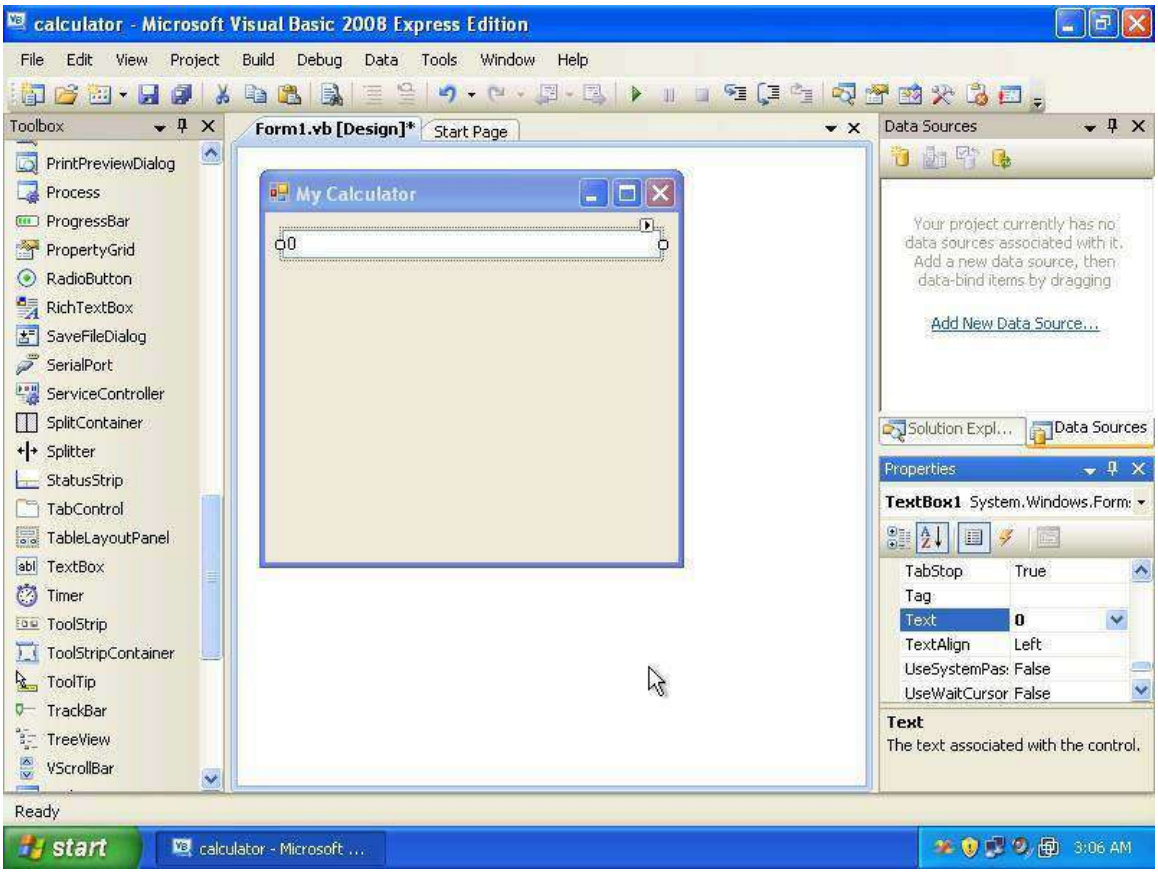
Change the form text property to **My Calculator**, you don't want your application to have the title **Form1** when it starts.



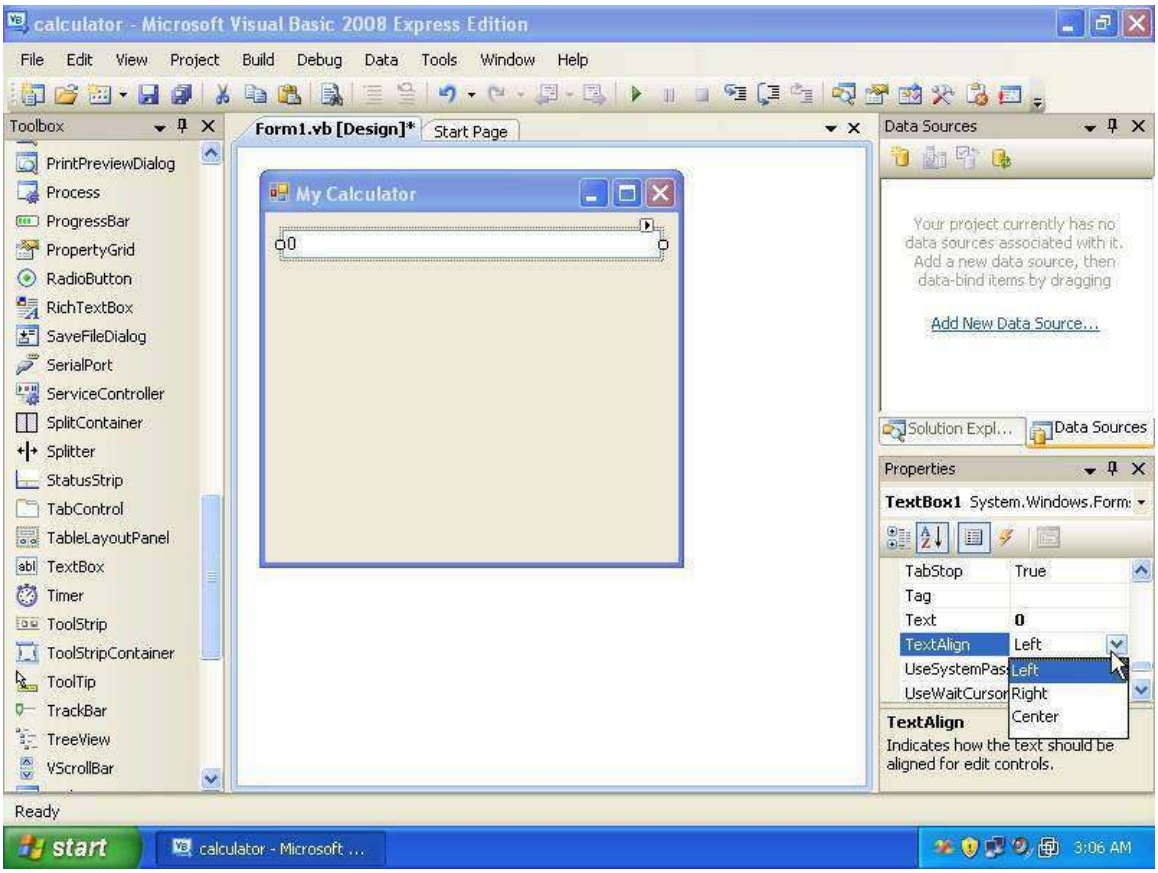
From the **ToolBox**, drop a **TextBox** control onto the form, and resize it.



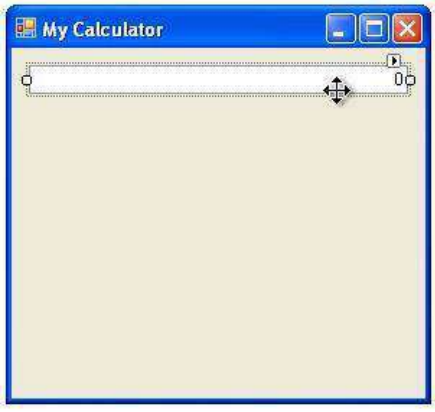
After setting the text box location, we will change its **Name** property. This property allows us to modify and access this control at runtime and change its behavior, or perform something for us. So, set it to **LCD** since it will be used to display the numbers. You can leave the name **TextBox1** without change, or select any other name. Keep in mind that the name you choose will be used instead of **LCD** in the code. Note that the **Name** property does not have any visible effect, you can't see the result of its change, because it will be used internally only to reference this control.



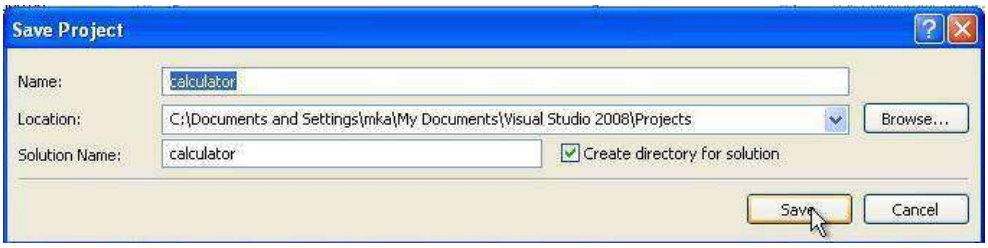
Now, we start working with the display, usually when the calculator starts it should display the number 0. In out case it does not. So we modify the Text Property and write 0 in that. Make sure you don't add any spaces before or after the 0.



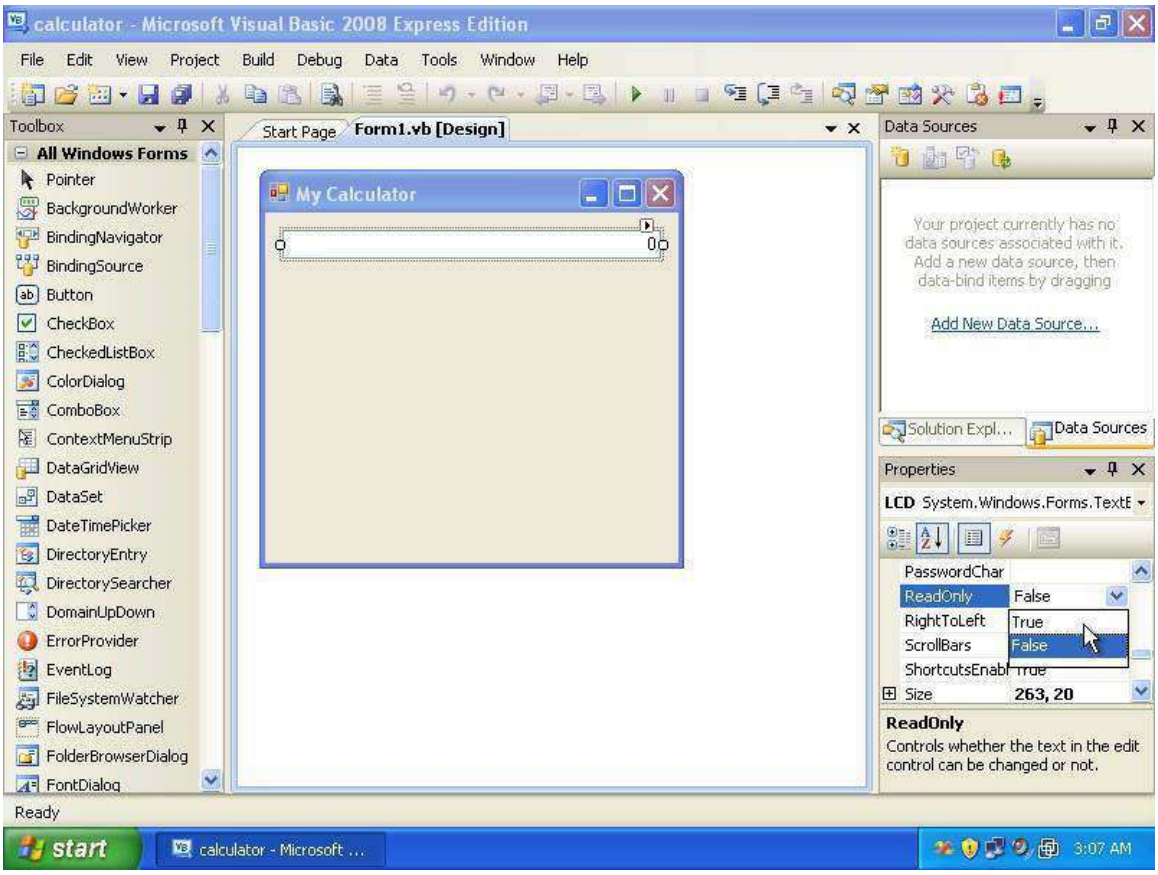
The second thing to note is that the number is aligned to the left, while calculators align the number to the right. Search for the **TextAlign** property and change it to Right.



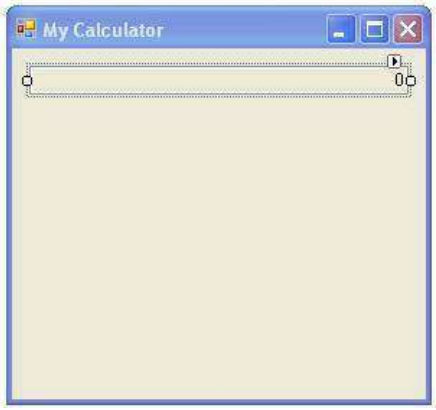
This what the window will look like.



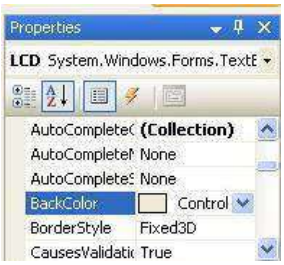
By the way, we forgot to save the project, so, press save all. It is good practice to save your work every few minutes.



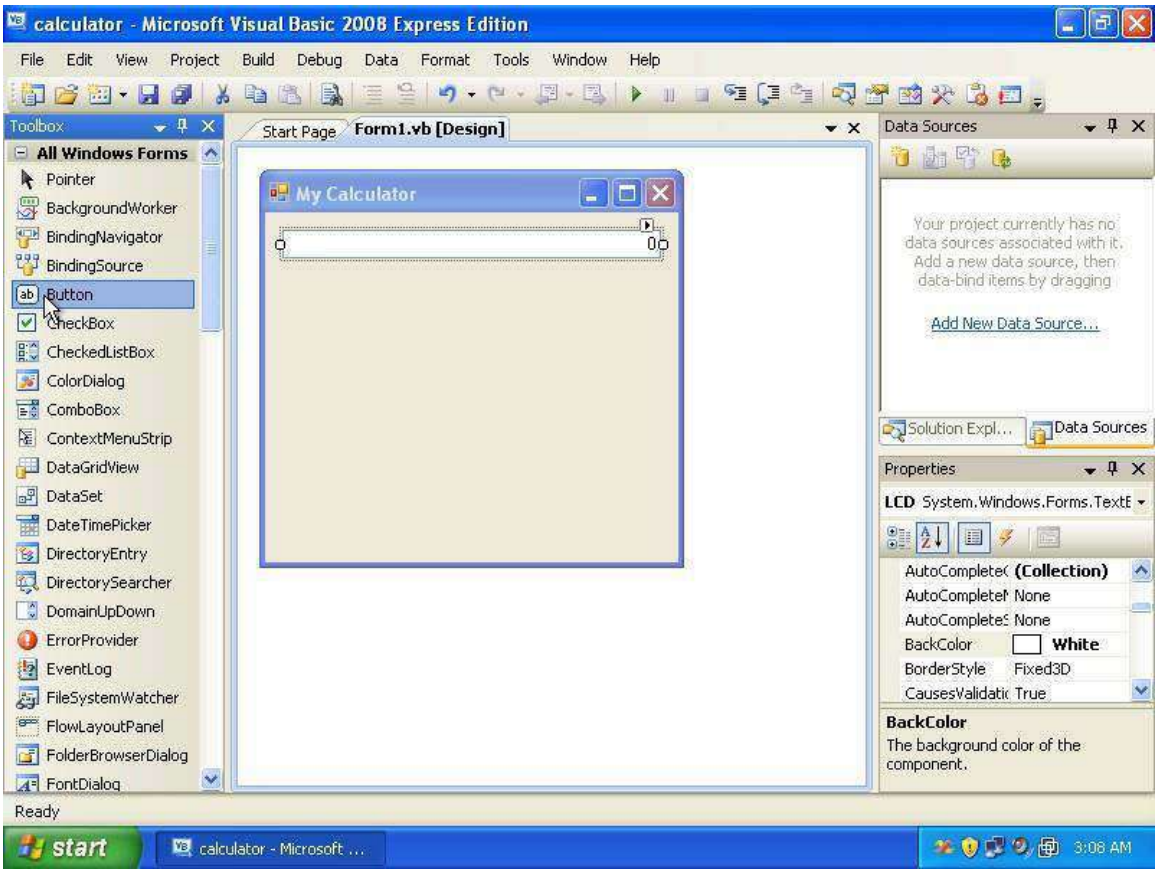
Now if you run the application, you will notice that you can access the text box via the cursor, and write some text, or remove the text. You don't want that to happen. Simply change the **ReadOnly** property to **True**. Notice that once the **ReadOnly** property is ture, the text box will look like this:



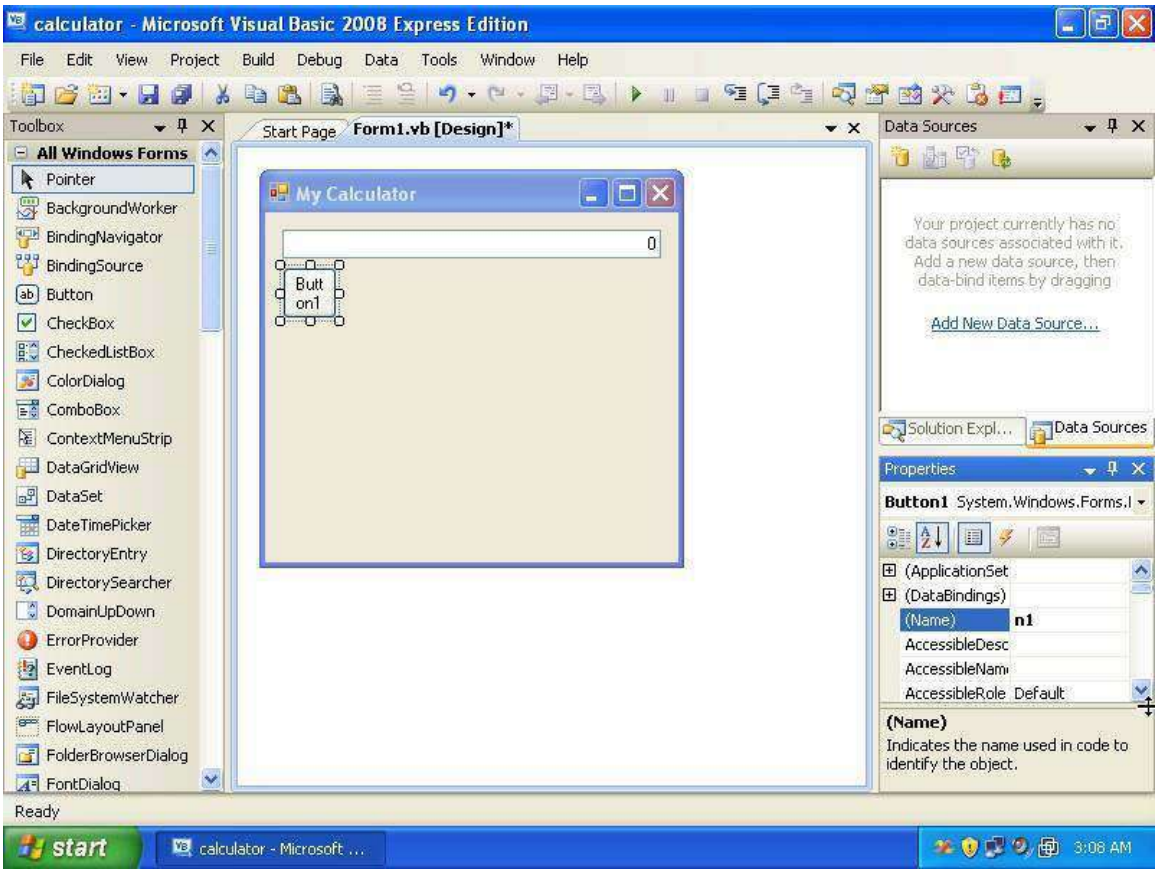
The solution to this problem is simple. Go to the **BackColor** property of the text box and select the white color for it.



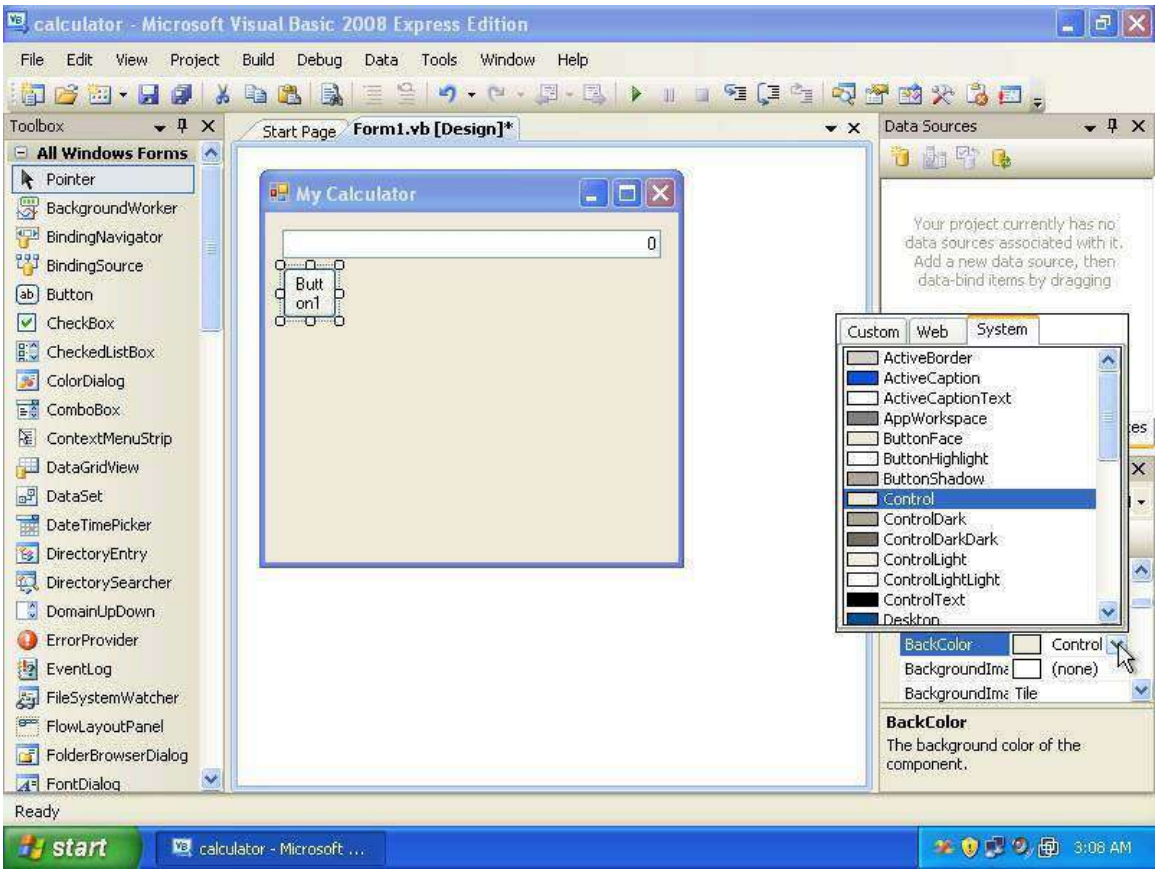
The window should look fine, next we will add the buttons to the form.



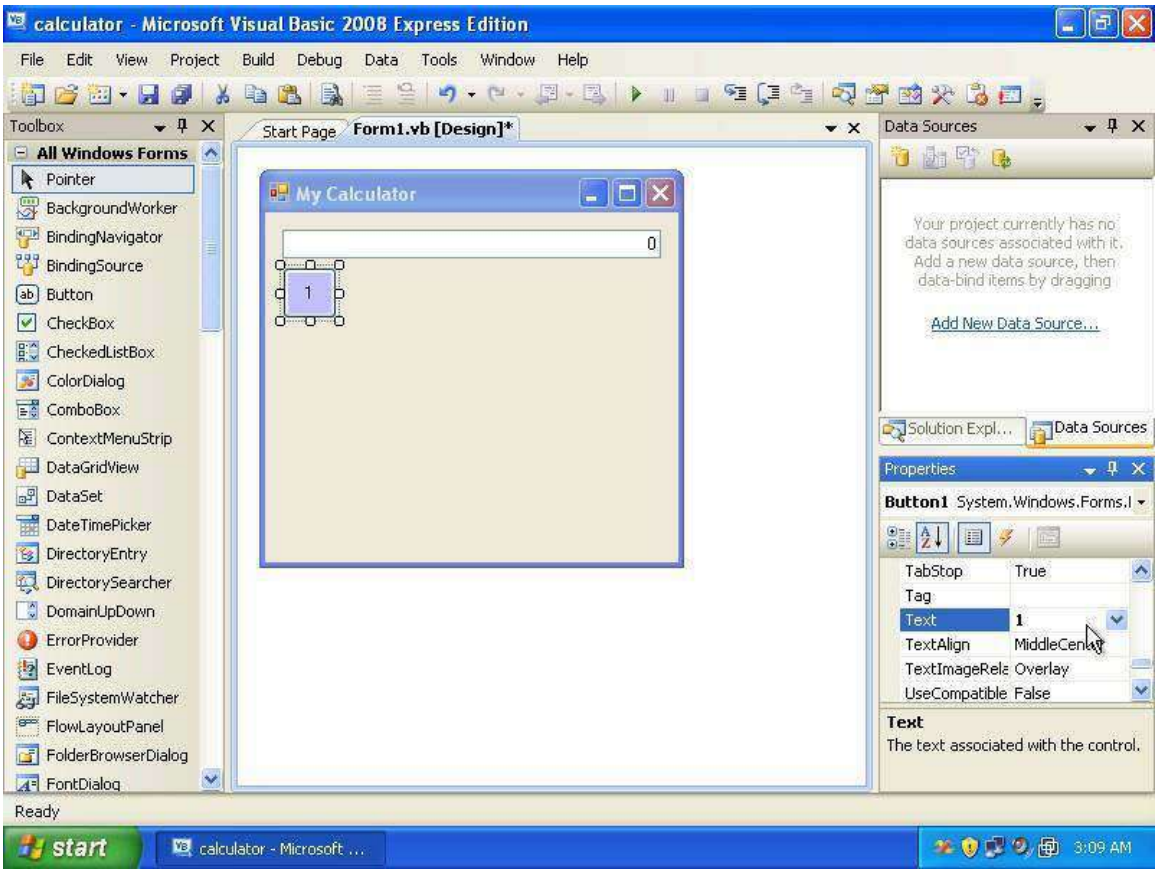
From the tool box window, drag and drop a **Button** onto the form.



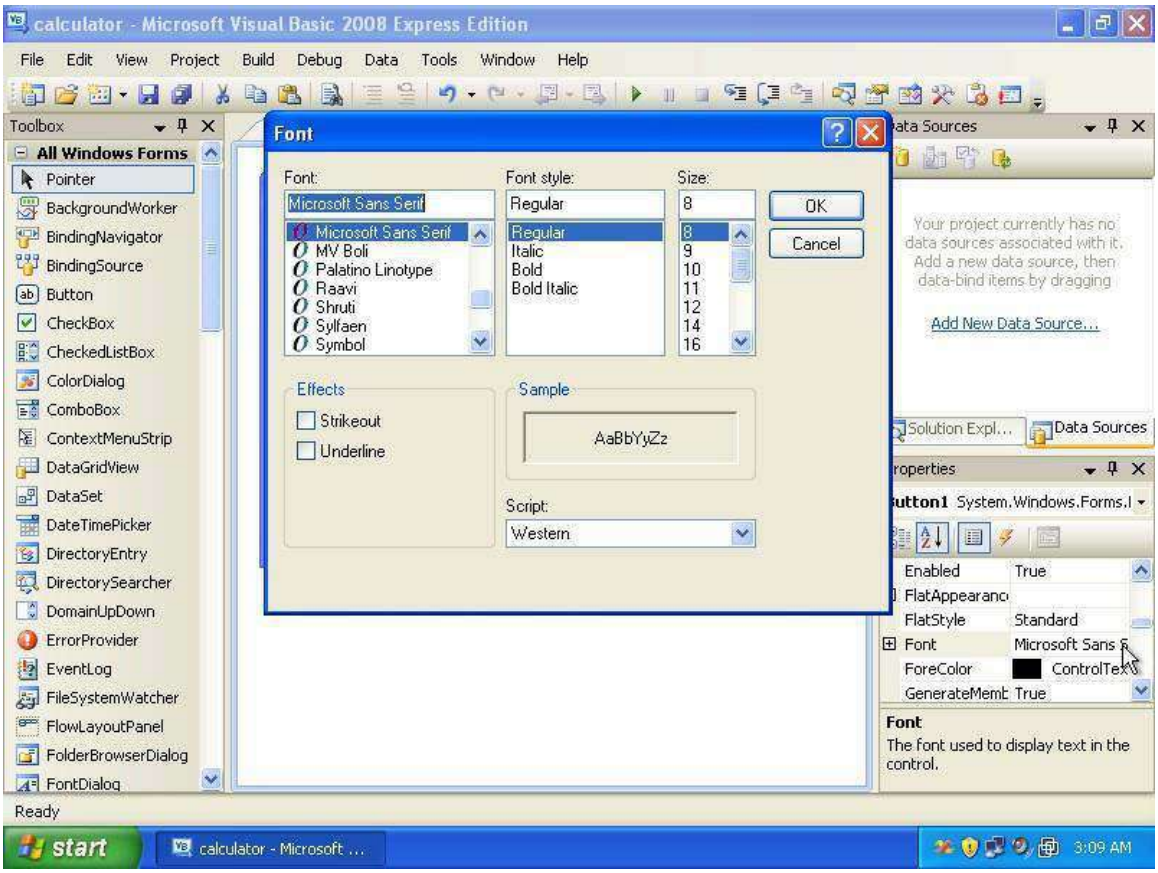
Change its **Name** property to **n1**. This will help us identify which number was pressed.



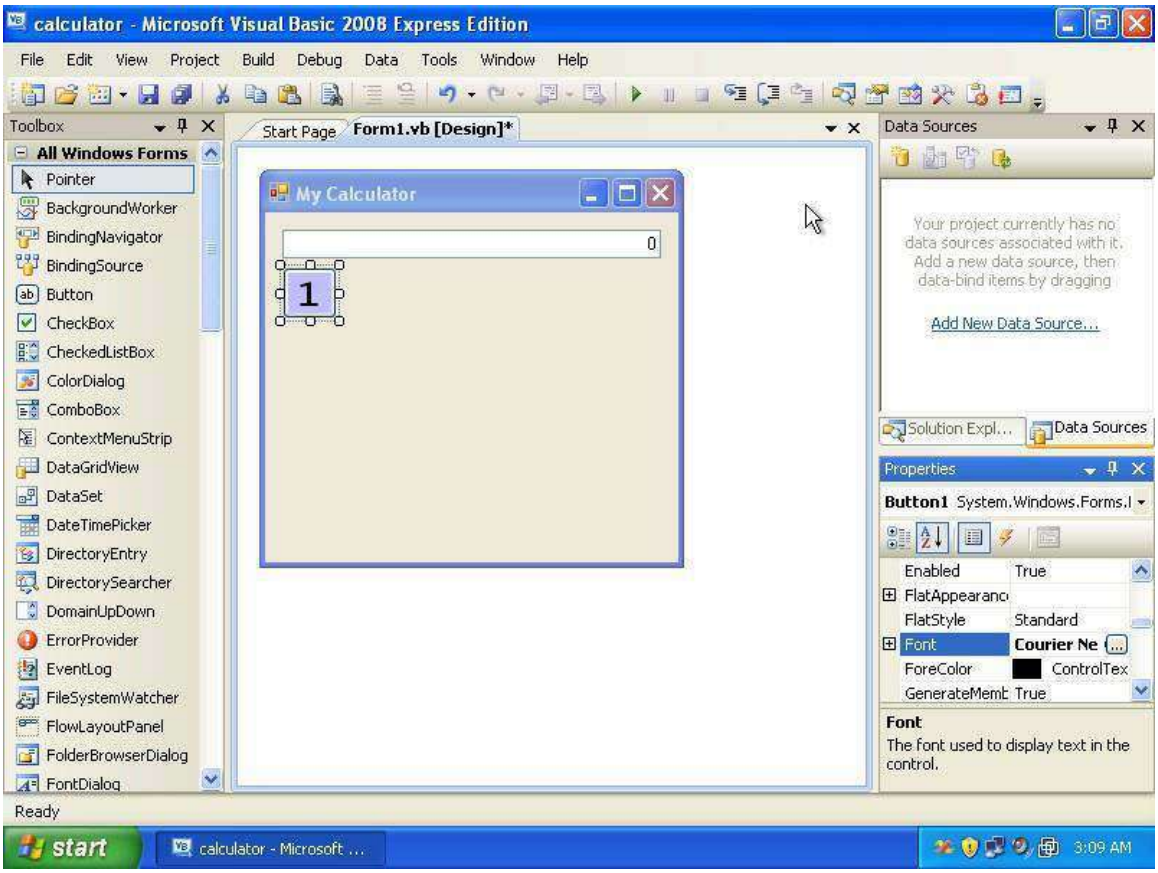
Change the **backcolor** property for the button, usually you can select from **Custom** a different color other than the ones the system provide.



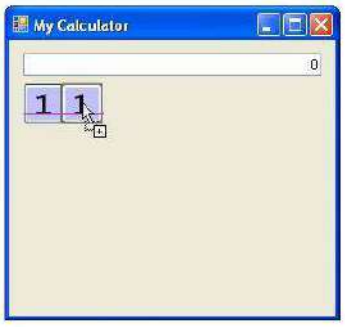
After the color changes, we will modify the text that the button is displaying, so change the text property into **1**.



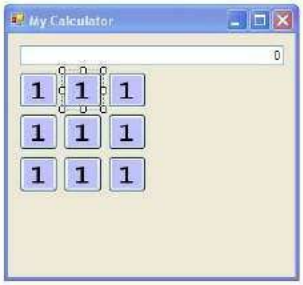
Now the number is very small, we can increase its size a little bit, so go to the **font** property, and set its font to courier new, and size to 20 for example.



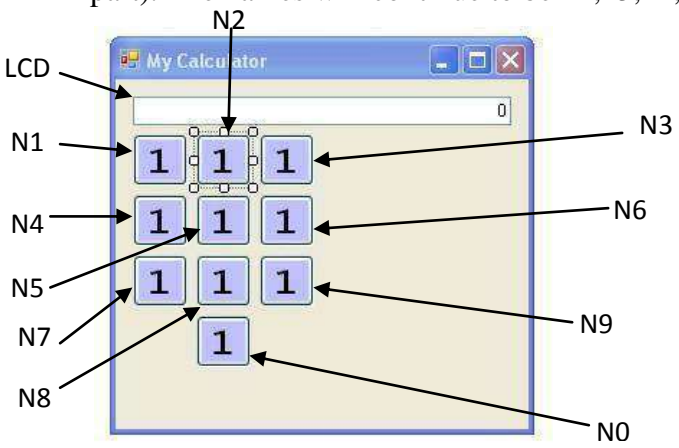
Now we can repeat the same operation to all the other nine buttons, or we can just copy this button and get the same result quickly. Just hold the **ctrl** key and drag the control.



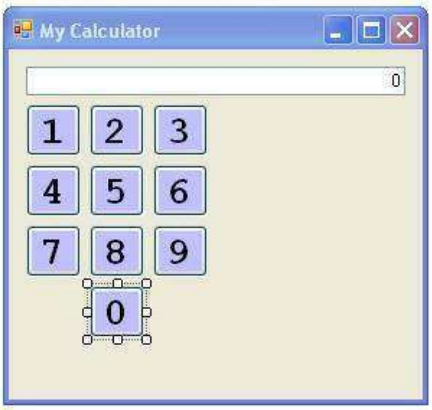
Repeat the operation again for all the numbers



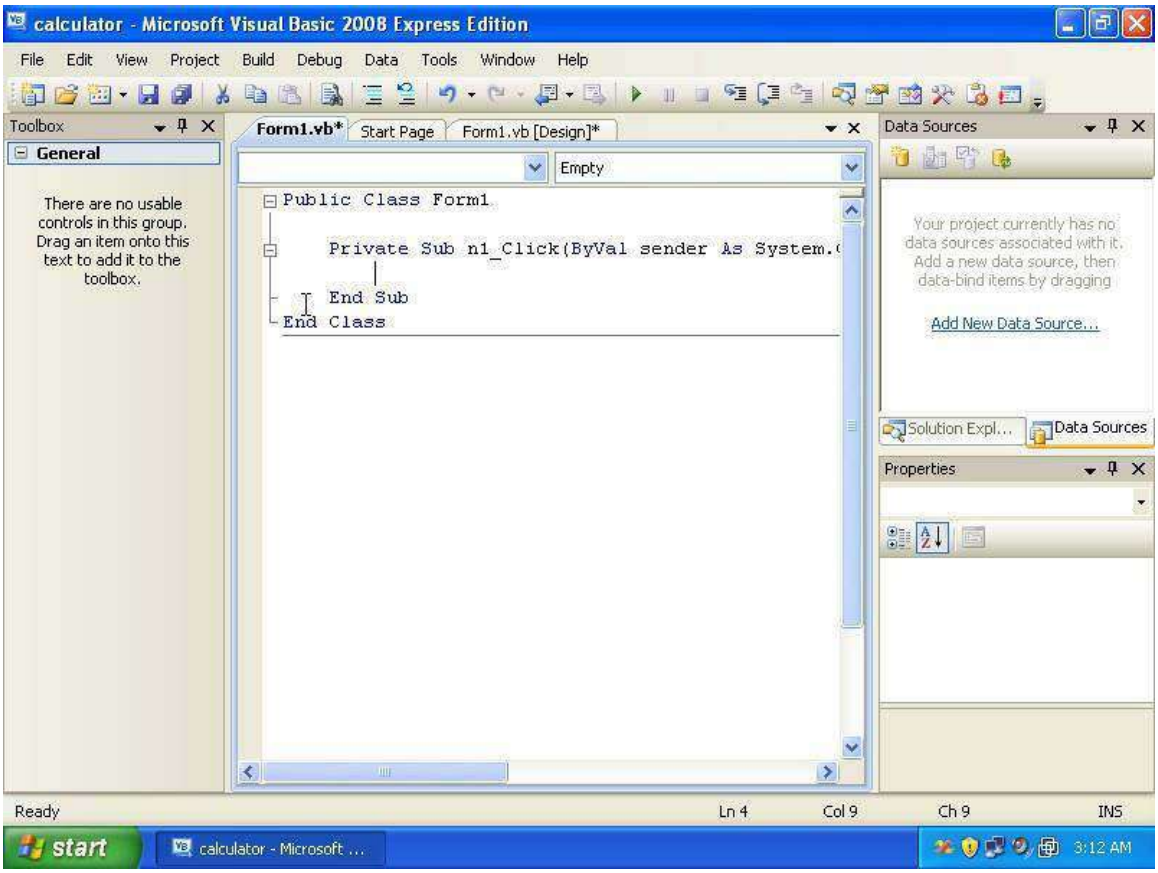
Now we change the names of the buttons (or you can leave them as they are and skip this part). The names will continue to be n2,n3,n4,n5,n6,n7,n8,n9, and n0.



Next we change the text property for each one to display the numbers from 1 to 9, and 0 finally.



Now if you run the application, you won't be able to do anything, i.e. the form appears, and pressing the buttons will have no effect. This is because we haven't tell the application what to do when you click any of the buttons. So, double click the button N1 to go to its event



What you see here is a procedure or subroutine. This is a small block of code that you write to tell the application what to do when a button is clicked. The **n1_Click** is the name of this procedure. It tells you that it get executed when the button whose name **n1** is clicked by the user. Write down the code to have it like this:

```
Private Sub n1_Click(ByVal sender As System.Object, ByVal e As
                        System.EventArgs) Handles n1.Click
      LCD.Text = LCD.Text & "1"
End Sub
```

Note:I made the text small so that it fits one line. In VB.NET the new line is a sentence terminator.

The code means the following:

`Private Sub` part defines the subroutine. It is part of the language (or keywords as they call it).

n1_Click is the name of that subroutine. You can use the name to call the subroutine whenever you need. The name can be anything you choose. More on that in later tutorials.

(ByVal sender As System.Object, ByVal e As System.EventArgs) : these are called the arguments. The allow the subroutine to be flexible by checking some inputs, to process things differently. More on that later.

Handles n1.Click : this one means that this subroutine will be automatically called whenever the button n1 is clicked by the end user.

LCD.Text = LCD.Text & "1" : this is the processing we are performing when we press the button 1. Its meaning is add to the current text on the display (which we call **LCD**) the number 1. Note that we used the Text property which we modified previously using the properties window.

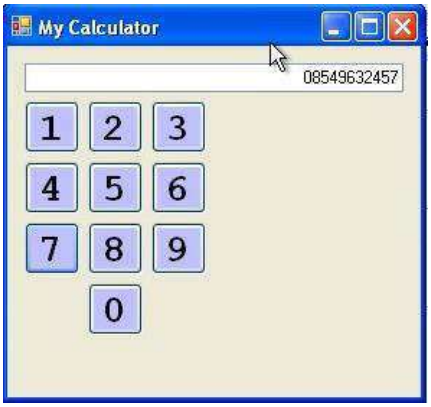
End Sub : signals the end of subroutine.

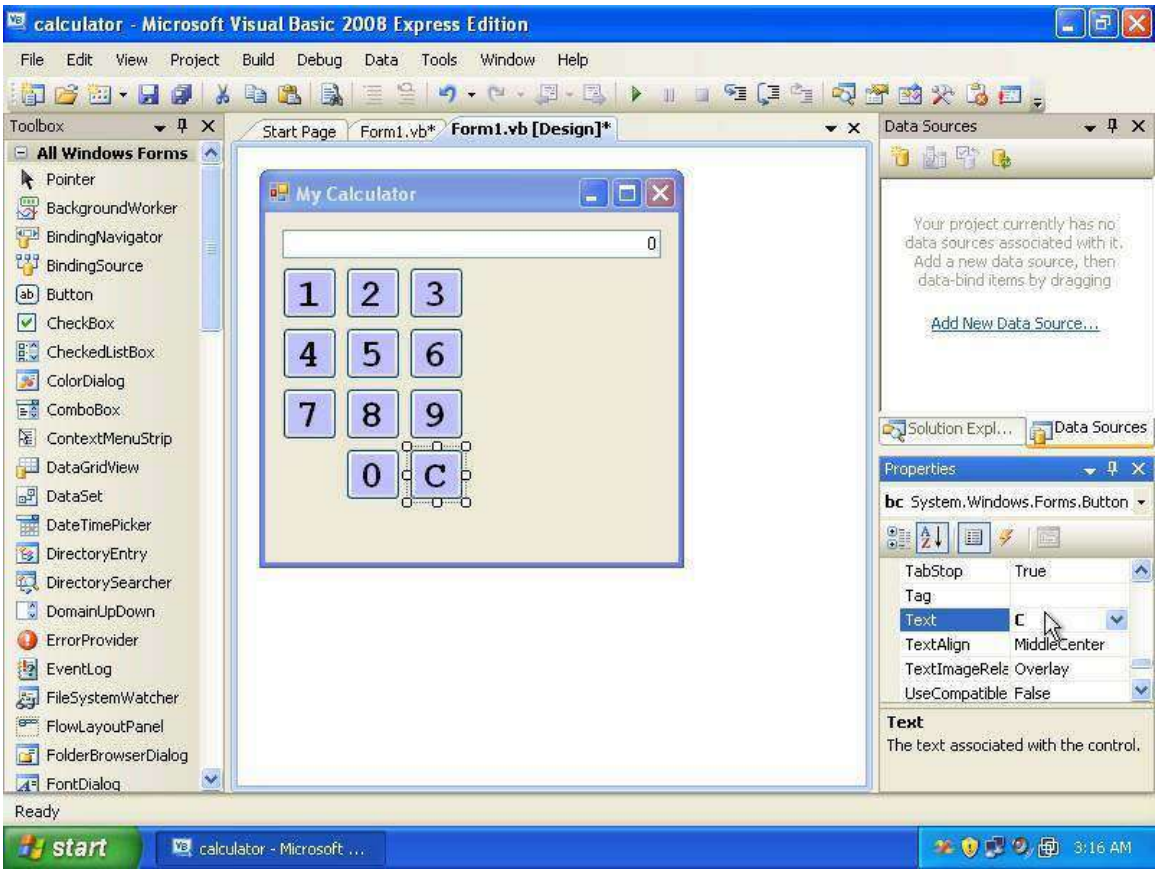
You should repeat the code for buttons n2,n3,n4,n5,n6,n7,n8,n9, and n0 to add the numbers 2,3,4,5,6,7,8,9,0 respectively. So the code should look like this:

```
    Private Sub n1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles n1.Click
        LCD.Text = LCD.Text & "1"
    End Sub

    Private Sub n2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles n2.Click
        LCD.Text = LCD.Text & "2"
    End Sub

    Private Sub n3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles n3.Click
        LCD.Text = LCD.Text & "3"
    End Sub

    Private Sub n4_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles n4.Click
        LCD.Text = LCD.Text & "4"
    End Sub
```

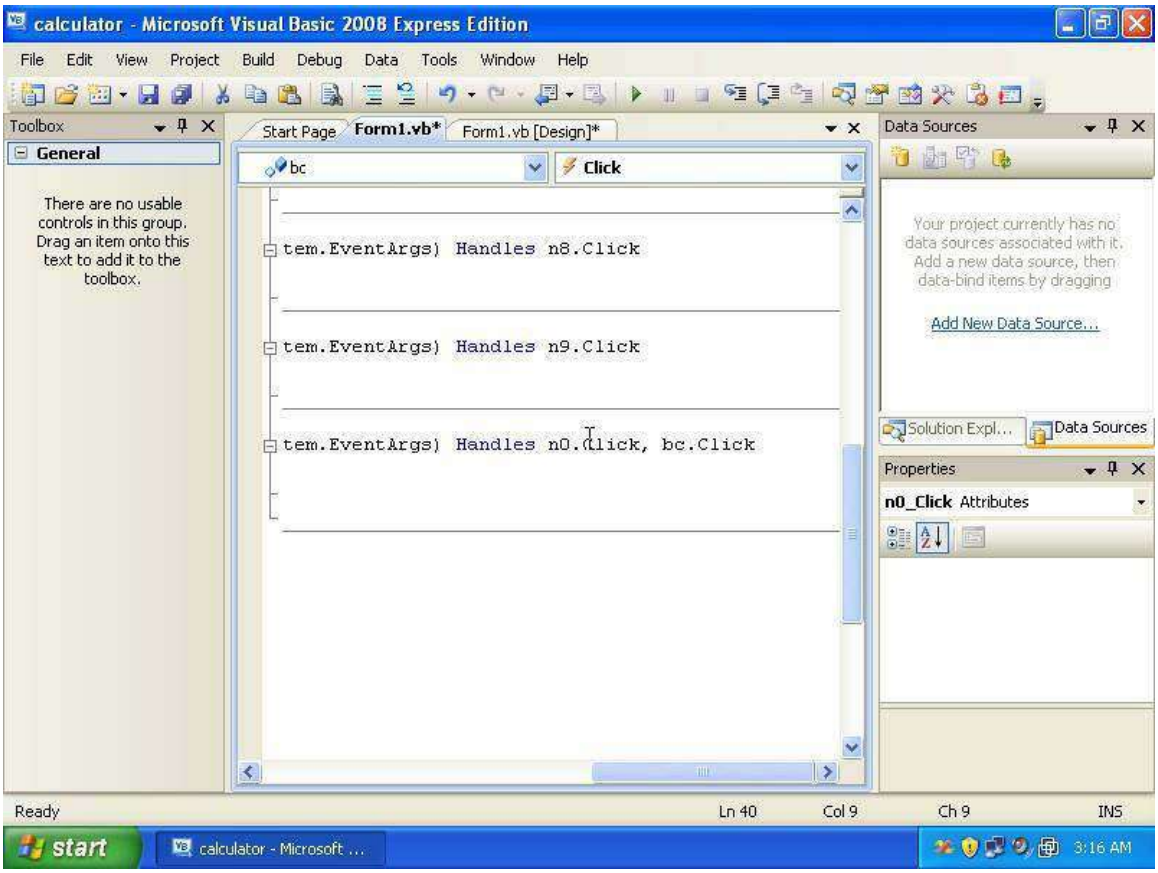
```
    Private Sub n5_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles n5.Click
            LCD.Text = LCD.Text & "5"
    End Sub

    Private Sub n6_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles n6.Click
            LCD.Text = LCD.Text & "6"
    End Sub

    Private Sub n7_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles n7.Click
            LCD.Text = LCD.Text & "7"
    End Sub

    Private Sub n8_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles n8.Click
            LCD.Text = LCD.Text & "8"
    End Sub

    Private Sub n9_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles n9.Click
            LCD.Text = LCD.Text & "9"
    End Sub

    Private Sub n0_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles n0.Click
            LCD.Text = LCD.Text & "0"
    End Sub
```

Now run the application and click few buttons:

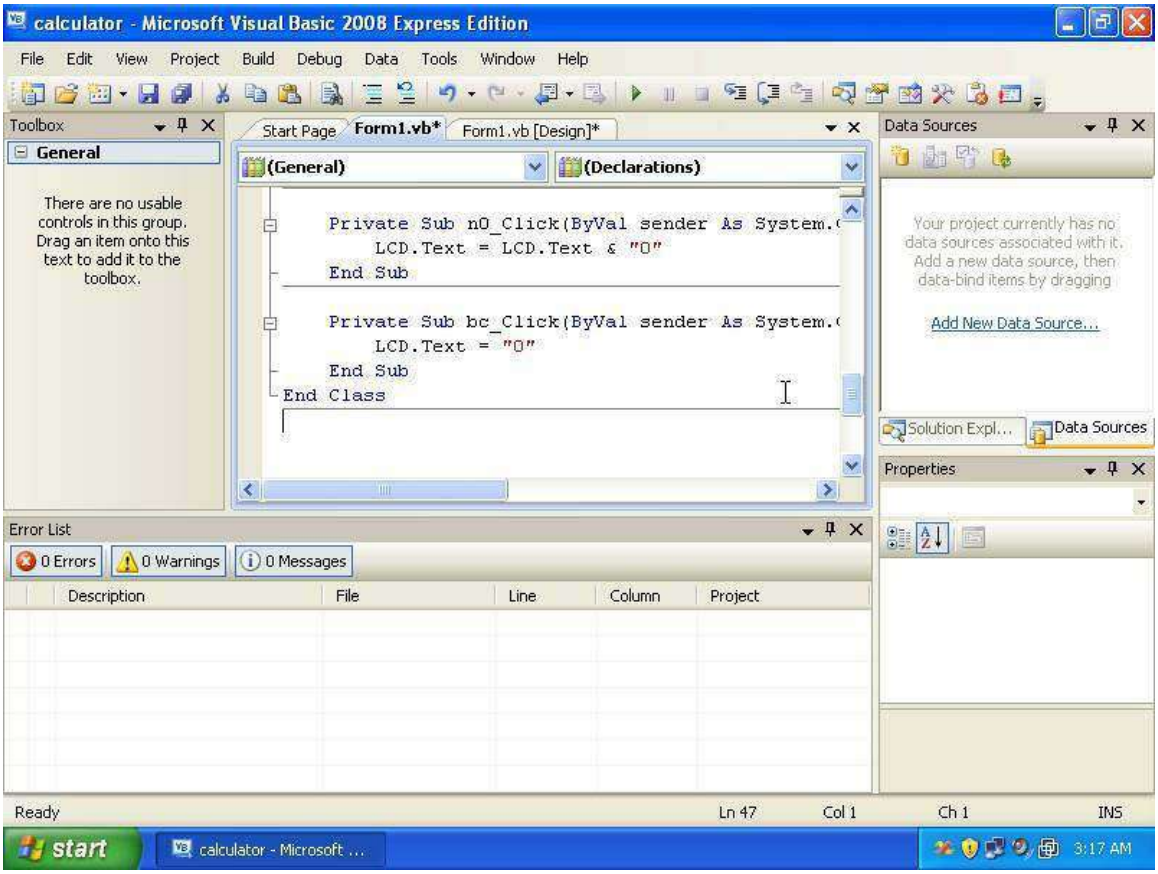




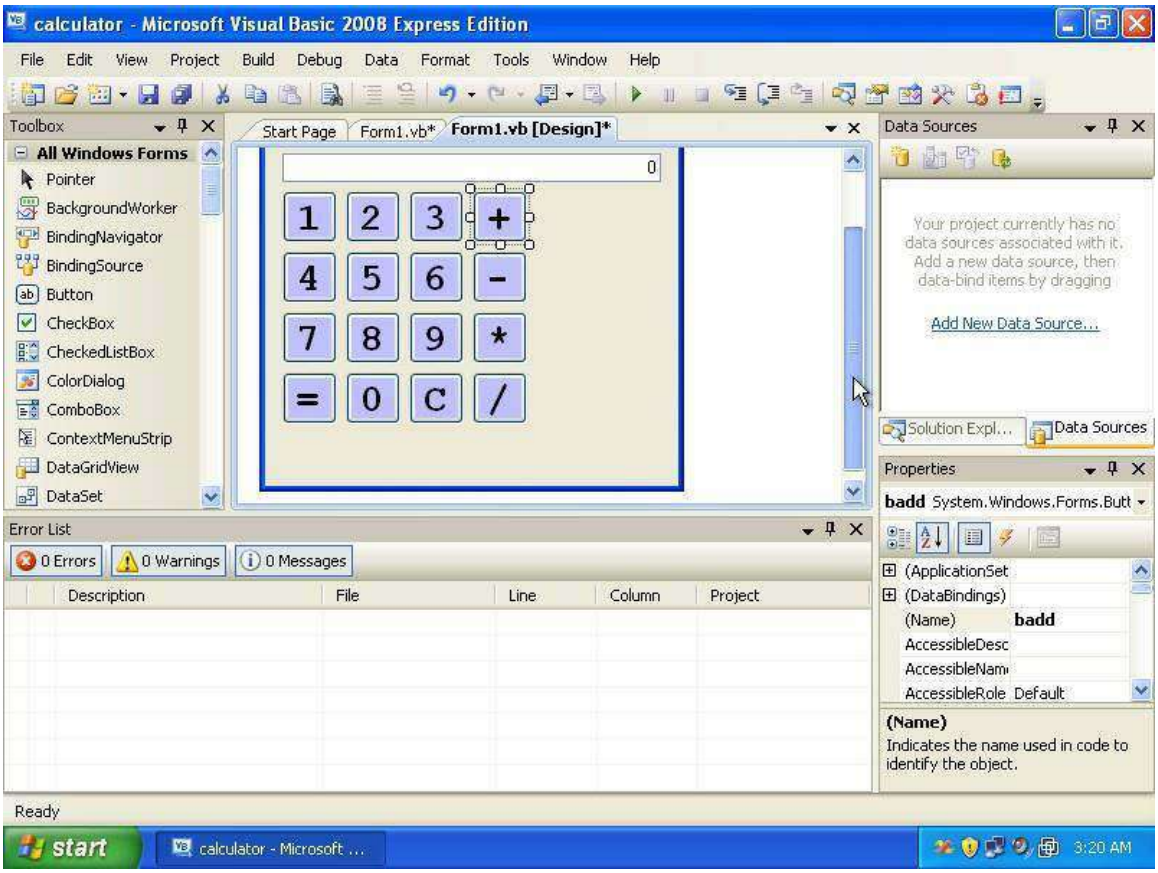
Next we create the clear button, copy the 0 button, and place the copy next to the 0. Change its name to **bc**, and text property to **C**.



Now if you double click the C button, you will go to the code handler of 0. Check the code and you will see that this subroutine will handle the events of **n0.click** and **bc.click**. Actually this will make both buttons behave the same way, so you should create a separate handler for **bc**. The reason why VB.NET linked them together into the same subroutine, is that you copied n0, so VB assumes that the copies should behave the same way as the original. What you should do now is remove the **, bc.Click** from the procedure and then create the handler for clear button.



Just go back to design window, then double click the **C** button and add the text shown above. Try to run the program, clicking some numbers and then pressing the **C** button.



Now assuming you know now how to copy a control and change its text property, now, add the operations as shown above. Next name them as follows:

`+ name is` **`bad`**

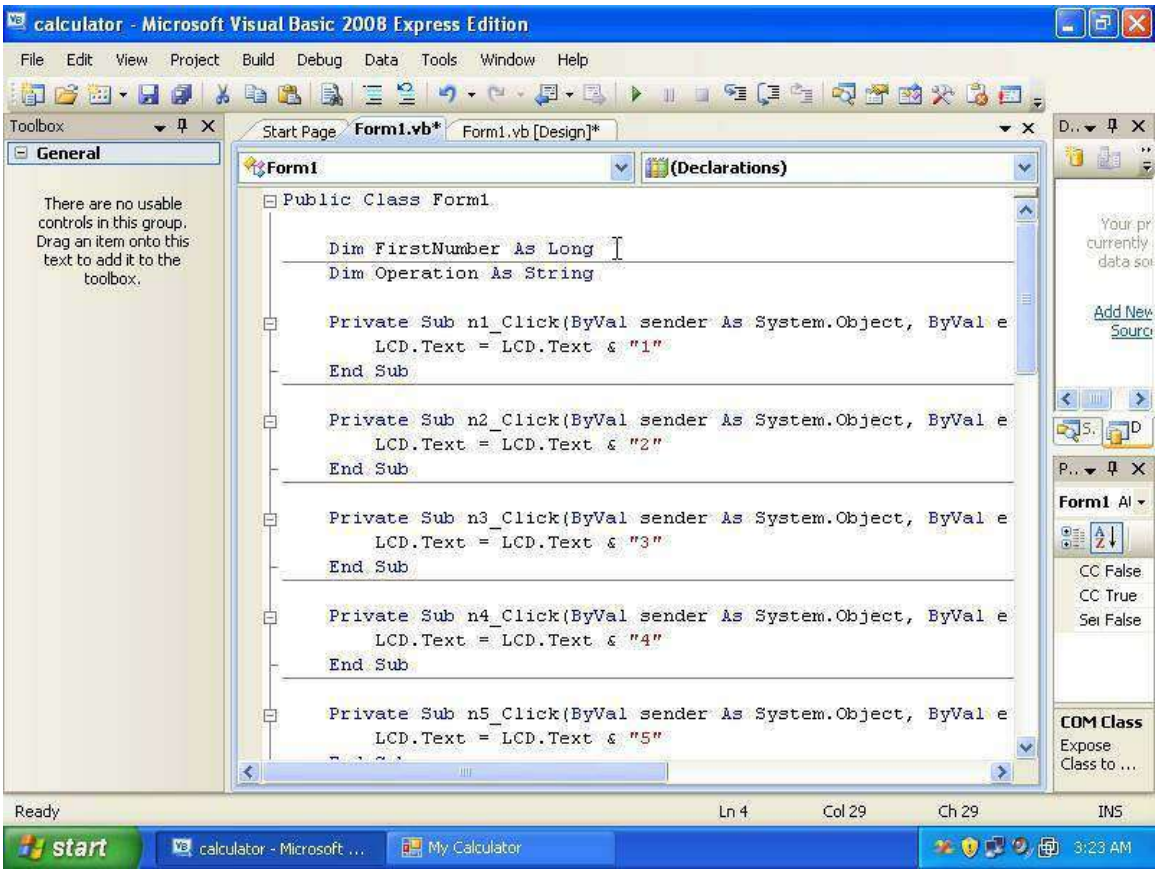
`- name is` **`bsub`**

`* name is` **`bmult`**

`/ name is` **`bdiv`**

`= name is` **`bequal`**

And then remove their handlers from the subroutines because we want to write new event handlers for them.



Double click the + button to go to the code window. Now add the code :

```
Dim FirstNumber As Long
Dim Operation As String
```

FirstNumber is a variable that helps the application remember integer numbers. The reason we use it is to keep track of the last number we entered into the calculator. When we press any operation the display is going to be cleared and so the number is lost, so this variable will store the number before it is removed from the display. Operation is a variable used to remember the last operation that is being pressed. It will store +,-,*,/.

Next add the following code event handles into the subroutines of +,-,*,/ respectively:

```
FirstNumber = LCD.Text
LCD.Text = "0"
Operation = "+"
```

The - event hanlder

```
        FirstNumber = LCD.Text
        LCD.Text = "0"
        Operation = "-"
```

The * event handler

```
        FirstNumber = LCD.Text
        LCD.Text = "0"
        Operation = "*"
```

The / event handler

```
        FirstNumber = LCD.Text
        LCD.Text = "0"
        Operation = "/"
```

So the code should look like

```
    Private Sub badd_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles badd.Click
        FirstNumber = LCD.Text
        LCD.Text = "0"
        Operation = "+"
    End Sub

    Private Sub bsub_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles bsub.Click
        FirstNumber = LCD.Text
        LCD.Text = "0"
        Operation = "-"
    End Sub

    Private Sub bmult_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles bmult.Click
        FirstNumber = LCD.Text
        LCD.Text = "0"
        Operation = "*"
    End Sub
```

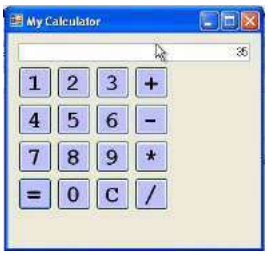
```
    Private Sub bdiv_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles bdiv.Click
        FirstNumber = LCD.Text
        LCD.Text = "0"
        Operation = "/"
    End Sub
```

The last thing to do is the = handler where the operation should be executed

```
    Private Sub bequal_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles bequal.Click
        Dim SecondNumber As Long
        Dim Result As Long

        SecondNumber = LCD.Text

        If Operation = "+" Then
            Result = FirstNumber + SecondNumber
        ElseIf Operation = "-" Then
            Result = FirstNumber - SecondNumber
        ElseIf Operation = "*" Then
            Result = FirstNumber * SecondNumber
        ElseIf Operation = "/" Then
            Result = FirstNumber / SecondNumber
        End If

        FirstNumber = Result
        LCD.Text = Result

    End Sub
```

The code first gets the value from the display, then it check the last operation, if it is addition, then the result will be the sum of the number in memory and the number just we get from the display. If subtraction then subtract one from the other and so on. Finally store the result into memory for further operations on it (you can neglect that, it is not that you have to do it), and then display the result by updating the text property of LCD.

Now run the application and try adding two numbers like 30 and 5:



Next you will notice that when you enter any number you always gets a zero before that number. This is meaningless and should not happen in a calculator. So we will update our code to get rid of the zero in case we clicked on any number (0,1,2,3,4,5,6,7,8,9) and there is a 0 in the display. So update the code to be:

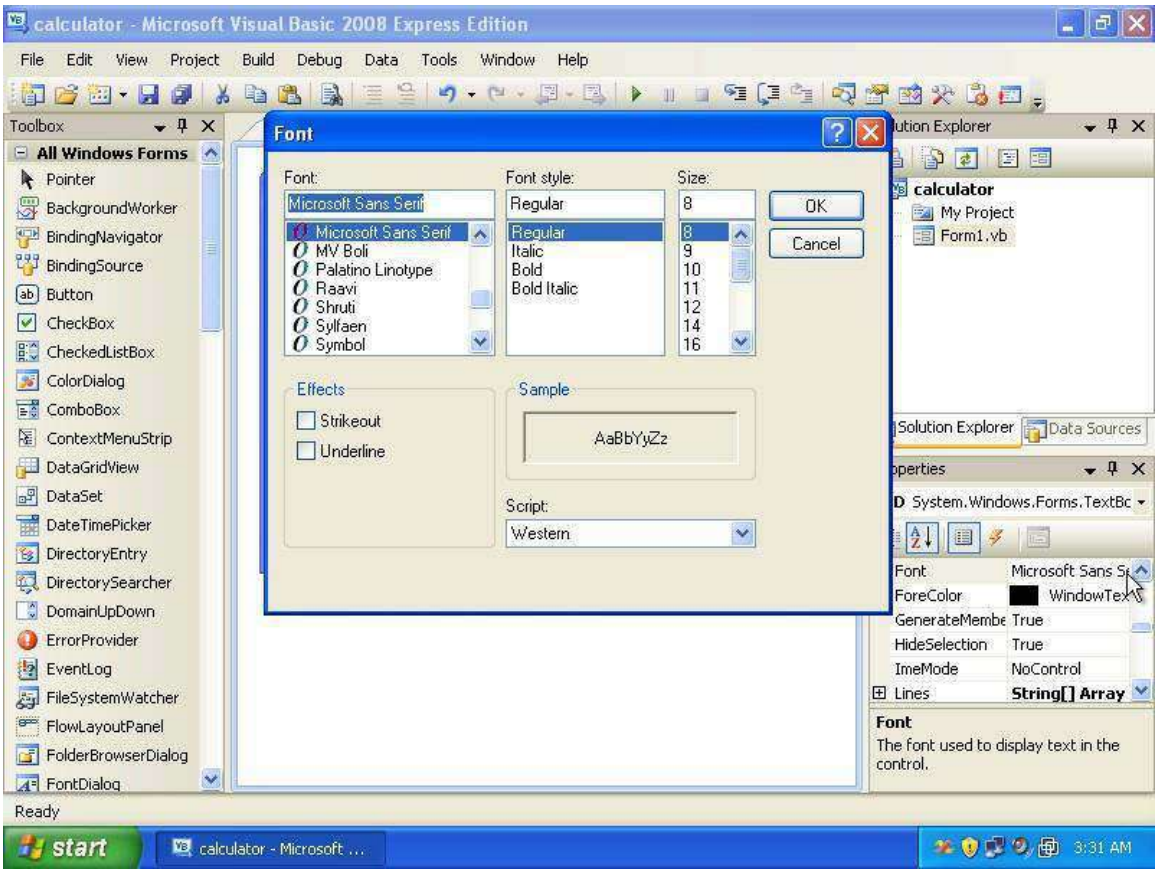
```
    Private Sub n1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles n1.Click
        If LCD.Text = "0" Then
            LCD.Text = "1"
        Else
            LCD.Text = LCD.Text & "1"
        End If
    End Sub

    Private Sub n2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles n2.Click
        If LCD.Text = "0" Then
            LCD.Text = "2"
        Else
            LCD.Text = LCD.Text & "2"
        End If
    End Sub

    Private Sub n3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles n3.Click
        If LCD.Text = "0" Then
            LCD.Text = "3"
        Else
            LCD.Text = LCD.Text & "3"
        End If
    End Sub

    Private Sub n4_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles n4.Click
        If LCD.Text = "0" Then
```

```
            LCD.Text = "4"
        Else
            LCD.Text = LCD.Text & "4"
        End If
    End Sub

    Private Sub n5_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles n5.Click
        If LCD.Text = "0" Then
            LCD.Text = "5"
        Else
            LCD.Text = LCD.Text & "5"
        End If
    End Sub

    Private Sub n6_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles n6.Click
        If LCD.Text = "0" Then
            LCD.Text = "6"
        Else
            LCD.Text = LCD.Text & "6"
        End If
    End Sub

    Private Sub n7_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles n7.Click
        If LCD.Text = "0" Then
            LCD.Text = "7"
        Else
            LCD.Text = LCD.Text & "7"
        End If
    End Sub

    Private Sub n8_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles n8.Click
        If LCD.Text = "0" Then
            LCD.Text = "8"
        Else
            LCD.Text = LCD.Text & "8"
        End If
    End Sub
```

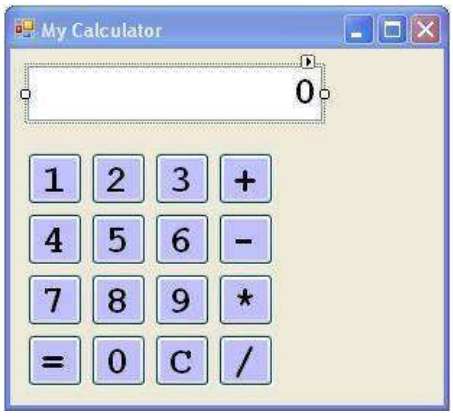
```
    Private Sub n9_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles n9.Click
        If LCD.Text = "0" Then
            LCD.Text = "9"
        Else
            LCD.Text = LCD.Text & "9"
        End If
    End Sub

    Private Sub n0_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles n0.Click
        If LCD.Text = "0" Then
            LCD.Text = "0"
        Else
            LCD.Text = LCD.Text & "0"
        End If
    End Sub
```

This is not the best way to write the code, but it is just to practice working in the environment. Later on we will get into the language itself and understand how it works after understanding most of the controls.

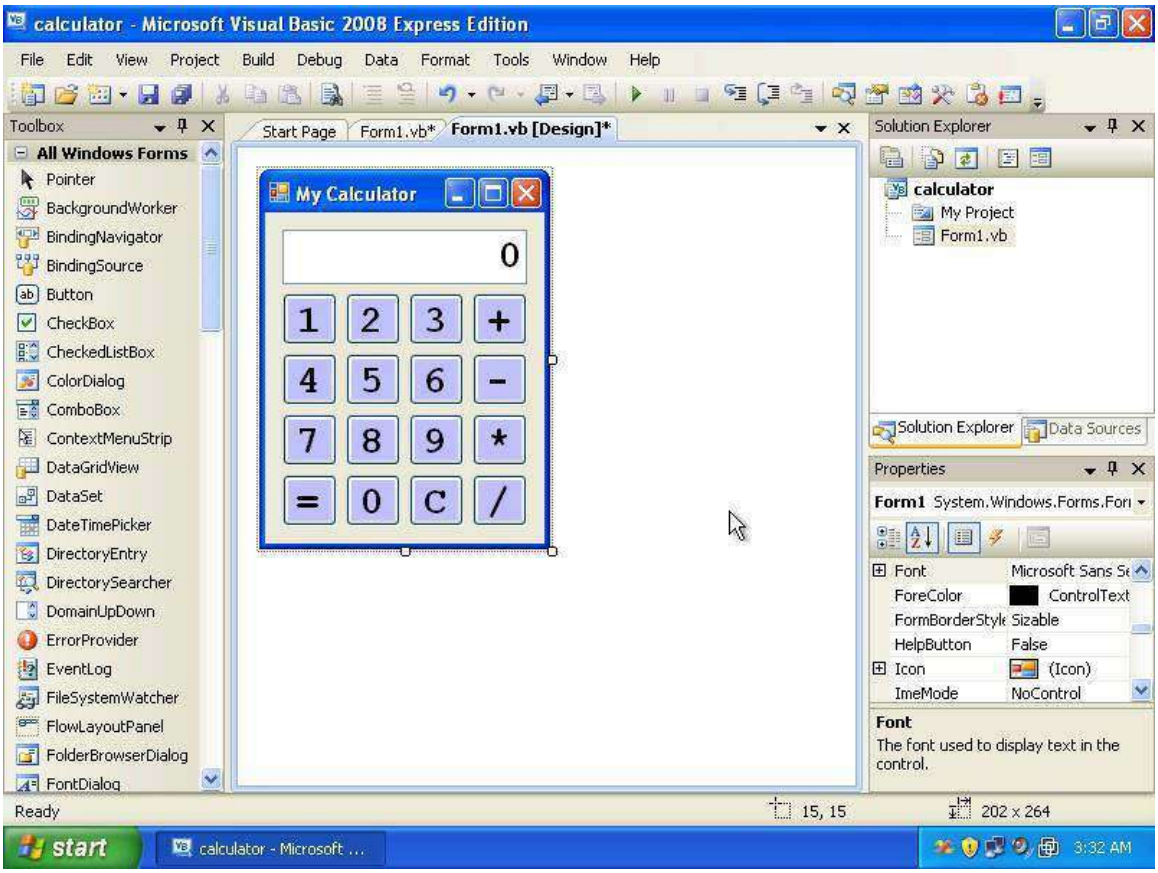
Now run the application and see that the zero disappears when you click any number. After checking that the calculator is working fine, we will make some changes into it. First we change the size of the font in the LCD to be a little bit bigger so select the LCD control (the textbox whose name is LCD), and change its Font property



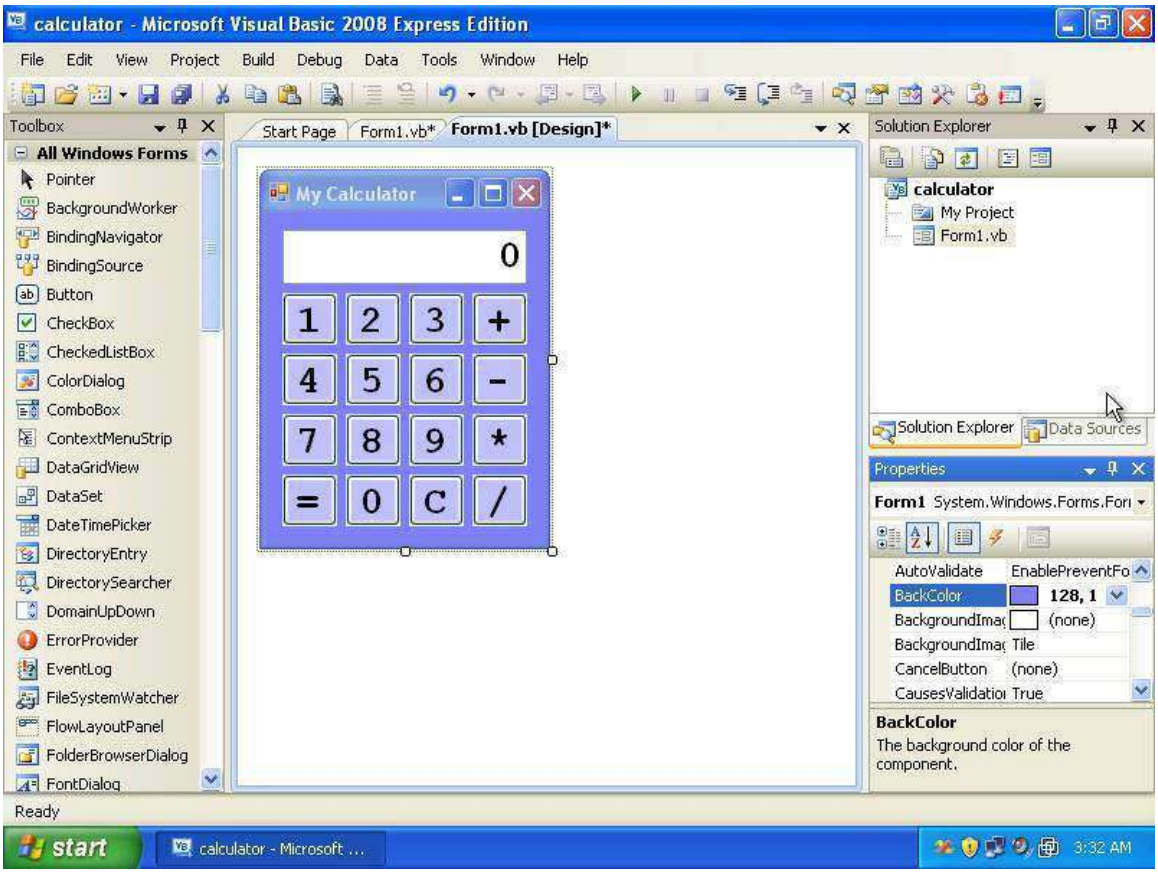
Try to get a result similar to this:



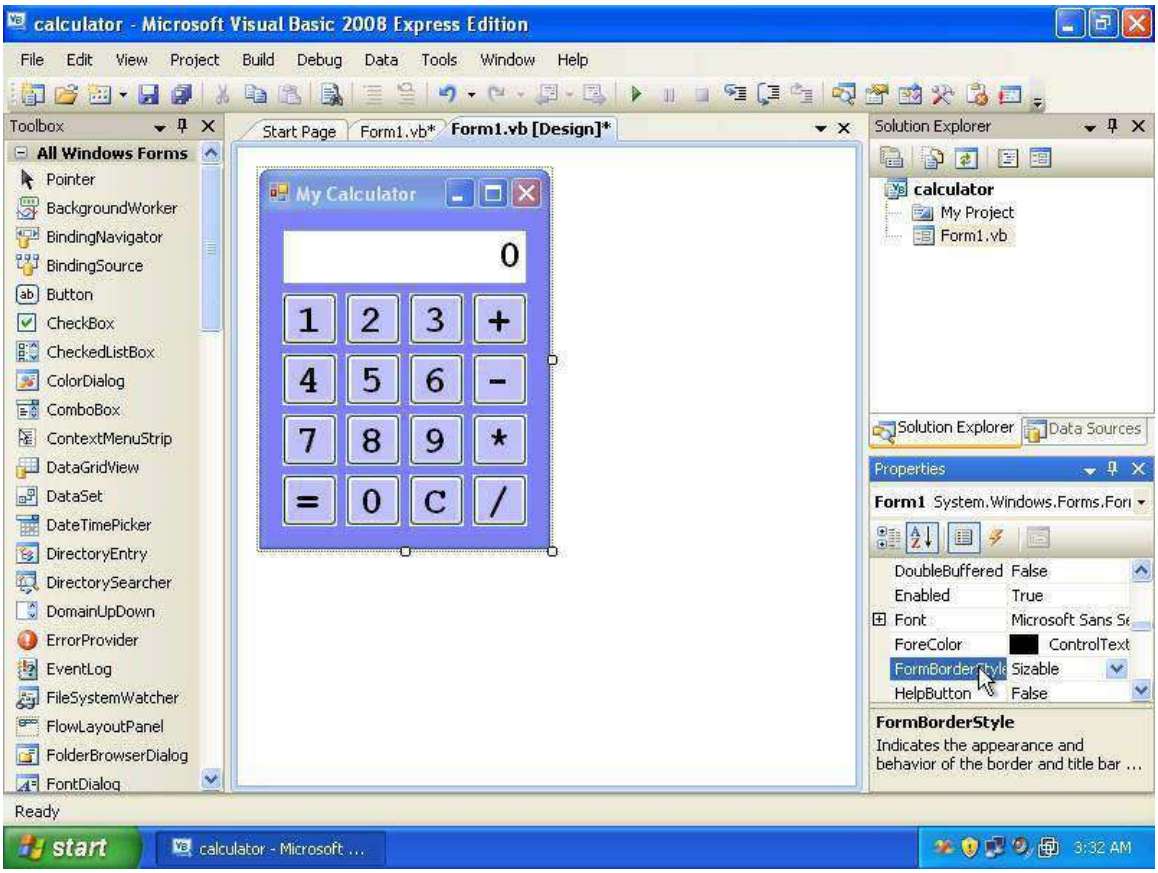
Next we will modify the form behavior so that it does not change size.



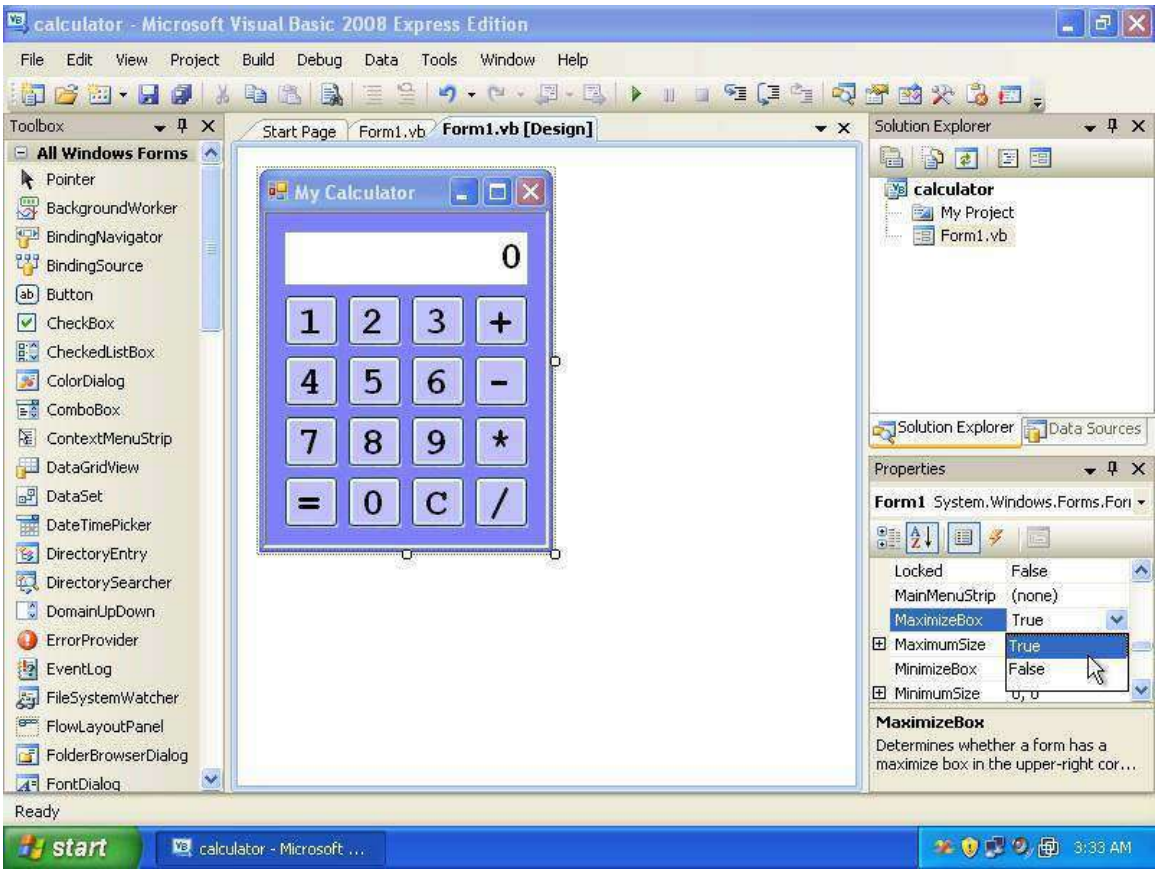
First try to position the controls again to look more professional, you can move all the controls together by selecting them all. The selection works by drawing a box that embed them all. Next click on any empty space on the form so that you can change its properties



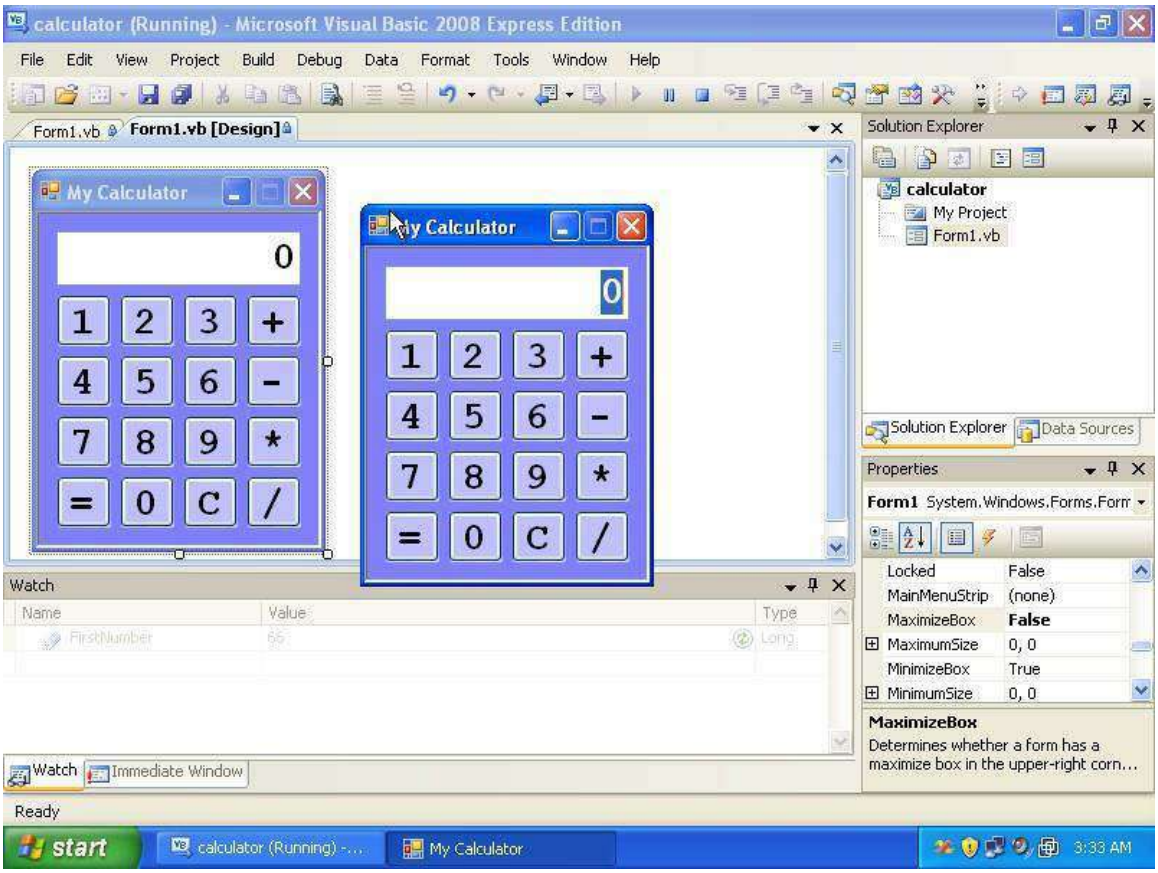
Change **Backcolor** property as you see fit.



Next change the **FormBorderStyle** property from **Sizable** to **FixedSingle**, this will prevent your calculator from being resized.



Finally change the **MaximizeBox** for your form to be **False**. This prevents the form from being maximized.



So this is how your calculator should look like.

Just in case it does not work, the code should be as below assuming you used the same names:

```
Public Class Form1

    Dim FirstNumber As Long
    Dim Operation As String

    Private Sub n1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles n1.Click
        If LCD.Text = "0" Then
            LCD.Text = "1"
        Else
            LCD.Text = LCD.Text & "1"
        End If
    End Sub
```

```
    Private Sub n2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles n2.Click
        If LCD.Text = "0" Then
            LCD.Text = "2"
        Else
            LCD.Text = LCD.Text & "2"
        End If
    End Sub

    Private Sub n3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles n3.Click
        If LCD.Text = "0" Then
            LCD.Text = "3"
        Else
            LCD.Text = LCD.Text & "3"
        End If
    End Sub

    Private Sub n4_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles n4.Click
        If LCD.Text = "0" Then
            LCD.Text = "4"
        Else
            LCD.Text = LCD.Text & "4"
        End If
    End Sub

    Private Sub n5_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles n5.Click
        If LCD.Text = "0" Then
            LCD.Text = "5"
        Else
            LCD.Text = LCD.Text & "5"
        End If
    End Sub

    Private Sub n6_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles n6.Click
        If LCD.Text = "0" Then
            LCD.Text = "6"
        Else
            LCD.Text = LCD.Text & "6"
        End If
```

```
    End Sub

    Private Sub n7_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles n7.Click
        If LCD.Text = "0" Then
            LCD.Text = "7"
        Else
            LCD.Text = LCD.Text & "7"
        End If
    End Sub

    Private Sub n8_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles n8.Click
        If LCD.Text = "0" Then
            LCD.Text = "8"
        Else
            LCD.Text = LCD.Text & "8"
        End If
    End Sub

    Private Sub n9_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles n9.Click
        If LCD.Text = "0" Then
            LCD.Text = "9"
        Else
            LCD.Text = LCD.Text & "9"
        End If
    End Sub

    Private Sub n0_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles n0.Click
        If LCD.Text = "0" Then
            LCD.Text = "0"
        Else
            LCD.Text = LCD.Text & "0"
        End If
    End Sub

    Private Sub bc_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles bc.Click
        LCD.Text = "0"
    End Sub
```

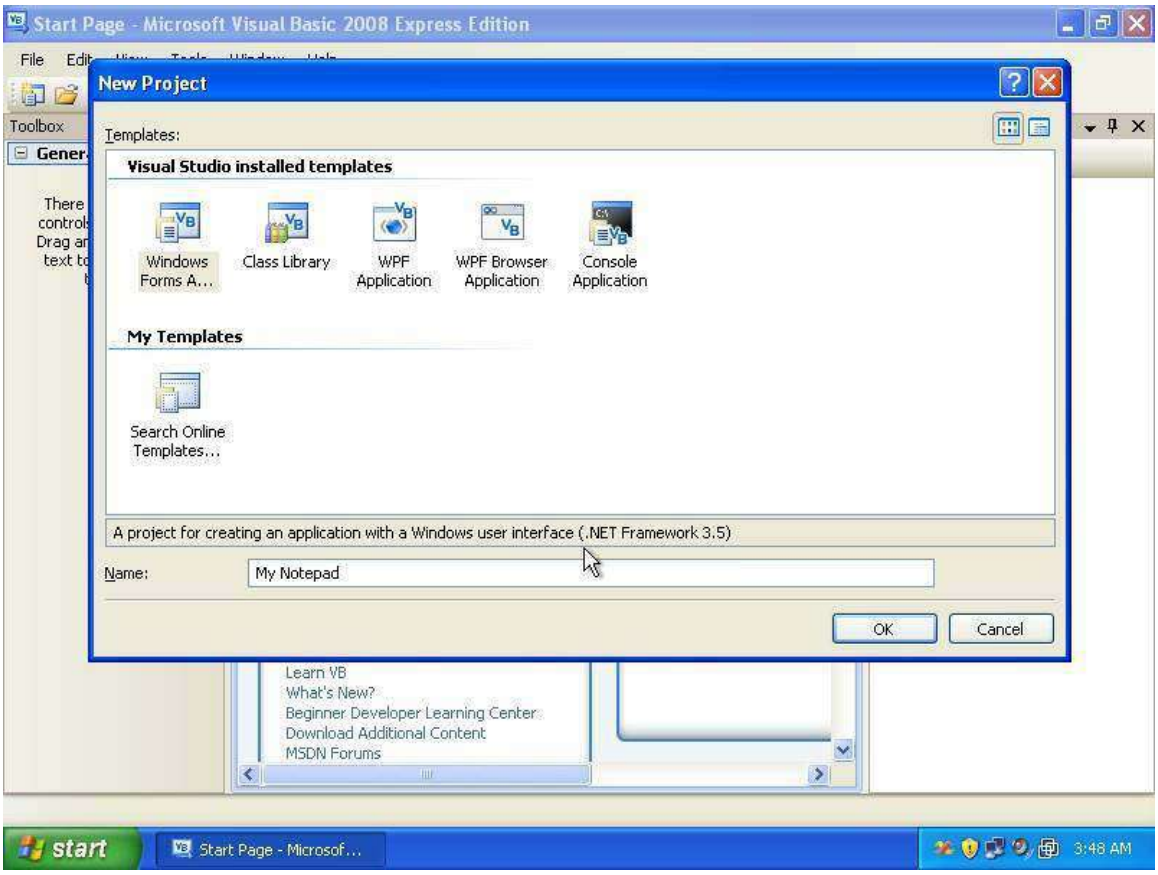
```
    Private Sub badd_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles badd.Click
        FirstNumber = LCD.Text
        LCD.Text = "0"
        Operation = "+"
    End Sub

    Private Sub bsub_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles bsub.Click
        FirstNumber = LCD.Text
        LCD.Text = "0"
        Operation = "-"
    End Sub

    Private Sub bmult_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles bmult.Click
        FirstNumber = LCD.Text
        LCD.Text = "0"
        Operation = "*"
    End Sub

    Private Sub bdiv_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles bdiv.Click
        FirstNumber = LCD.Text
        LCD.Text = "0"
        Operation = "/"
    End Sub

    Private Sub bequal_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles bequal.Click
        Dim SecondNumber As Long
        Dim Result As Long

        SecondNumber = LCD.Text

        If Operation = "+" Then
            Result = FirstNumber + SecondNumber
        ElseIf Operation = "-" Then
            Result = FirstNumber - SecondNumber
        ElseIf Operation = "*" Then
            Result = FirstNumber * SecondNumber
        ElseIf Operation = "/" Then
            Result = FirstNumber / SecondNumber
```

```
        End If

        FirstNumber = Result
        LCD.Text = Result

    End Sub
End Class
```

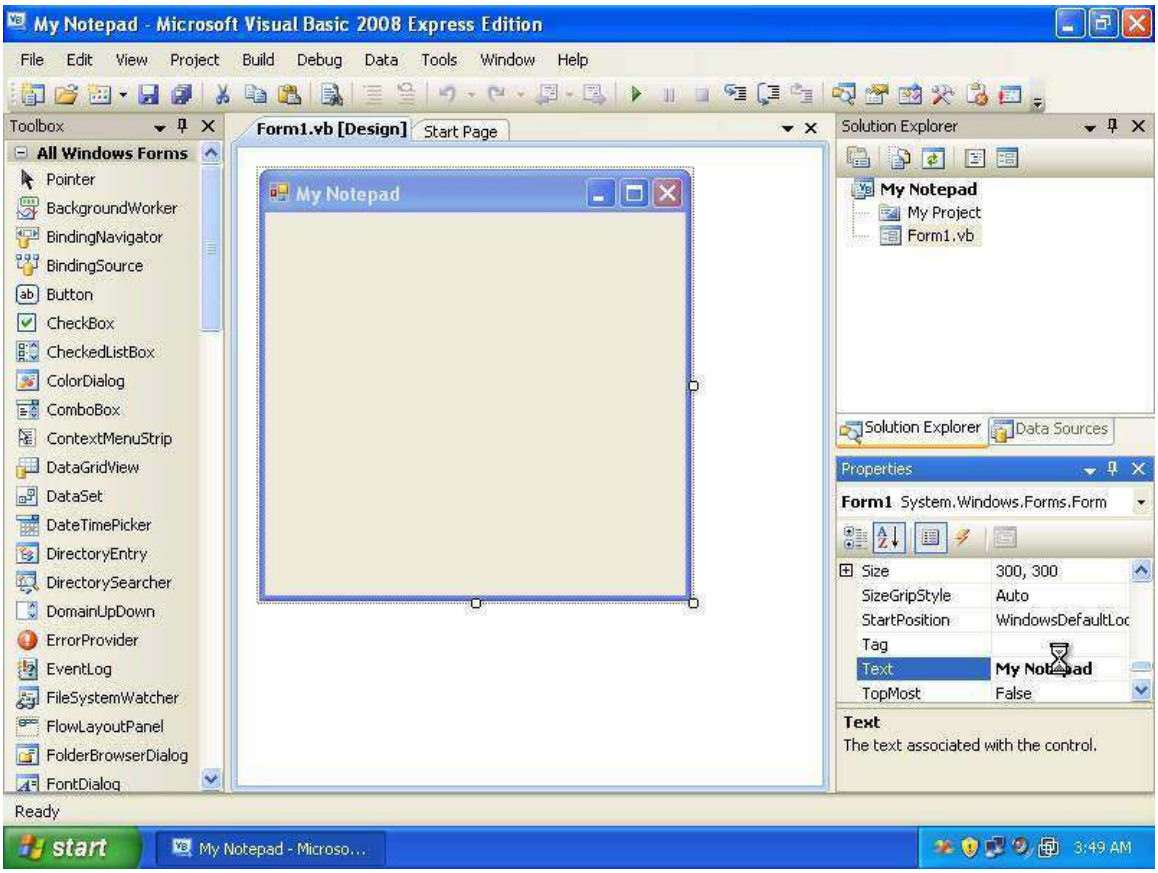
Chapter 4: Dialogs and Menus

Dialogs and Menus

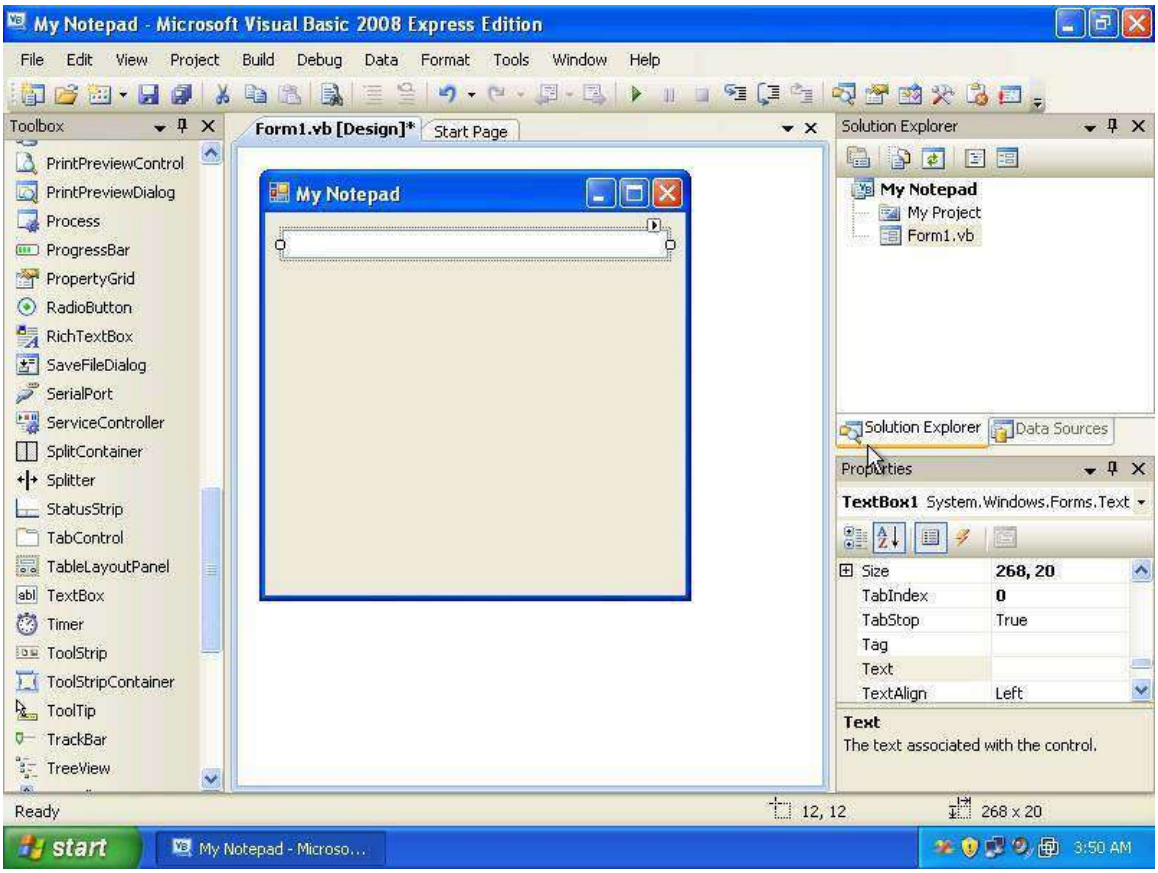
In this chapter we will be designing a simple Notepad application. Please notice that this chapter focuses only on the controls not the code. More about the code later…



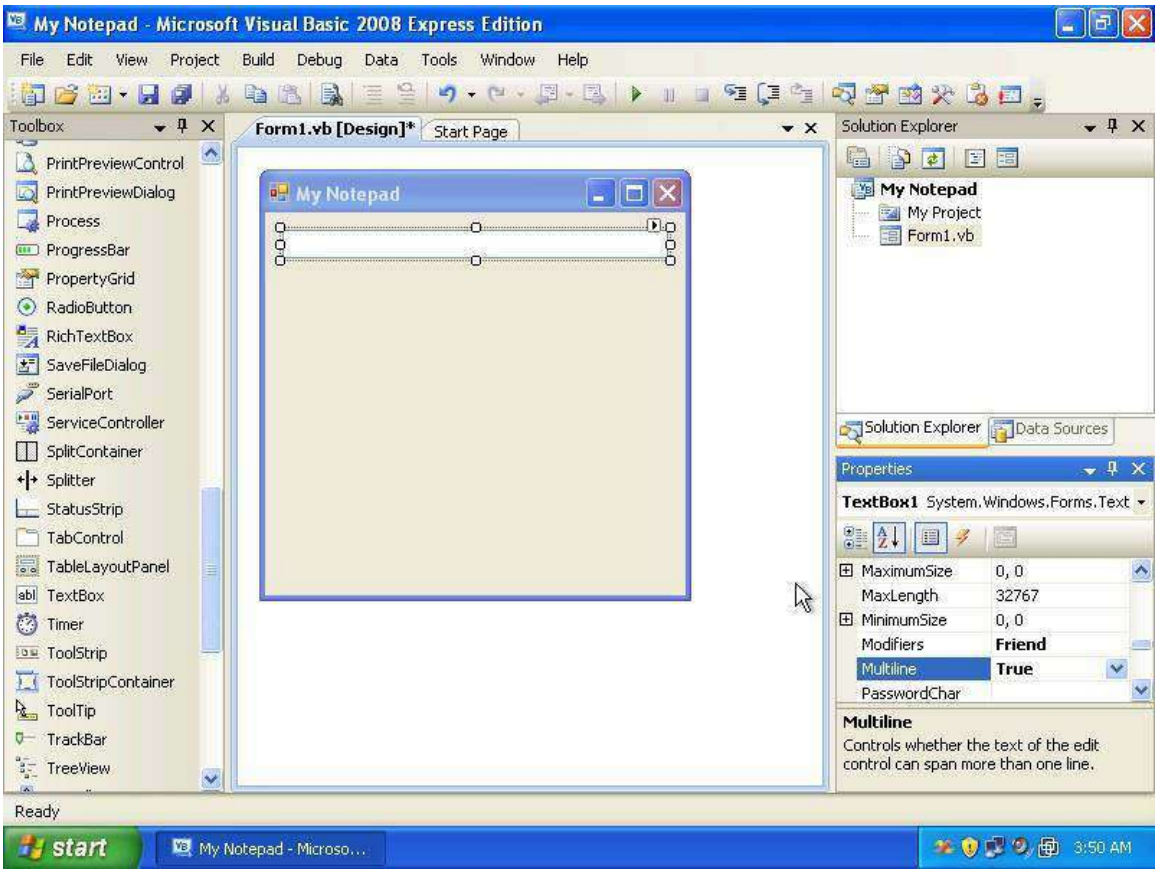
Create a new project and name it: My Notepad



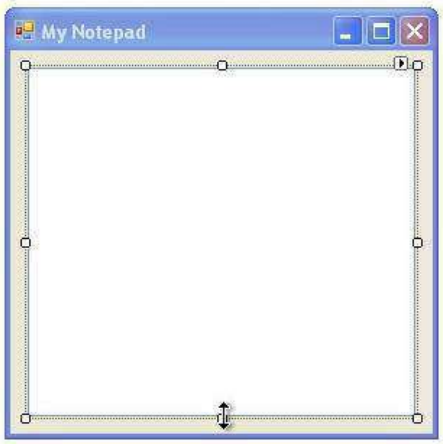
Rename the form title to **My Notepad** by modifying the **Text** property of the form.



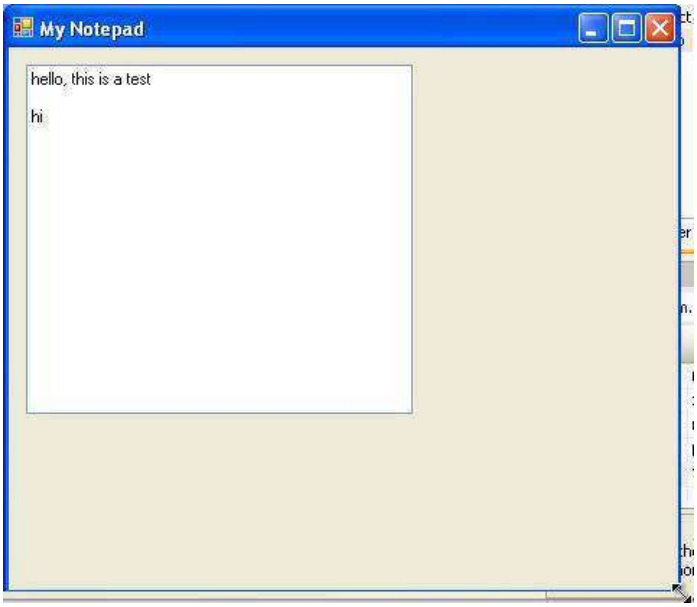
Drop a text box on the form.



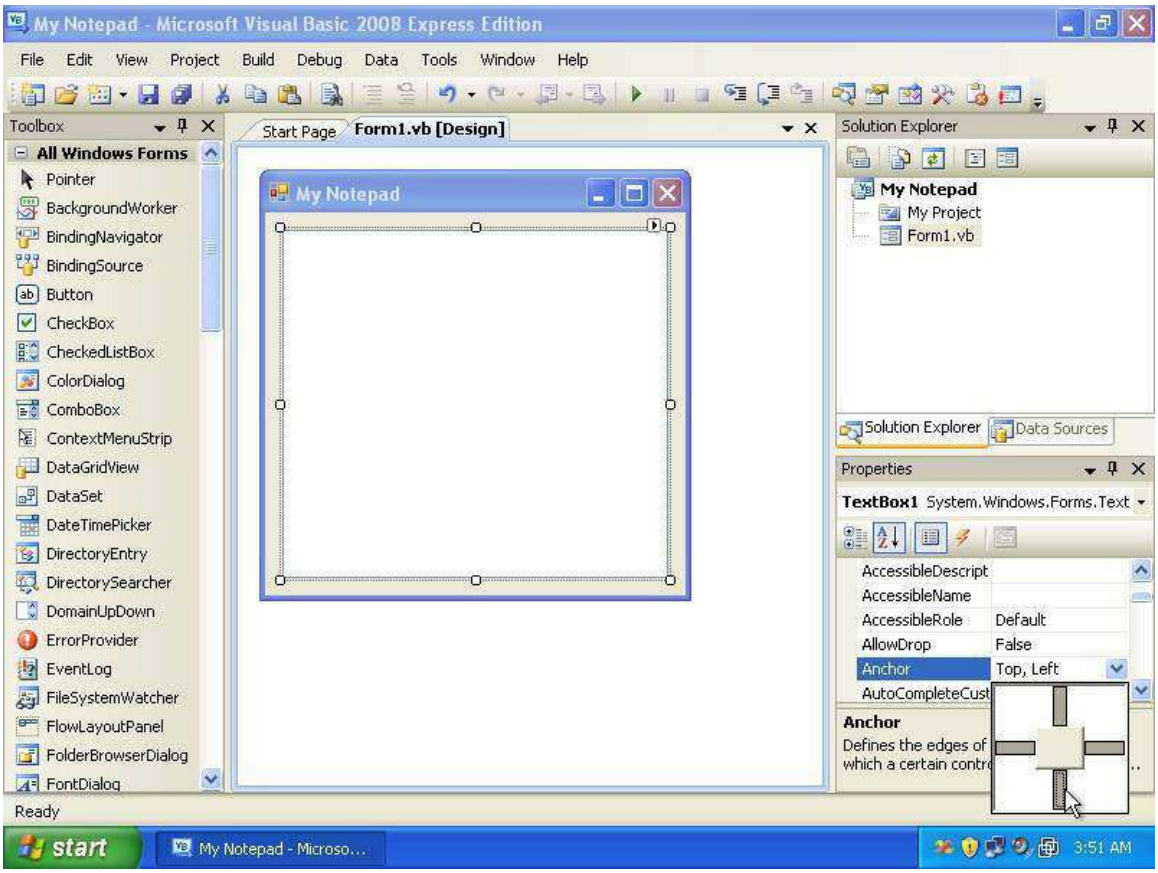
The problem with text boxes are that you can only use it to write a single line. To solve this, modify the text box property **Multiline** to **True**. This will allow you to write multiple lines in the text box and modify its height. Resize the text box to take the full size of the window



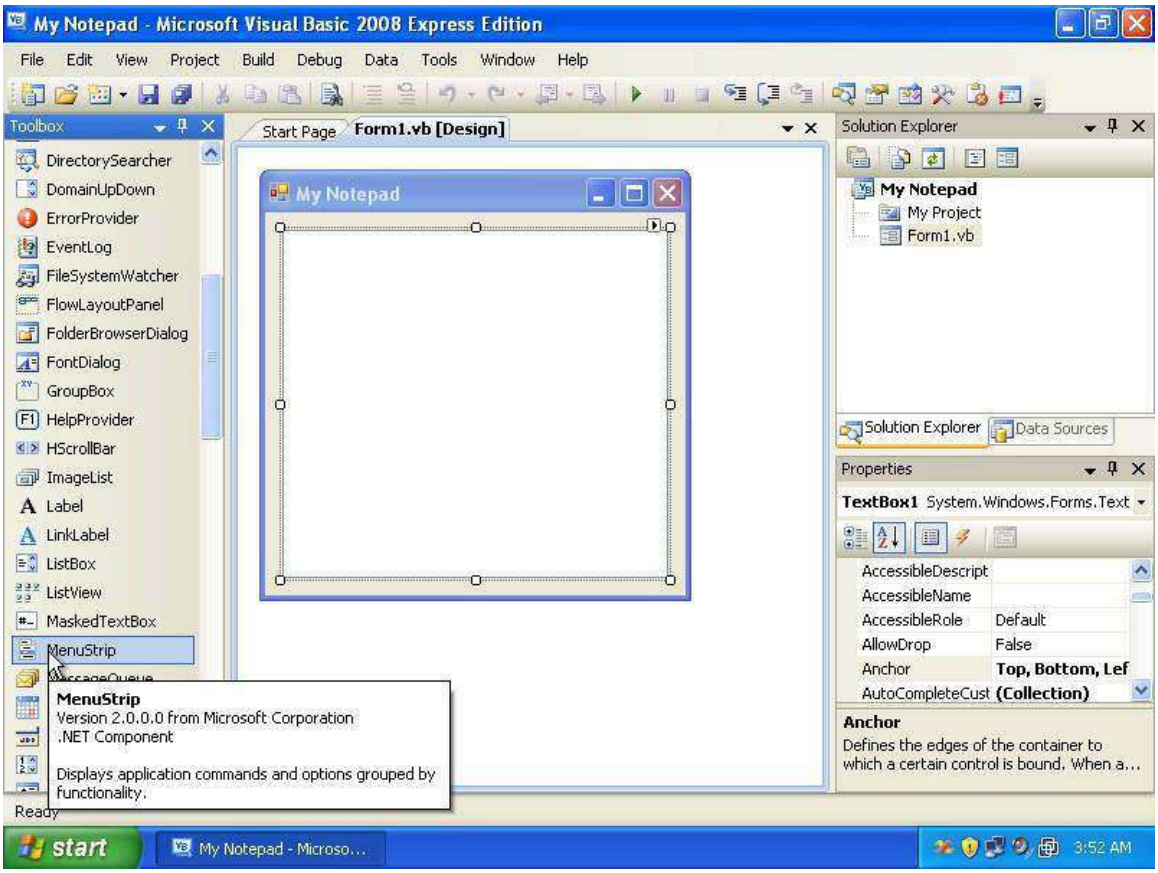
Run the application, and try to write some text. You can see it works fine.



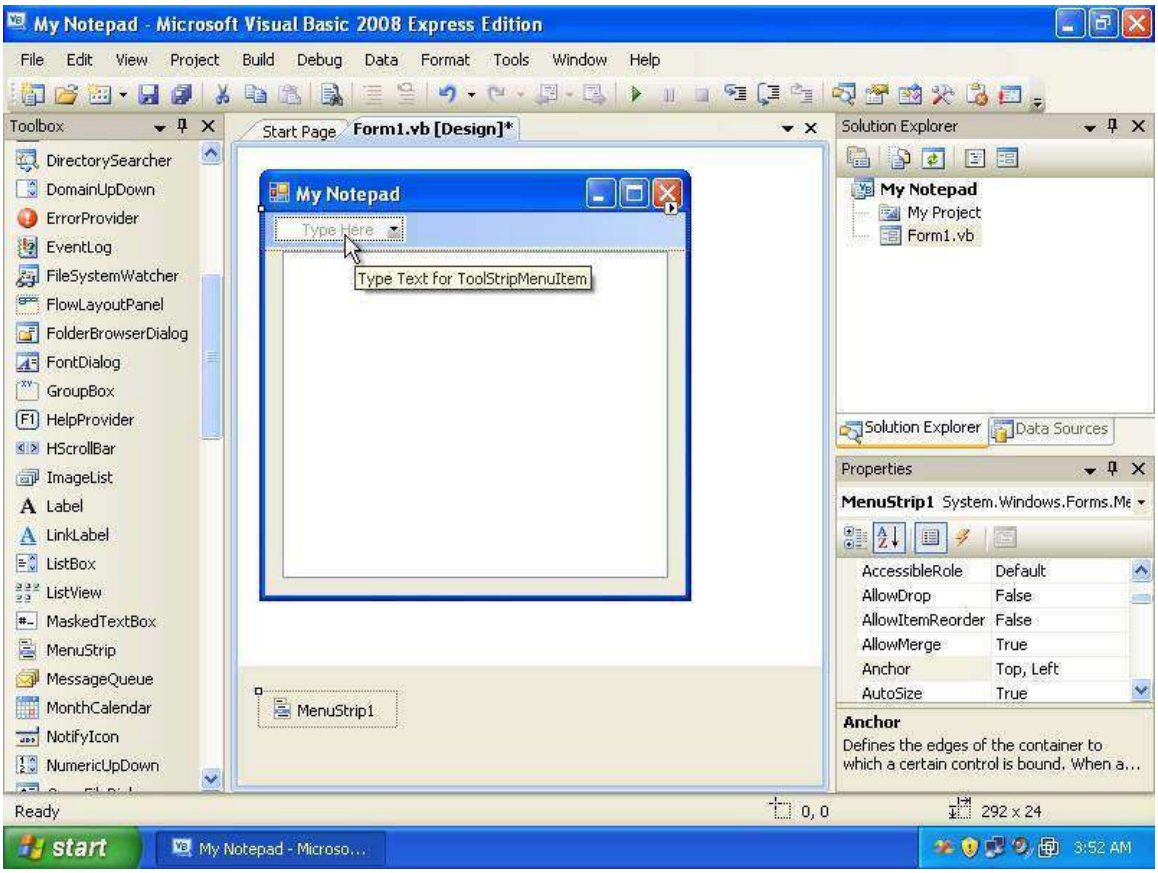
Next try to resize the window. Now you see there is a problem. The text box does not resize itself to match window size.



To solve this select the text box and change the **Anchor** property. This property specifies how should the text box resize itself when its parent container resize itself (the window). The anchor side specifies how far the edge of the control should be from the border of the outer window (or any other control). Specifying the Anchor to be **Top**, **Left**, **Right**, **Bottom** means that whenever the size of the window changes, the text box changes itself so that its sides keeps the same distance from window border. Try changing the anchor settings and test the application and resizing the window.

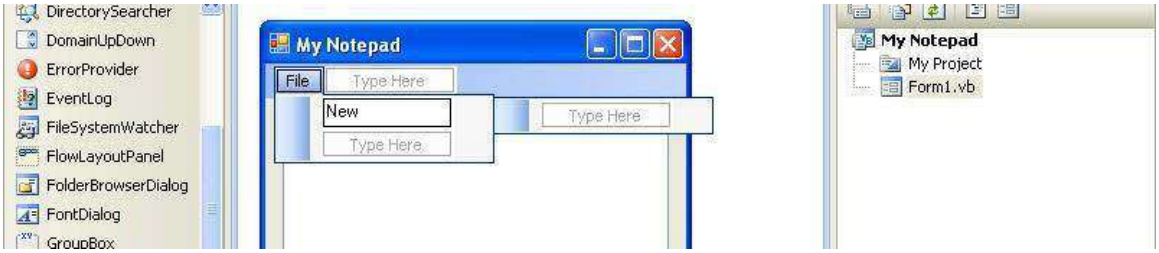


Next we start working with menus. Search for a control called **MenuStrip** and drop it on the window.



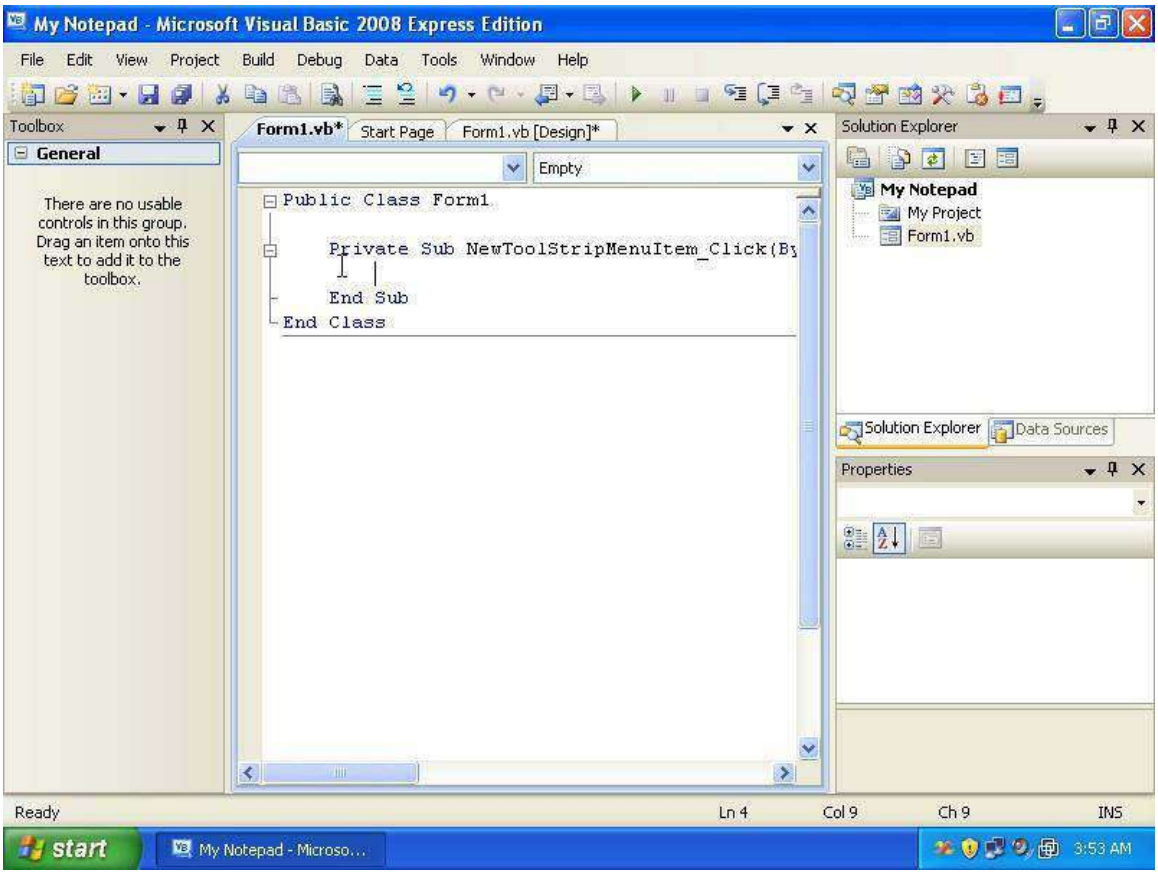
The control appears under the window as **MenuStrip1** which is the name of the control. The actual menu appears on the form itself. You can create the menu quickly by just start typing. Try typing **File.**

Notice that when you write a menu entry, you can extend it horizontally, and vertically.



Under the file menu add the **New**, **Open**, **Save**, and **Close** menu items.

Next you will write the code to handle the events for these menu items. Now in the workspace just double click the **New** menu item.



As with the previous tutorial you get the code editor. Write down the following:

```
TextBox1.Text = ""
```

This will clear the text box and allow you to write next text. Add the following code to the close menu item.

```
End
```

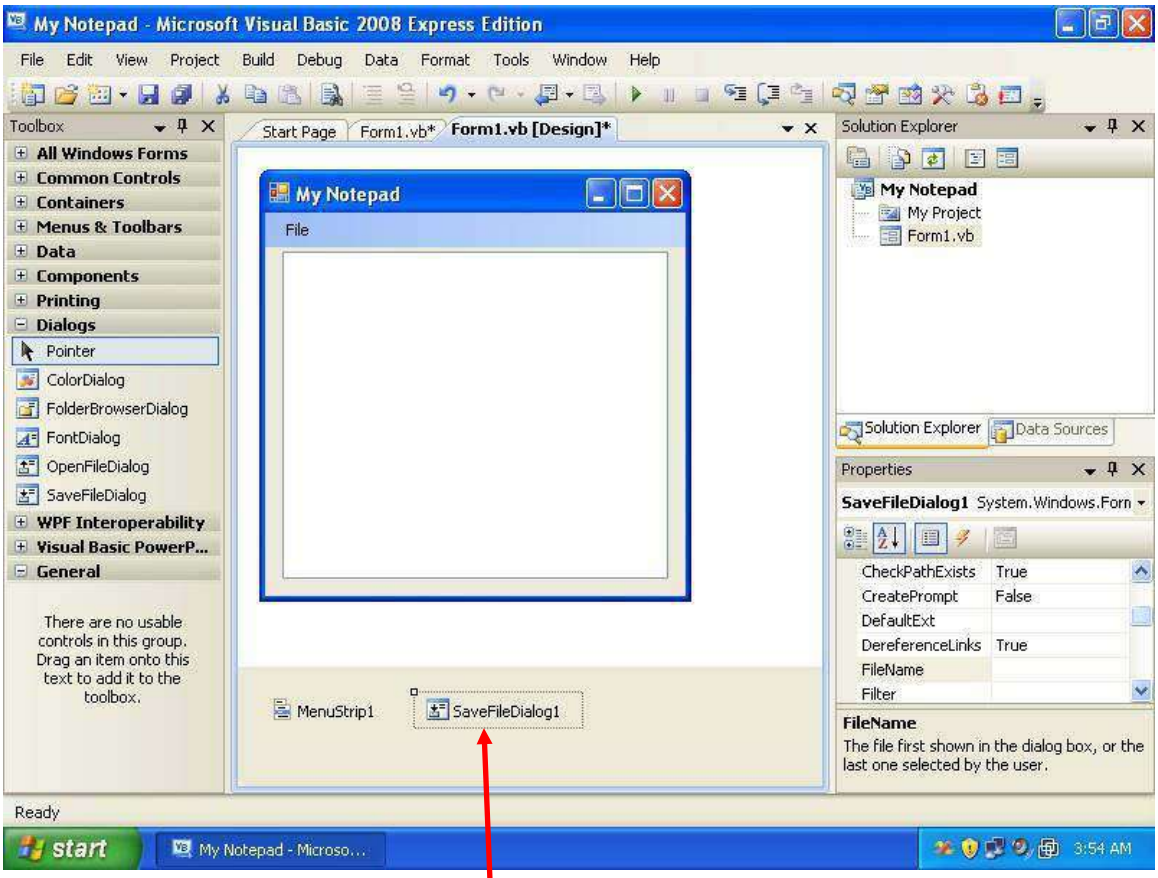
This will close the application. The code should look like this:

```
Private Sub NewToolStripMenuItem_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles NewToolStripMenuItem.Click
      TextBox1.Text = ""
End Sub
```

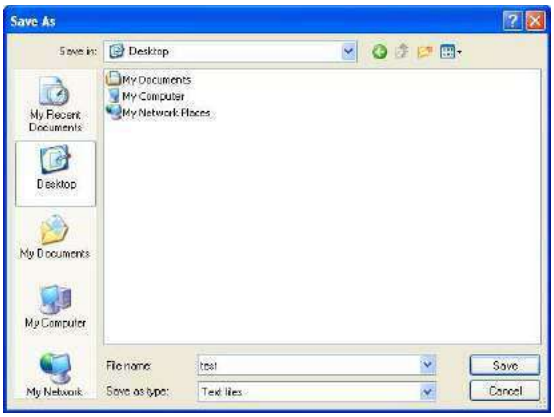
```
Private Sub CloseToolStripMenuItem_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles CloseToolStripMenuItem.Click
        End
End Sub
```

Notice that I haven't change controls names (the menu items name property). You can change the name to a more friendly one as we did in the previous tutorial. Also the NewToolStripMenuItem.Click and CloseToolStripMenuItem.Click are on the same line as Handles .

Run the application, try to write some text, then select File->New. Then try File->Close. Next we see how to save text.



Search for a control called **SaveFileDialog** and drop it on the form. You won't see any visual change to the form itself. You can see there is save file dialog available under the window. This control allows you to specify where to save files on the file system and it looks like the window below:



Let us modify the filter property of the dialog. Click on **SaveFileDialog1** to display its properties. 

Now change the **Filter** property to be like that:

Set it to be: **Text files|*.txt**

This property prevents the user form mistakenly saving the file in formats other than text. The **Text files** part is displayed to the end user, while the ***.txt** is used to filter the files and make sure you only select or overwrite text (ending with .txt) files.

The next step is to write the code to save the text written in your application into the disk. Add the code in the handler of the **Save** menu item by double clicking it then typing:

```
SaveFileDialog1.ShowDialog()
If SaveFileDialog1.FileName = "" Then
      Exit Sub
End If

' this part saves the file
FileSystem.FileOpen(1, SaveFileDialog1.FileName, OpenMode.Output)
FileSystem.Print(1, TextBox1.Text)
FileSystem.FileClose(1)
```

The code is explained as follows:

`SaveFileDialog1.ShowDialog()`: this instructs the application to show the Save dialog on the screen so that you can specify the file name. When you select a name for the file, you can get it from the control using the **FileName** property.

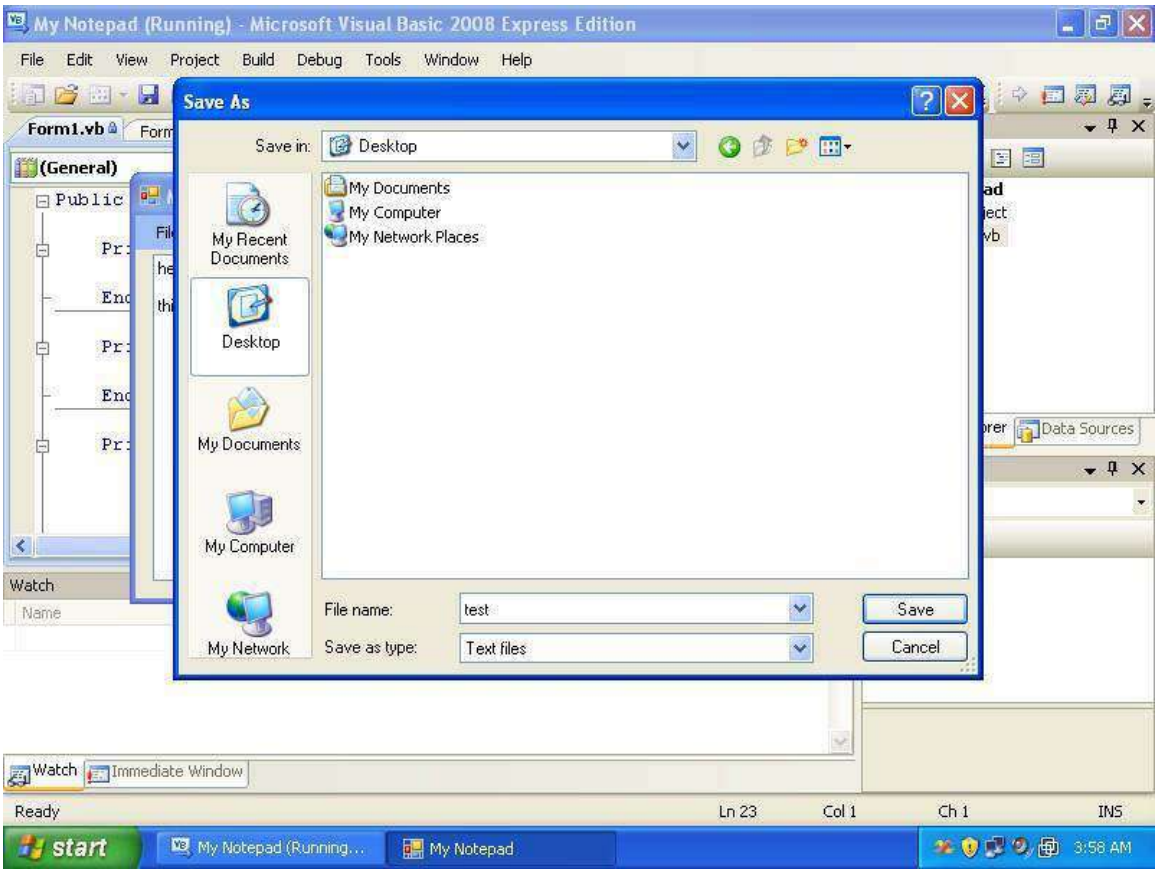
```
If SaveFileDialog1.FileName = "" Then
      Exit Sub
```

```
End If
```

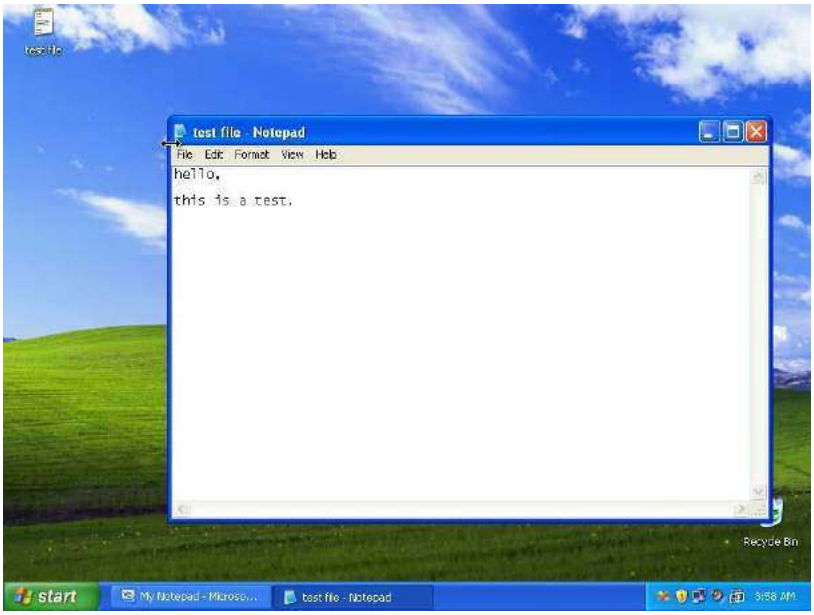
The code above checks if the **FileName** is not specified, in other words if you pressed the cancel button when the dialog is shown, the **FileName** will be empty. So in this case no saving will happen and the execution path of the code exits the subroutine.

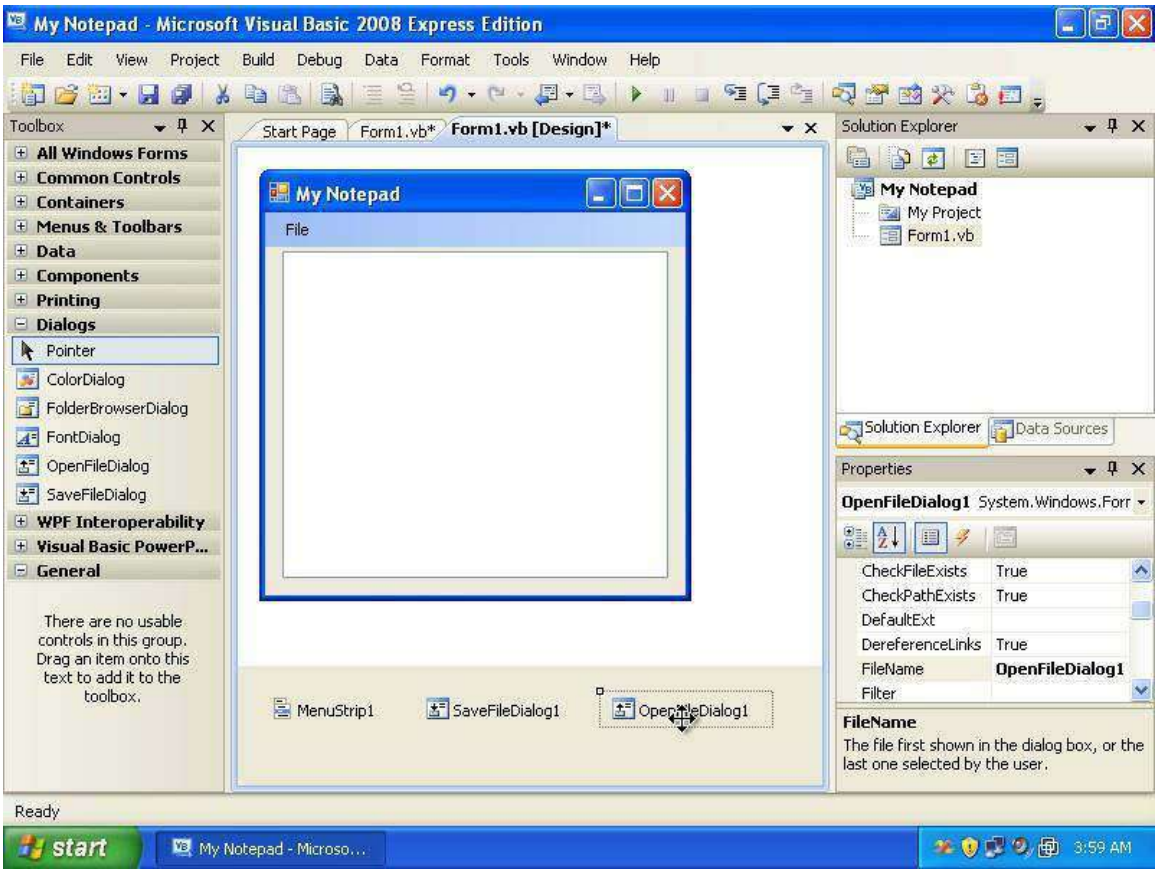
```
FileSystem.FileOpen(1, SaveFileDialog1.FileName, OpenMode.Output)
FileSystem.Print(1, TextBox1.Text)
FileSystem.FileClose(1)
```

This part saves the text into the file. The SaveFileDialog1.FileName property allows you to get the name of the file. TextBox1.Text gets the text from the text box.



Run the application, write some text, and then select **File->save**. Save your file into the desktop. You can open this text file via the standard notepad as shown below:



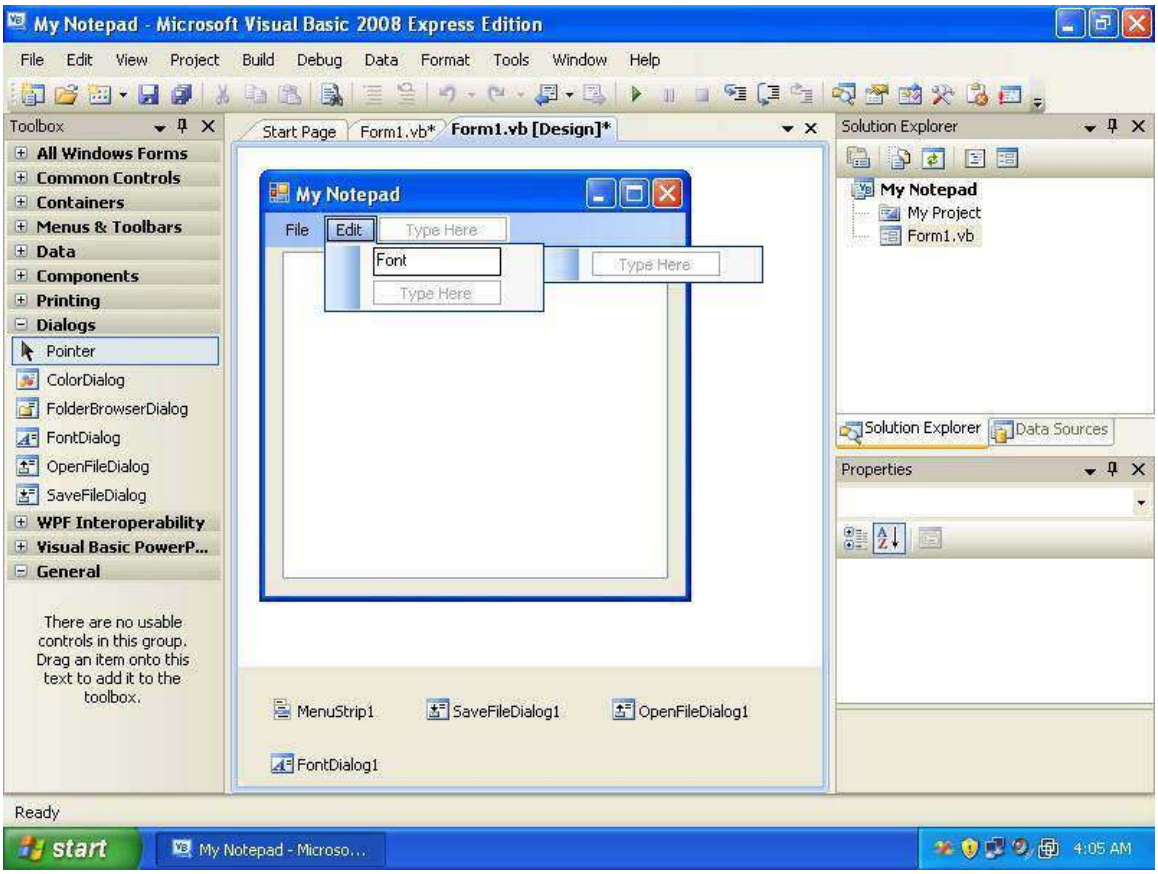


Now we work on the **File->Open** part. Search for the **OpenFileDialog** control and drop it on the form. Change **FileName** property and remove all the text from it. And change the filter property to: **Text files|*.txt**. and finally go to the **File->Open** event handler by double clicking the Open menu item, And then add the following code:

```
OpenFileDialog1.ShowDialog()
If OpenFileDialog1.FileName = "" Then
      Exit Sub
End If

' this part loads the file
Dim Tmp As String
Tmp = ""
FileSystem.FileOpen(1, OpenFileDialog1.FileName, OpenMode.Input)
Do While Not FileSystem.EOF(1)
      Tmp = Tmp & FileSystem.LineInput(1)
      If Not FileSystem.EOF(1) Then
            Tmp = Tmp & Chr(13) & Chr(10)
      End If
```

```
Loop
FileSystem.FileClose(1)
TextBox1.Text = Tmp
```

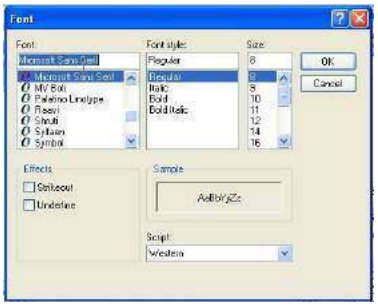
The **OpenFileDialog** works very similar to the **SaveFileDialog** so there is no much to explain about it. After testing this part, we will work with the font. We want to add the ability to change the size and type of the font in the text box to make it easier to read. So add menu entries **Edit**, and **Font**, then add the **FontDialog** control.



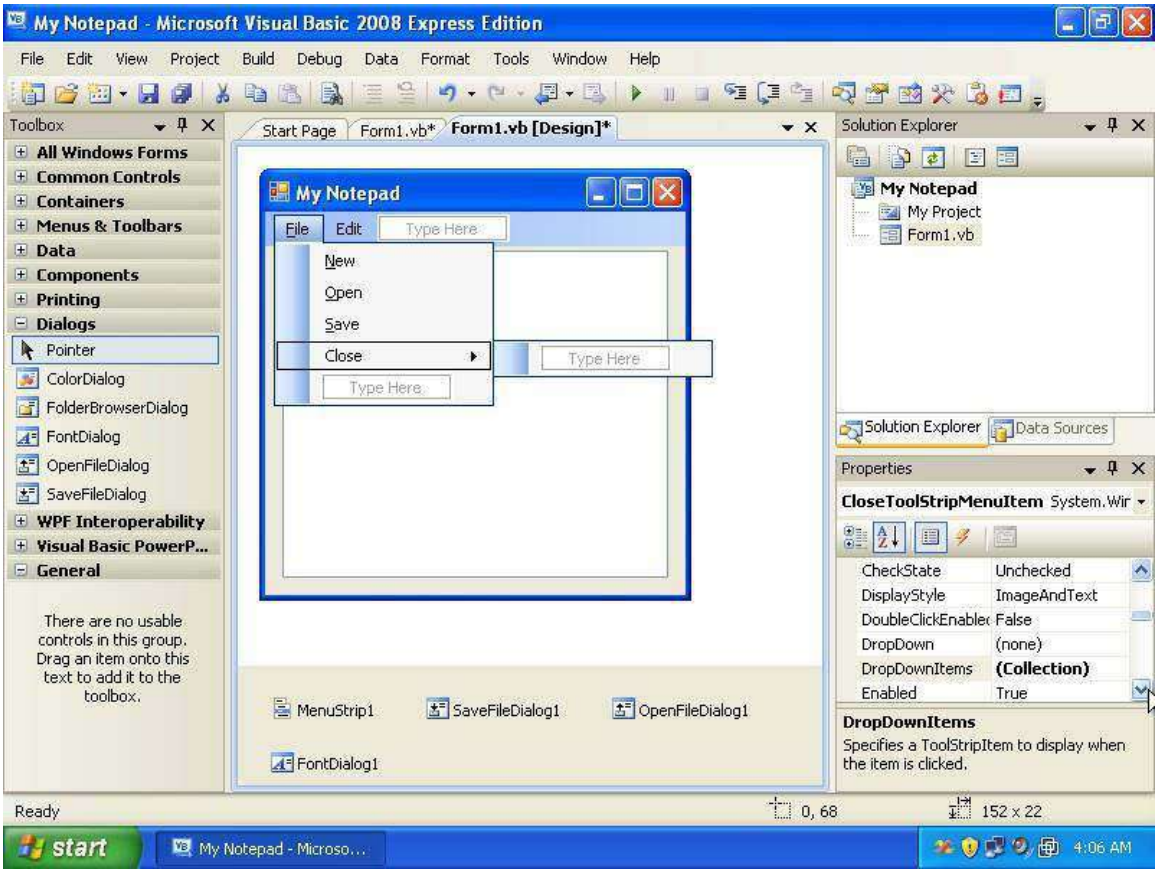
Next double click the **Font** menu item to built its handler.

```
FontDialog1.ShowDialog()
TextBox1.Font = FontDialog1.Font
```

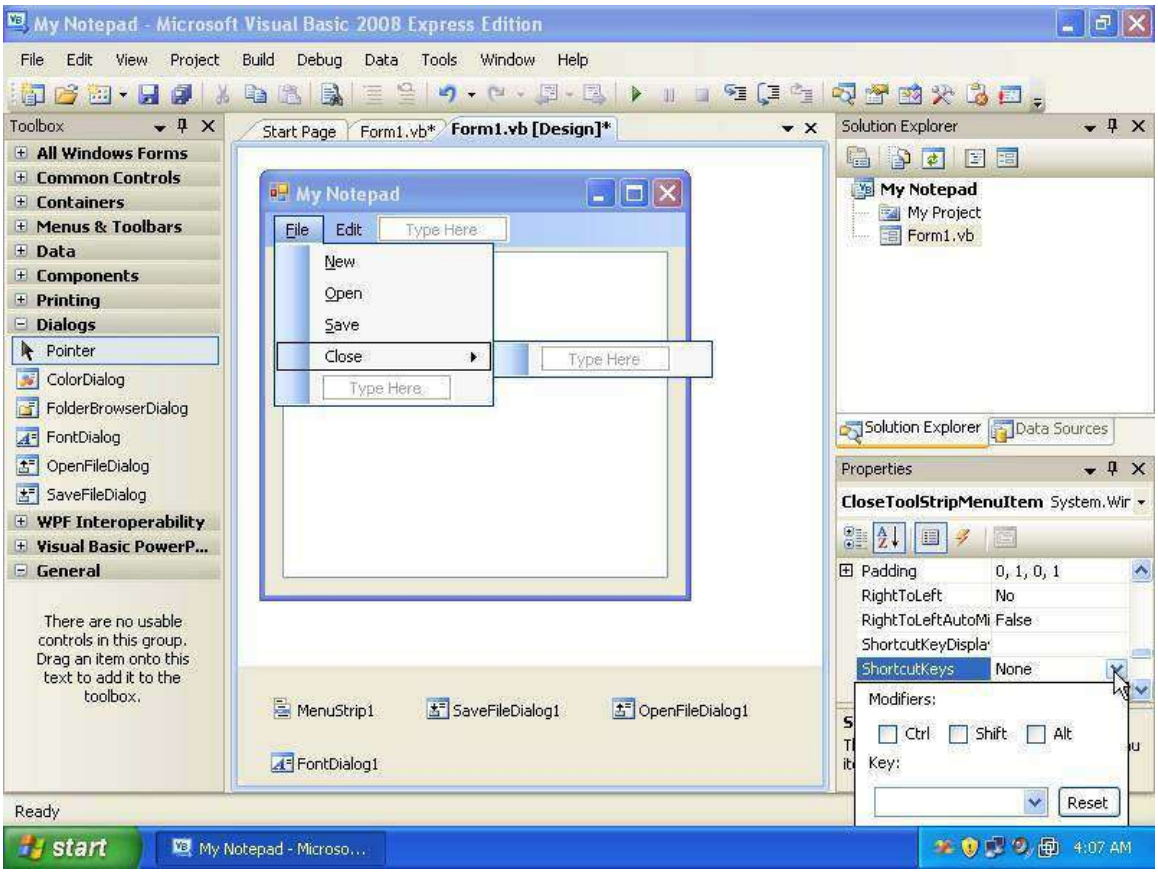
Try to run the application and select the Font menu item



Now let us improve out menu now. Right now the menu cannot be accessed using the keyboard. You can make it accessible using the Alt key and some letter. For example click once on the file menu item. Now you can change the text displayed on the menu item. Modify it to be &File. This will have the effect of adding the ability to access the menu item using the Alt+F combination. Perform the same operation for other menu items to be &New, &Open, &Close, &Save, &Edit. The letter after the & symbol is always the access key.



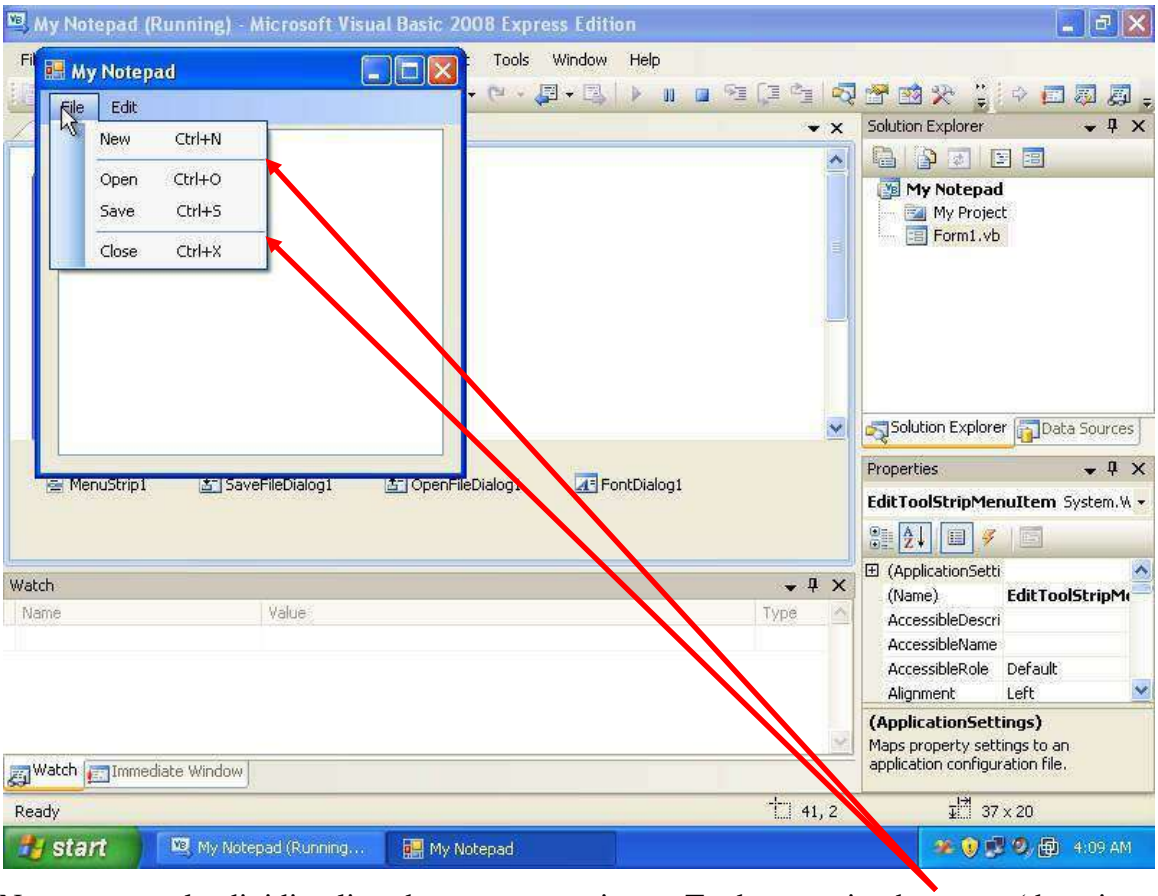
Try to run the application, then pressing Alt+F, then O to show the open file dialog as a test to see if it works.



Next we will add shortcut keys. This is easy, click the menu item once to show its properties. Change the **ShortcutKeys**, by marking the **Ctrl** check box and selecting the **X** button for the **Close** menu item.

Repeat the same step for other menu items

Run the application and press **Ctrl+O** and you will see the **Open File Dialog** directly.



Next try to make dividing lines between menu items. To do so, write the text – (the minus sign) in the text part of the menu item. Notice that you can drag and change the position of the menu items, so try to position those dividing lines to be something similar to the above.

Now we start defining the **Copy**, **Cut**, and **Paste** commands. So, add these menu items under **Edit**, and write the code for each one of these:

1- for the copy

```
Clipboard.SetText(TextBox1.SelectedText)
```

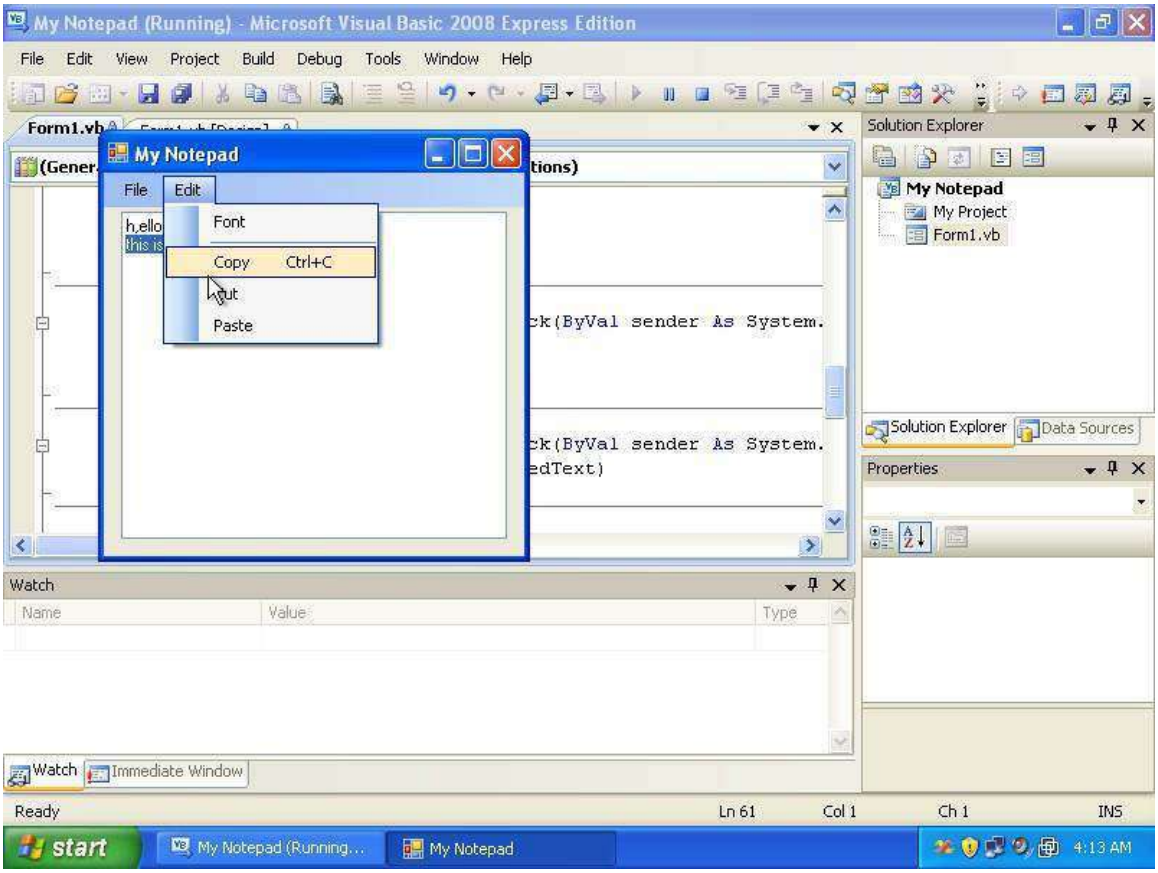
2- for the cut

```
Clipboard.SetText(TextBox1.SelectedText)
TextBox1.SelectedText = ""
```

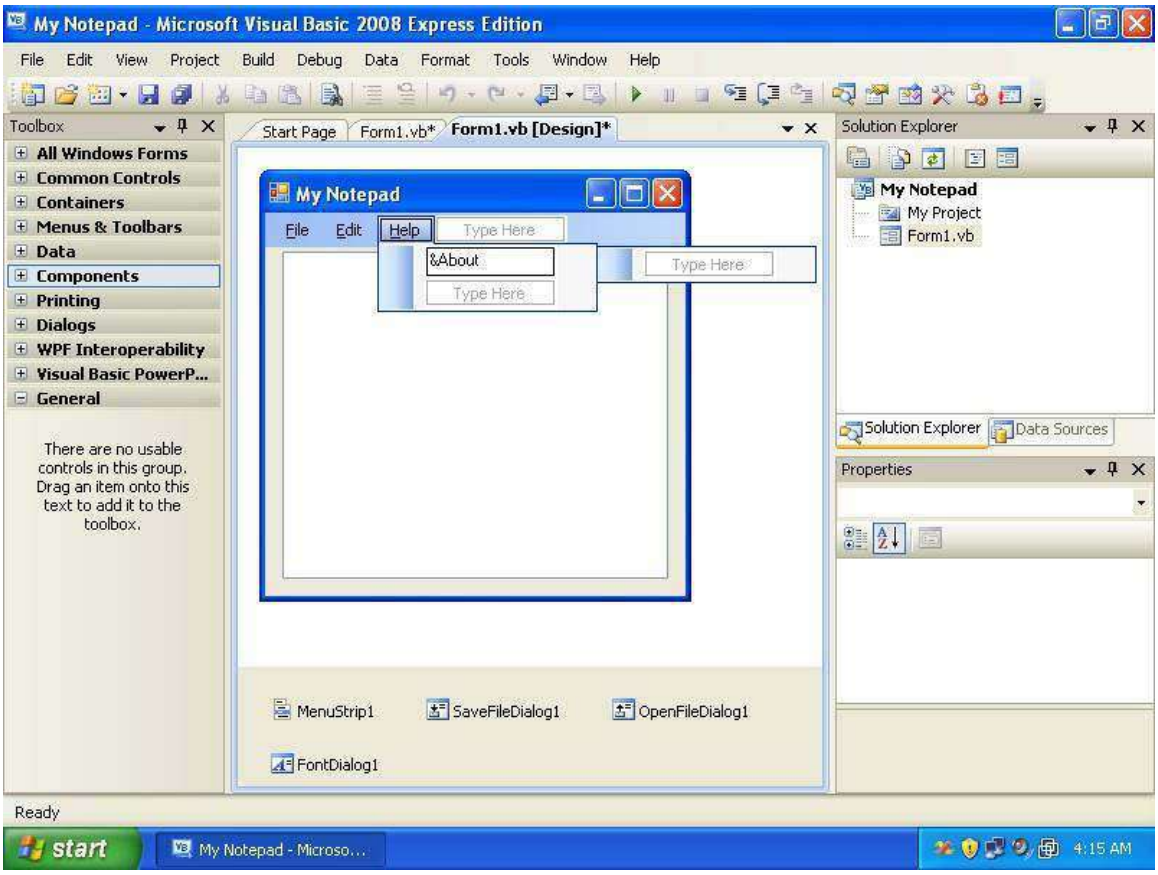
3- for the paste

```
TextBox1.SelectedText = Clipboard.GetText
```

Run the application and test the copy, cut, and paste.



Now let us just add an **About** menu item, under the **help** menu item.



Add the following code in the **About** menu item.

```
MsgBox("This application is a test of making a notepad application out
of VB.NET", MsgBoxStyle.OkOnly, "About My Notepad")
```

The final code should look like this:

```
Public Class Form1

    Private Sub NewToolStripMenuItem_Click(ByVal sender As
          System.Object, ByVal e As System.EventArgs) Handles _
          NewToolStripMenuItem.Click
        TextBox1.Text = ""
    End Sub
```

```
    Private Sub CloseToolStripMenuItem_Click(ByVal sender As
          System.Object, ByVal e As System.EventArgs) Handles _
          CloseToolStripMenuItem.Click
        End
    End Sub

    Private Sub SaveToolStripMenuItem_Click(ByVal sender As
          System.Object, ByVal e As System.EventArgs) Handles _
          SaveToolStripMenuItem.Click
        SaveFileDialog1.ShowDialog()
        If SaveFileDialog1.FileName = "" Then
            Exit Sub
        End If

        ' this part saves the file
        FileSystem.FileOpen(1, SaveFileDialog1.FileName,
OpenMode.Output)
        FileSystem.Print(1, TextBox1.Text)
        FileSystem.FileClose(1)
    End Sub

    Private Sub OpenToolStripMenuItem_Click(ByVal sender As
          System.Object, ByVal e As System.EventArgs) Handles _
          OpenToolStripMenuItem.Click
        OpenFileDialog1.ShowDialog()
        If OpenFileDialog1.FileName = "" Then
            Exit Sub
        End If

        ' this part loads the file
        Dim Tmp As String
        Tmp = ""
        FileSystem.FileOpen(1, OpenFileDialog1.FileName, OpenMode.Input)
        Do While Not FileSystem.EOF(1)
            Tmp = Tmp & FileSystem.LineInput(1)
            If Not FileSystem.EOF(1) Then
                Tmp = Tmp & Chr(13) & Chr(10)
            End If
        Loop
        FileSystem.FileClose(1)
        TextBox1.Text = Tmp
    End Sub
```

```
    Private Sub FontToolStripMenuItem_Click(ByVal sender As
          System.Object, ByVal e As System.EventArgs) Handles _
          FontToolStripMenuItem.Click
        FontDialog1.ShowDialog()
        TextBox1.Font = FontDialog1.Font
    End Sub

    Private Sub CopyToolStripMenuItem_Click(ByVal sender As
          System.Object, ByVal e As System.EventArgs) Handles _
          CopyToolStripMenuItem.Click
        Clipboard.SetText(TextBox1.SelectedText)
    End Sub

    Private Sub CutToolStripMenuItem_Click(ByVal sender As
          System.Object, ByVal e As System.EventArgs) Handles _
          CutToolStripMenuItem.Click
        Clipboard.SetText(TextBox1.SelectedText)
        TextBox1.SelectedText = ""
    End Sub

    Private Sub PasteToolStripMenuItem_Click(ByVal sender As
          System.Object, ByVal e As System.EventArgs) Handles _
          PasteToolStripMenuItem.Click
        TextBox1.SelectedText = Clipboard.GetText
    End Sub

    Private Sub AboutToolStripMenuItem_Click(ByVal sender As
           System.Object, ByVal e As System.EventArgs) Handles _
           AboutToolStripMenuItem.Click
      MsgBox("This application is a test of making a notepad application
           out of VB.NET", MsgBoxStyle.OkOnly, _ "About My Notepad")
    End Sub
End Class
```

Notice that some lines are too long so I divided them to multiple lines. In visual basic, writing a command on multiple line requires you to add the underscore symbol (_) to tell the compiler that these two (or more) lines are actually one long line. Try to run the application and test it.

Chapter 5: Understanding variables

Understanding variables

When a computer processes information or data, that data should be placed in its memory first, then it performs different operations on that. You application as well should process the information in memory. To accomplish that the you allocate memory by defining variables. Simply, a variable is small piece of memory that allows the program to process data within it.

To define a variable in VB.NET you use the following format:

```
Dim MyInt As Integer
```

Where the `Dim` tells the computer that you are going to define a variable. Next you write variable name (in this case it is `MyInt`) and finally the data type `As Integer` which tells the computer you are going to perform integer operations on the data in that variables.

So, in order to try out integer variables, create a simple windows application, and place a button on your form, and modify its event so that it looks something similar to this:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
        System.EventArgs)  Handles Button1.Click
    Dim MyInt As Integer
    MyInt = 55 + 4
    MsgBox(MyInt)
End Sub
```

Try this code out… you see that `MyInt` stores the result of the operation. You can place any type of operation like +,-,*,/ and more… but for now let us try to create another variable and use it with this one. Modify the code to be something like this:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
        System.EventArgs) Handles Button1.Click
    Dim MyInt As Integer
    Dim A As Integer, B As Integer
    A = 10
    B = 30
    MyInt = A + B + 2
    MsgBox(MyInt)
```

```
End Sub
```

In this case we are using 3 variables (A,B,MyInt). We compute using the values inside the A,B and store the result in MyInt. So basically any of the following statements is valid:

`MyInt = 22`	places value 22 in MyInt
`MyInt = A`	copies the value of A into MyInt (10)
`MyInt = B`	copies the value of B into MyInt (30)
`MyInt = A + B`	adds A,B and places the result in MyInt (40)
`MyInt = A - B`	subtructs B from A (-20)
`MyInt = A + A + B + 2`	adds double value of A, value of B and 2 to the total (52)
`MyInt = B*A-A`	multiply A by B then subtructs A (290)
`MyInt = A/2`	divides A by 2 (10)
`MyInt = B/A`	divides B by A (3)
`MyInt = MyInt + 1`	gets the value of MyInt (usually 0) and adds one to that (total is 1)

and you can come up with any form of statement that helps you solve your problem.

Now we start understanding what is the data type. Each variable can be one data type. The data type tells the computer what you are going to process in this variable. For example **Integer** means you are working with whole numbers (like 30,40,55) and not fractions. So if you try out:

```
MyInt = 3.4
```

What you get is the value 3. That is because this data type can not store floating point numbers. To solve this issue you should use **Singe**. To make things clear try to execute the following code:

```
Dim I As Integer
I = 22 / 7
MsgBox("the PI (int) :" & I)
```

```
Dim S As Single
S = 22 / 7
MsgBox("the PI (single) :" & S)
```

When you run this code, you will see that it give you 3 as the value of PI (the integer case), then it gives you a value of 3.142857 for PI (the single case).

So why do you use integers and singles? Why not use single all the time? The answer is performance and storage. The computer processes integer operations faster than that of floating point numbers. And also some data types takes less storage than others. Because of that if you are going to develop an application that performs lots integer operations it makes sense to use integers to speed things up. In the end you select variable types depending on the nature of your problem.

There are similar data types to Integer and Single. These are Long (to store integers) and Double(to store floating point numbers). The difference between these and the previous ones is that they can represent wider range of numbers. To demonstrate this try out the following code:

```
MsgBox("integer " & Integer.MaxValue)
MsgBox("long " & Long.MaxValue)
MsgBox("single " & Single.MaxValue)
MsgBox("double " & Double.MaxValue)
```

The Integer.MaxValue gets the maximum value that an Integer can store. The same is true for other data types. Of course better representation requires more memory and also more processing time. Another example to demonstrate this is by calculating the value of PI using Double and Single. Try out the following code:

```
Dim I As Integer
I = 22 / 7
MsgBox("the PI (int) :" & I)

Dim S As Single
S = 22 / 7
MsgBox("the PI (single) :" & S)
```

```
Dim D As Double
D = 22 / 7
MsgBox("the PI (double) :" & D)
```

I did not write the Long data type here, but you can try it. It gives the same result as that of the integer case because Long data type can only store non-floating point values.

Another type of variables is the ones used to store complete statements, the String data type.

```
Dim Str1 As String
Dim Str2 As String

Str1 = "hello"
Str2 = " my friend"

MsgBox(Str1)
MsgBox(Str2)
```

As you can see, this data type stores letters and symbols. And actually you can do a number of operations on these. The simplest is to use the & operator to combine two strings.

```
Dim Str1 As String
Dim Str2 As String
Dim Str3 As String

Str1 = "hello"
Str2 = " my friend"
Str3 = Str1 & Str2

MsgBox(Str3)
```

You can see that Str3 now holds the complete statement “hello my friend”. More into string data type will come with time, but here we are focused on the very basics of variables.

Another important data type is the Boolean data type. This one is used to evaluate logic operations and it can only store True and False.

```
Dim B1 As Boolean
Dim B2 As Boolean
Dim B3 As Boolean
Dim B4 As Boolean

B1 = True
B2 = False
B3 = 88 > 10
B4 = "asmith" > "john"
MsgBox("b1 is:" & B1)
MsgBox("b2 is:" & B2)
MsgBox("88 > 10 is: " & B3)
MsgBox("asmith comes after john is:" & B4)
```

As you can see above B3 will check if 8 is greater than 10, and if so, it stores the value True, otherwise it stores False. B4 here shows how you can compare two strings. It checks to see if **asmith** comes alphabetically after **john** (which is the meaning of > sign in string comparison), and obviously this is not correct, so the value of B4 is false.

Finally you have the Date data type which is used to store the time and date. You can test it using the example below:

```
Dim D1 As Date
Dim D2 As Date
Dim D3 As Date
Dim D4 As Date
D1 = Now
D2 = Now.Date
D3 = "8:10:22 AM"
D4 = "2009-03-01"
MsgBox(D1)
MsgBox(D2)
MsgBox(D3)
MsgBox(D4)
```

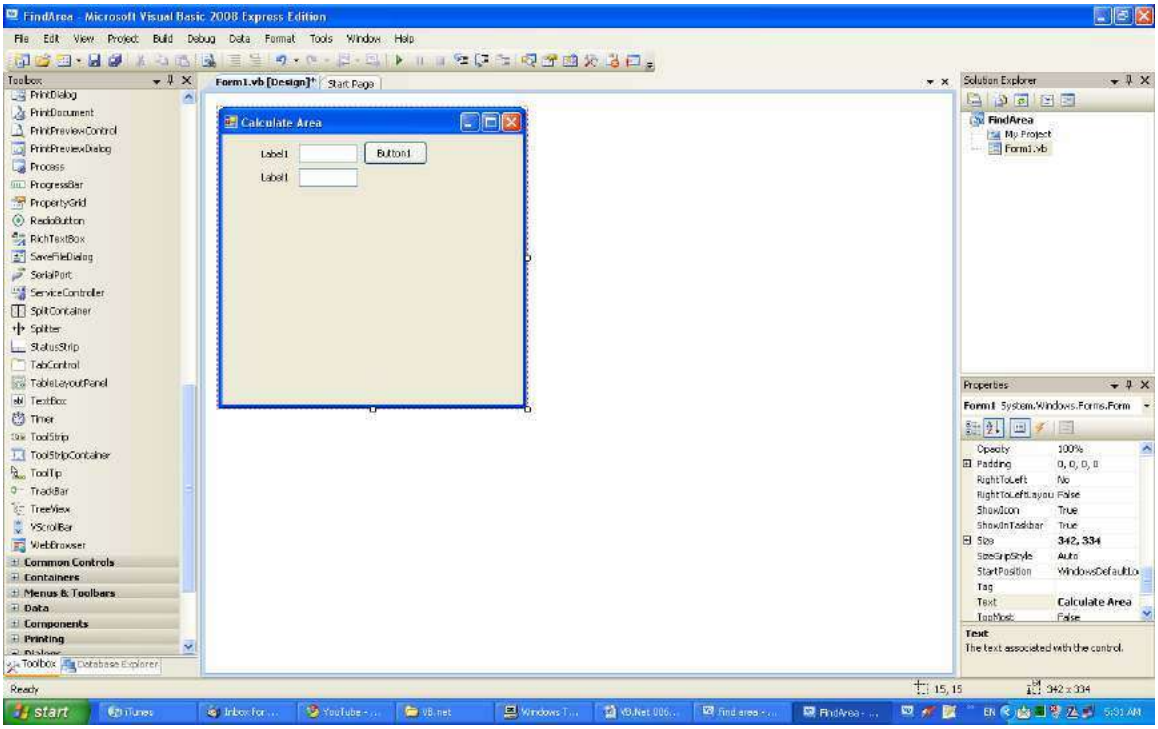
The **Now** in the above example is used to get current system date and time. **Now.Date** is used to get system date alone.

Chapter 6: Variables again, group box, list box

Variables again, group box, list box

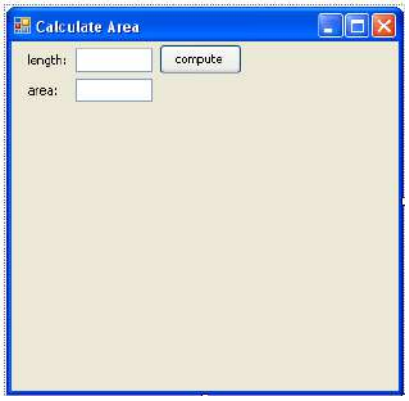
This chapter is about using Group boxes, and little about variables and list boxes. We will create a small application to find the area of squares, rectangles, and triangles. So let us proceed by creating a new windows application project, name it "find area" and save it.

Next set the form title to "Calculate Area", I assume now you already know how to do that. Next place a two labels, two text boxes, and a button on the form to look something like this:

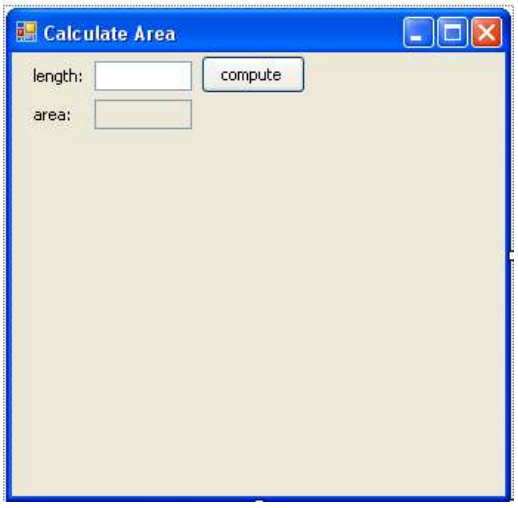


Now change the first label to be "length:", second label to be "area:", and the button text to be "compute". (to do that just click the control and change the text property".

You should get something like this:



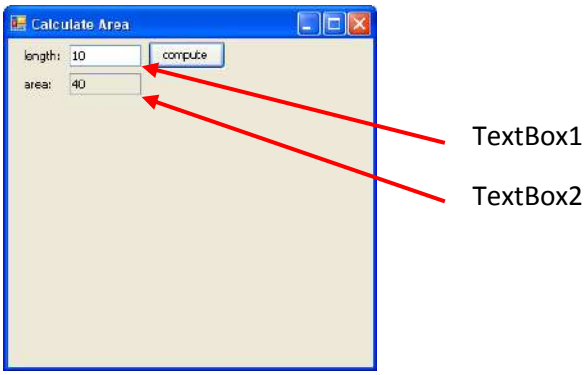
The user enters the length of the square in the first text box, and presses the compute button, and he/she should get the area in the second text box. The second text box should not be modified by the user. To do so, change the ReadOnly property of the second text box to be "true".



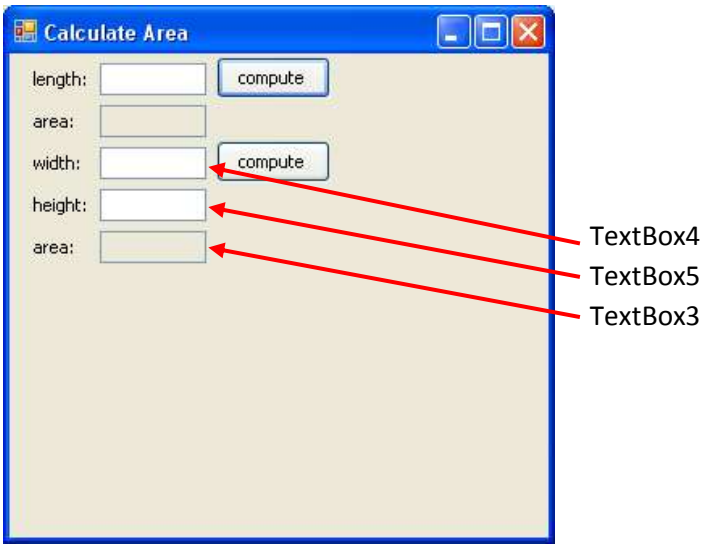
To write the code that gets the length, calculate the area, and display it, first check the name of the text boxes because you will need these. Next double click the compute button to write its event.

```
Dim L As Double      ' define the length variable
Dim A As Double      ' define the area variable

L = TextBox1.Text    ' get the length from the window.
A = L * L            ' calculate the area

TextBox2.Text = A    ' display the area in the second
                     ' text box
```

Keep in mind that TextBox1, and TextBox2 are the names of the controls used to get the data and display the result. And also remember you can change them if you want to. Now run the application, enter a value of 10 and press compute.



Chang the length and press compute again and see the new results. Next we will do something similar for the rectangle. Place three labels, three text boxes, and a command button to be something like this:

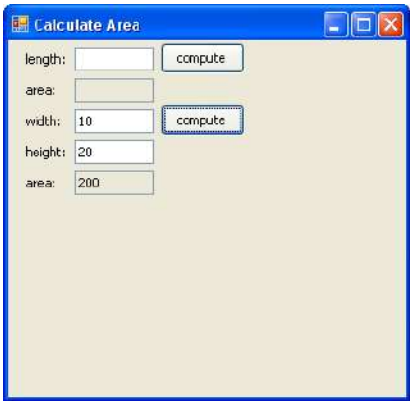


Note that control names here are just the way I got them, you might get the order differently, or you might want to rename the controls. Next double click the compute button for the rectangle and write the following code.

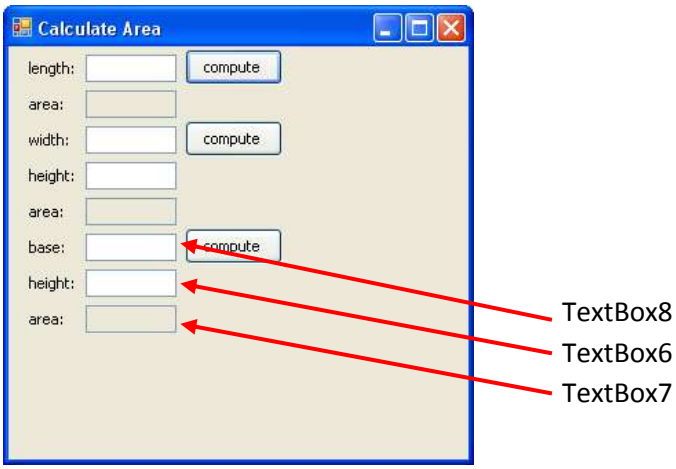
```
Dim W As Double      ' define the width variable
Dim h As Double      ' define the height variable
Dim A As Double      ' define the area variable
```

```
W = TextBox4.Text    ' get the width
h = TextBox5.Text    ' get the height
A = W * h            ' compute the area

TextBox3.Text = A    ' dsplay the area
```

Now run the application, and test it.



The last part is the triangle. Repeat what you did with the rectangle and you should get something similar to this:

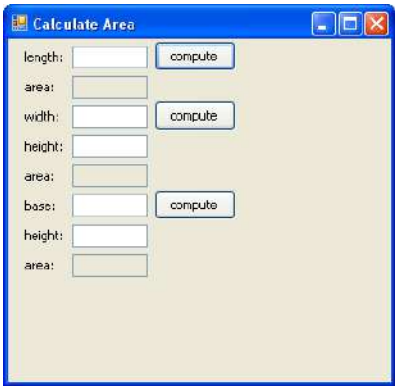


And next set the code to compute the area to be:

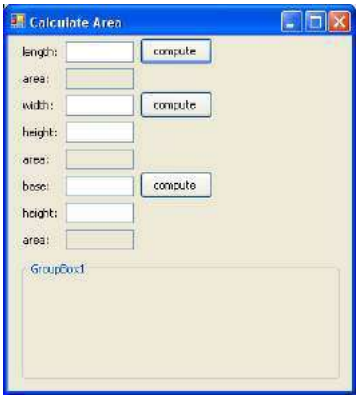
```
Dim base As Double              ' define the base
Dim Height As Double            ' define the height
Dim Area As Double              ' define the area

base = TextBox8.Text            ' get the base
Height = TextBox6.Text          ' get the height
```

```
Area = 0.5 * base * Height       ' compute the area
                                 ' (1/2 x base x height)

TextBox7.Text = Area             ' display the result.
```

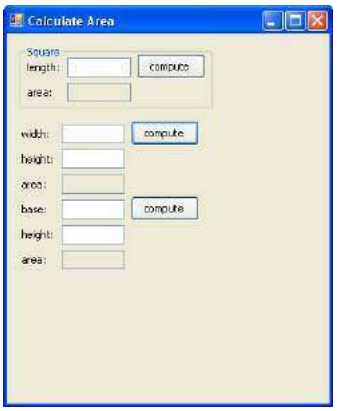
and check the code. It works. Now look at our window.



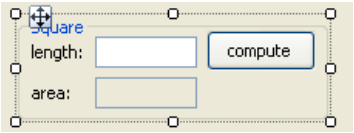
It does not look good actually and a little bit confusing because there are three compute buttons, and the fields are mixed. The solution to this is to use GroupBox control. Now search with the controls and add a GroupBox control on the form.



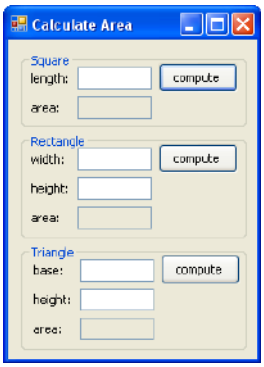
Next select the labels, textboxes and the button that is used for the square, and drag them so that they fit inside the group box. Click then on the group box and set its text property to be square. You should get something similar to this:



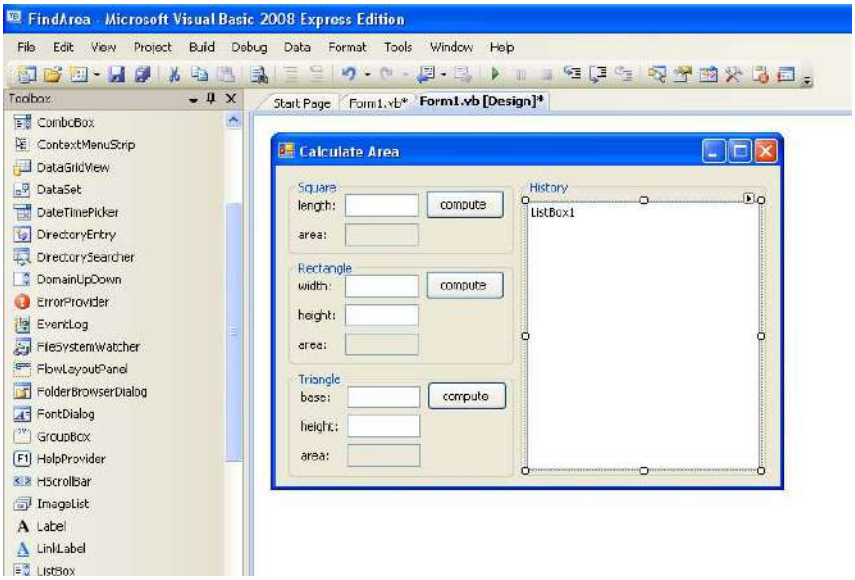
Notice that you might need to move the controls a little bit to make them fit. The group box can be moved using the arrows symbol that appears when you click inside it.



Repeat the same thing for the rectangle and triangle to get something like this:



As you can see group box allows easy movement of controls on the form at design time and also gives a better view of your application. Now we start working with list box. It is used to view a list (which is no surprise). What we want to do here is displaying each area being computed on the right side in such a way the previous results and computations are displayed as well. To do so, add a group box to the right of your window and put inside it a listbox control. You should have something like this:



The ListBox1 that appears here does not appear while the application is running. What you see now is the name of the listbox control. Now we modify the code of the square so that we add the area computed into the list box. Double click on the compute button of the square, and modify its code to be like this:

```
Dim L As Double      ' define the length variable
Dim A As Double      ' define the area variable

L = TextBox1.Text    ' get the length from the window.
A = 4 * L            ' calculate the area

TextBox2.Text = A    ' display the area in the second text
                     ' box

ListBox1.Items.Add("the area of the square is:" & A)
                     ' add the item
```

ListBox1 is used to communicate with the list, **.Items** is used to access the list of items that it is displaying (which is at the beginning of execution is empty), **.Add** is used to add the information or message into the Items of the list box. "the area of the square is:" & A is evaluated first, and then the result is added into the list box items.

For the rectangle, modify its compute button to be something like this:

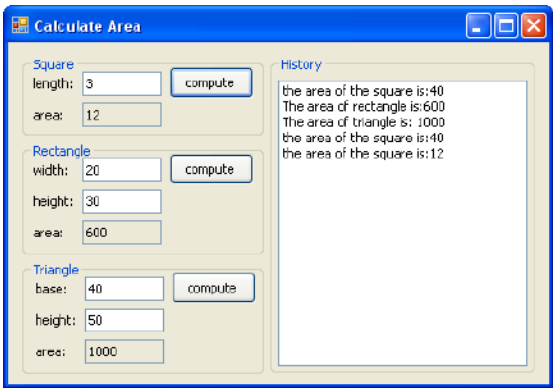
```
Dim W As Double      ' define the width variable
```

```
Dim h As Double      ' define the height variable
Dim A As Double      ' define the area variable

W = TextBox4.Text    ' get the width
h = TextBox5.Text    ' get the height
A = W * h            ' compute the area

TextBox3.Text = A    ' dsplay the area

Dim S1 As String
Dim S2 As String

S1 = "The area of rectangle is:" ' the message
S2 = S1 & A              ' write the message then the area, and
                         ' store the new message in S2

ListBox1.Items.Add(S2)   ' add the message to the list box
```

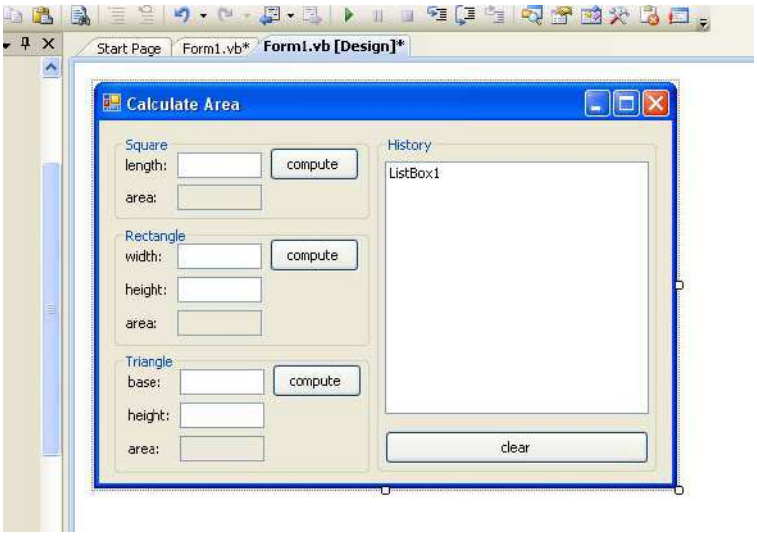
The same method to add item to the list box is called, but the way we generate the message is a little bit different. Finally the rectangle code should be:

```
Dim base As Double              ' define the base
Dim Height As Double            ' define the height
Dim Area As Double              ' define the area

base = TextBox8.Text            ' get the base
Height = TextBox6.Text          ' get the height

Area = 0.5 * base * Height      ' compute the area (1/2 x base x height)

TextBox7.Text = Area            ' display the result.

Dim Str As String
Str = "The area of triangle is: " & Area    ' generate the whole message
                                            ' directly
ListBox1.Items.Add(Str)         ' add the message to the list
```

Now you see also that the message is generated in a different way again. So you can choose the way that best suits you. Now run the application and calculate few areas.



You should get something similar to this at run time. You can compute more than once and see that the list box keeps adding new results. Next we see how to make a clear button to clear the content on the form (for the listbox and the textboxes). Simply add a button on the form, you should get something similar to this:



And add the following code in the clear button's event:

```
TextBox1.Text = ""          ' clear all the text boxes
TextBox2.Text = ""
TextBox3.Text = ""
TextBox4.Text = ""
TextBox5.Text = ""
TextBox6.Text = ""
TextBox7.Text = ""
TextBox8.Text = ""

ListBox1.Items.Clear()      ' clear the list box
```

As you can see text boxes are cleared by setting their text property to empty string “”, while listboxes you just give a Clear command for them. Run the application, compute a number of areas, and finally hit clear and see how the listbox and all textboxes are cleared from the text.

So basically we worked just a little bit with variables, saw how to work with groupbox, and saw the basic operation of listboxes. More details will be given into listboxes, but for now we need to focus more on the programming aspect of the language.

Chapter 7: IF statement

IF statement

In VB.NET there are many forms for the IF statement. They all work by evaluating some expression and if the expression is correct (evaluated to true) then the code within the IF block is executed. Now check out the first simple form of IF statement

```
If expression Then
    Statement
    Statement
    …
End If
```

The expression here is logical one. For example A>10, A<99, B>=A and so on. If the expression is correct, the statements inside the IF block get executed. The statement could be any valid VB.NET statement (even another IF statement). Now here is an example of the IF statement that always get executed:

```
If 10 < 100 Then
    ' display a friendly message
    MsgBox("You must see this message")
End If
```

Since 10 is always smaller than 100 the condition always evaluates to true and you will always see the message. Now change it to be:

```
If 10 > 100 Then
    ' display a friendly message
    MsgBox("You must see this message")
End If
```

The code within the block will never get executed. Now start a new project, put a button on the form and go to its event handler and add the following code:

```
Dim A As Integer
Dim B As Integer

A = InputBox("enter the value of A")
B = InputBox("enter the value of B")
```

```
If A > B Then
    MsgBox("A is greater than B")
End If

If A < B Then
    MsgBox("A is smaller than B")
End If

If A = B Then
    MsgBox("A is equal to B")
End If
```

The InputBox is a function that reads a value from the keyboard. So the program reads two numbers and check their status. Run the program and try different values for A and B to see how it works. Also debug the program (by pressing F10 to execute one statement at a time) and see how the code get executed internally.

Another variation of the IF statement is the IF … ELSE. It has he following format:

```
If expression Then
    Statement
    Statement
    …
Else
    Statement
    Statement
    …
End If
```

The statements in black get executed when the expression is true, while the statements in red are ignored. However if the expression is evaluated to false then the statements in black are ignored while the statements in red are executed. To see how it works consider the following example:

```
If 10 > 100 Then
    ' this message is never displayed
    MsgBox("10 is greater than 100")
Else
    ' this message is always displayed
```

```
    MsgBox("10 is smaller than 100, what a surprise!!!")
End If
```

The last form of the IF statement is the IF…ELSEIF… statement. Think of it as a multiple if statements combined into one. The form is as follows:

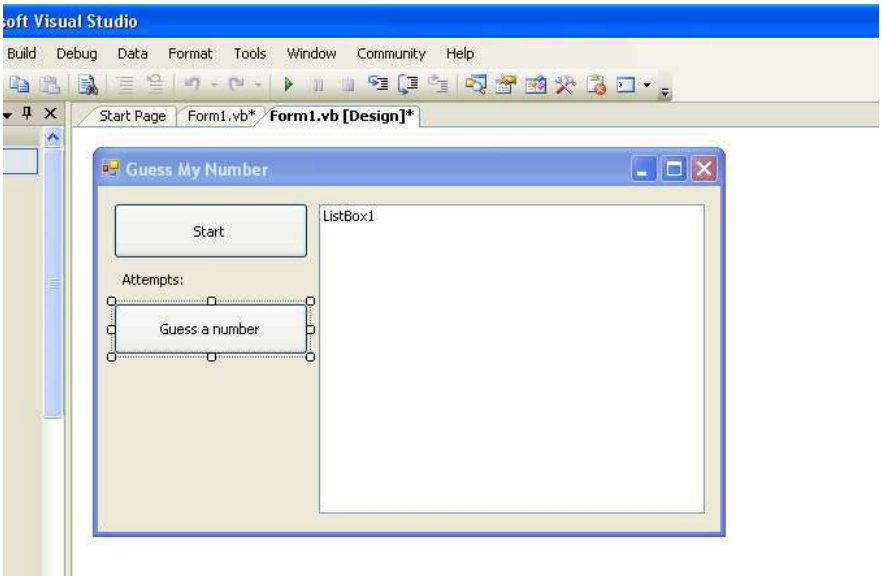
```
If expression1 Then
    Statement
    Statement
    …
ElseIf expression2 Then
    Statement
    Statement
    …
ElseIf expression3 Then
    Statement
    Statement
    …
Else
    Statement
    Statement
    …
End If
```

In this case if expression1 is evaluated to true, then its statements are executed and then the rest of the IF statement is ignored. If not, the expression2 is evaluated and its corresponding statements are executed and the rest of the checks are ignored… so check out the example below to have an idea about how it works:

```
If MyAge < 13 Then
    ' you must be a child
    MsgBox("Child")
ElseIf MyAge < 20 Then
    ' you are a teenager
    MsgBox("Hello Teenager")
ElseIf MyAge < 35 Then
    ' Your age is acceptable
    MsgBox("Hi there young man")
Else
    ' the person is old
```

```
            MsgBox("Hello there old man")
        End If
```

So basically this is how the if statement works. We will create a simple Number Guessing Game and see how the IF statement helps us to do it. So basically start a new project, and Create a form with two buttons, a list box and a label as shown below:



Now for the Start button's event write the following code:

```
    Dim SecretNumber As Integer
    Dim Attempts As Integer

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Randomize()
        SecretNumber = Int(Rnd() * 100)
        Attempts = 0
        ListBox1.Items.Clear()
        Label1.Text = "Attempts:" & Attempts.ToString
    End Sub
```

The variables SecretNumber and Attempts are declared outside the subroutine so that their value will persist during program execution. The statements

```
        Randomize()
        SecretNumber = Int(Rnd() * 100)
```

Are used to generate a random number. The numbers are usually generated using some pattern. Each execution the same pattern of numbers appears. The first statement Randomize()makes sure that does not happen. The Rnd()function is used to generate a random number between 0 and 1. Multiply that by 100 you get a value between 0 and 100.

```
        Attempts = 0
        ListBox1.Items.Clear()
        Label1.Text = "Attempts:" & Attempts.ToString
```

These statements resets the number of guessing attempts the play has made, and clears the listbox from previous attempts.

The code for the second button is:

```
    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
        Dim MyNumber As Integer
        Dim Tmp As String
        Tmp = InputBox("Enter a number between 1 and 100", "Guessing
game")

        If IsNumeric(Tmp) Then
            MyNumber = Tmp
        Else
            MsgBox("you should enter a number")
            Exit Sub
        End If

        If MyNumber = SecretNumber Then
            MsgBox("You Guessed the correct number", MsgBoxStyle.OkOnly)
        ElseIf MyNumber > SecretNumber Then
            ListBox1.Items.Add("you should enter a lower number")
            MsgBox("your guess is wrong")
        Else
            ListBox1.Items.Add("you should enter a higher number")
            MsgBox("your guess is wrong")
        End If
        Attempts = Attempts + 1
        Label1.Text = "Attempts:" & Attempts.ToString
```

```
End Sub
```

The code is explained as follows:

```
Dim MyNumber As Integer
Dim Tmp As String
Tmp = InputBox("Enter a number between 1 and 100", "Guessing game")
```

Here we define a number variable to store our guess in. We also need a string variable. This one will hold the value enter by the user so that we can check if it is a number of not (because the user can enter text value instead of a number).

```
If IsNumeric(Tmp) Then
    MyNumber = Tmp
Else
    MsgBox("you should enter a number")
    Exit Sub
End If
```

The IsNumeric is a function that is used to check if a string represent a number or not. So this part will assign the number inside Tmp into MyNumber if it is a proper number representation. Otherwise you get a message telling you about the error and the execution to the subroutine terminates because of the Exit Sub statement. Next:

```
If MyNumber = SecretNumber Then
    MsgBox("You Guessed the correct number", MsgBoxStyle.OkOnly)
ElseIf MyNumber > SecretNumber Then
    ListBox1.Items.Add("you should enter a lower number")
    MsgBox("your guess is wrong")
Else
    ListBox1.Items.Add("you should enter a higher number")
    MsgBox("your guess is wrong")
End If
```

This is the important part were we check the number against what the computer generated. If the numbers are a match then we display a message telling the user about

his guess. If not the user get a wrong guess message and the computer tells if you should guess a higher or lower number. Finally:

```
Attempts = Attempts + 1
Label1.Text = "Attempts:" & Attempts.ToString
```

Will only update the number of attempts.

So this is a very simple introduction to IF statements, and we will be using these more often in later chapters.

Chapter 8: FOR statement

FOR statement

Almost every language has some kind of looping statement (in case you don't know what that does, it allows the execution of a number of statements several times). In VB.NET there are a number of looping statements, these are REPEAT, DO and FOR. We will talk about the easiest of them all which is the FOR loop. The FOR loop is written like this:

```
For variable = Min To Max Step JumpStep
    Statement
    Statement
    …
Next
```

The code will execute the statements between the For and Next parts by setting the variable to Min, increasing it by one every time until it reaches Max. To make things clear consider this example

```
For A = 1 To 10
    MsgBox("The value of A is:" & A)
Next
```

The result of executing the code above is ten message boxes telling you the value of A every time.

Now let us consider another example. Here you have a form with a textbox and a ComboBox. You select font size from the combo box and the text size changes accordingly.

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
                       System.EventArgs) Handles MyBase.Load
    ' this part fills the combobox with the sizes of font that we
    ' can pick from
    Dim I As Double
    For I = 12 To 70
        ComboBox1.Items.Add(I)
    Next
End Sub
```

```
Private Sub ComboBox1_SelectedIndexChanged(ByVal sender As
            System.Object, ByVal e As System.EventArgs) Handles
            ComboBox1.SelectedIndexChanged
    ' this part changes font size
    Dim F As Font
    F = New Font("COURIER NEW", ComboBox1.Text)
    TextBox1.Font = F
End Sub
```

You notice two things in the example, first the loop does not start from 1, it starts from 12, you can start from any value you like, for example start from 283732, -12, 0, 88888, etc. Second the data type of the variable I is double. You can use Single, Double, Integers, Long… You are not restricted here.

If we want to display the numbers between 5 and 50 by adding 5 to the previous in each step then:

```
Dim Counter As Integer
For Counter = 5 To 50 Step 5
    MsgBox(Counter)
Next
```

Assume we need the values 0, 0.1, 0.2, 0.3, 0.4, 0.5… 1.0. This can be done in two ways:

```
Dim Counter As Integer
Dim V As Double
For Counter = 0 To 10
    V = Counter / 10.0
    MsgBox(V)
Next
```

This method requires extra variable, and does not take advantage of the for loop. A better way is to use the STEP keyword with double or single data type to make it easy for us:

```
Dim Counter As Double
For Counter = 0 To 1 Step 0.1
    MsgBox(Counter)
Next
```

One last important thing to notice is that the initial value of the variable should always be smaller than or equal to the value after the To keyword, otherwise the for loop does not get executed and it is skipped. For example:

```
For Counter = 10 To 1
    MsgBox(Counter)
Next
```

Will never give you message box at all. To fix this and make the count down work, just put a negative step value:

```
For Counter = 10 To 1 Step -1
    MsgBox(Counter)
Next
```

These are most of the details needed to work with the For loop. The next example is a simple one showing how to use the FOR loop to identify Prime number.

Prime numbers are numbers that can only be divided by themselves and 1 with remainder=0. So this means if we have number 9212, we should check the remainder of dividing this number over all the values from 9212 to 2 and it should never give a zero if it is a prime. Without for loop this is very hard to compute. The code to calculate the prime number is:

```
Dim MyNumber As Integer
Dim RemainderIsZeroFlag As Boolean
Dim I As Integer

' read a number from the screen
MyNumber = InputBox("Enter a number")

' this is a flag to tell us when the condition
' of prime number is not satisfied
RemainderIsZeroFlag = False

' start checking all the numbers
For I = 2 To MyNumber - 1
    ' if the condition is not satisfied
```

```
        If MyNumber Mod I = 0 Then
            ' mark that the remainder is not zero
            RemainderIsZeroFlag = True
        End If
    Next

    ' if there was any remainder then tell the user
    ' that the number is not prime, else it is.
    If RemainderIsZeroFlag Then
        MsgBox("The number is not prime")
    Else
        MsgBox("The number is prime")
    End If
```

Next chapter we will start working with arrays and collections, and things will get more exciting.

Chapter 9: Arrays

Arrays

In many cases you need to perform processing on huge amount of data. For example you may want to find an average of 7000 values in a file, or reading unknown number of people names into your application and recalling them back. In such case it is inconvenient to define all these variables separately. Instead you define an array which holds all the records.

Think of an array as a big variable that you can get and store values in it by specifying position. For example, below is an array of integer, let us call it **A**:

Element 0 in A	33
Element 1 in A	123
Element 2 in A	3
Element 3 in A	-5
Element 4 in A	19
Element 5 in A	77
Element 6 in A	1212
Element 7 in A	0

The cells in green are just shown here to show you the position of the variable, while the cells in purple show you the content of each variable. As you can see here this array can store 8 variables (starting from 0 and ending at 7). The first location is always 0, some people might find it confusing. You can ignore the first location and avoid any confusion, but you need to know where does the indexing of the array start.

In VB.Net to create an array like the above, you write the following:

```
Dim A(7) As Integer
```

This statement generates an array that can store integers. The array has 8 elements (starting with index 0, and ending with 7). When the statement above is executed, you get the following result in memory:

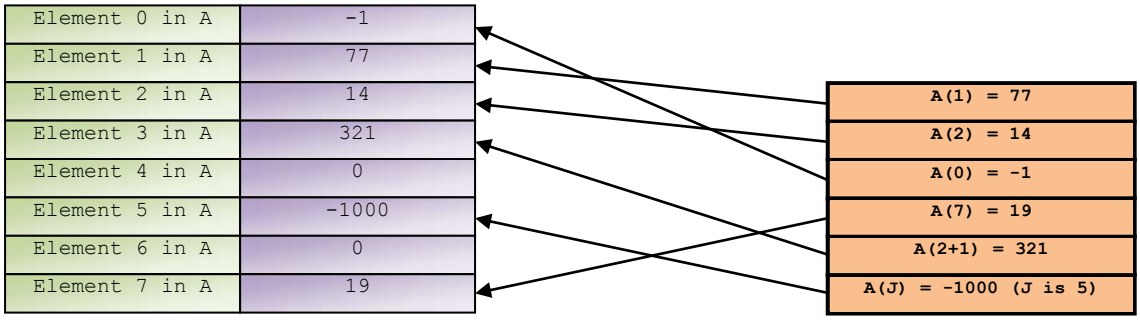
Element 0 in A	0
Element 1 in A	0
Element 2 in A	0
Element 3 in A	0
Element 4 in A	0
Element 5 in A	0
Element 6 in A	0
Element 7 in A	0

To fill the array, you use something like below:

```
' fill element in locations 0, 1, 2, and 7 (no need to fill in order)
        A(1) = 77
```

```
        A(2) = 14
        A(0) = -1
        A(7) = 19

' fill element in location 3 (can use expression to find the location)
        A(2 + 1) = 321

' fill element in location 5 (can use variable to find the location)
        Dim J As Integer
        J = 5
        A(J) = -1000
```

In order to understand how this works, check out the graph below:



Now, let us fill the array with numbers from 0 to 7:

```
' define array first
Dim MyNumbers(7) As Double

' define a counter
Dim I As Integer

' fill the array
For I = 0 To 7
    MyNumbers(I) = I
Next
```

If we want to fill the values from the keyboard (let us say, we want the user to enter 10 names):

```
' define the array
Dim My10Names(9) As String
```

```
        ' define a counter
        Dim C As Integer

        ' fill the array
        For C = 0 To 9
            My10Names(C) = InputBox("Enter the number number " & C)
        Next
```

To get the values out of the array, you use the same format used above. For example the following code shows the content of an array in a list box:

```
        ' display the result
        ListBox1.Items.Clear()
        For I = 0 To 7
            ListBox1.Items.Add(My10Names(I))
        Next
```

To test what we have learned, you can find a simple application on the web site that shows you how to use arrays, by filling them, finding max and min values, and finding average. You need to add a DataGridView, and 4 CommandButtons to your form. The code is:

```
Public Class Form1

    ' we define two arrays one to store names, another to store marks
    Dim Names(9) As String
    Dim Marks(9) As Integer

    ' we define a variable to store how many element of the array we
    ' used
    Dim StCount As Integer = 0

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

        ' read the name and mark and put them in the next empty slot
        Names(StCount) = InputBox("Enter the name of student")
        Marks(StCount) = InputBox("Enter the mark")

        ' the new name and mark should be displayed on the data grid
```

```
        DataGridView1.Rows.Add(Names(StCount), Marks(StCount))

        ' move the counter to the next empty slot
        StCount = StCount + 1
    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
        ' find the maximum mark
        Dim I As Integer            ' used for counting
        Dim MaxPos As Integer       ' used to remember the index of
                                    ' maximum mark
        MaxPos = 0                  ' assume first mark is the maximum
        For I = 1 To StCount - 1    ' loop over all other slots
' is there an element with a mark greater  than the current maximum?
            If Marks(I) > Marks(MaxPos) Then
                MaxPos = I                      ' we found a new max,
                                                ' update our maximum
            End If
        Next

        MsgBox("student " & Names(MaxPos) & " has the maximum mark")
    End Sub

    Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button3.Click
        ' find minimum
        ' it is identical to the previous, except for the condition
        Dim I As Integer
        Dim MinPos As Integer

        MinPos = 0
        For I = 1 To StCount - 1
            If Marks(I) < Marks(MinPos) Then
                MinPos = I
            End If
        Next

        MsgBox("student " & Names(MinPos) & " has the minimum mark")

    End Sub
```

```
    Private Sub Button4_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button4.Click
        Dim I As Integer            ' I is counter
        Dim AVG As Double           ' Used to store the sum and finding
                                    ' the average
        AVG = 0                     ' The avg is zero
        For I = 0 To StCount - 1    ' Loop over all elements in the
                                    ' array
            AVG += Marks(I)         ' Add each element to the some of
                                    ' the previous ones
        Next
        AVG = AVG / StCount         ' divide the total by number of
                                    ' elements to get the average
        MsgBox("the average is:" & AVG)
    End Sub

End Class
```

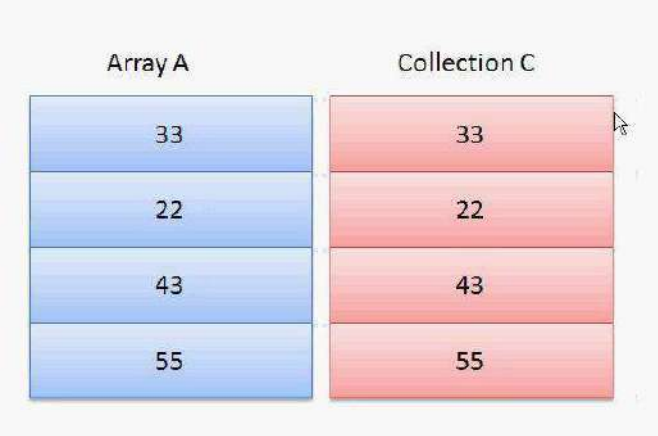
Next chapter we will start working with collections, and see how they are simpler than arrays.

Chapter 10: Collections

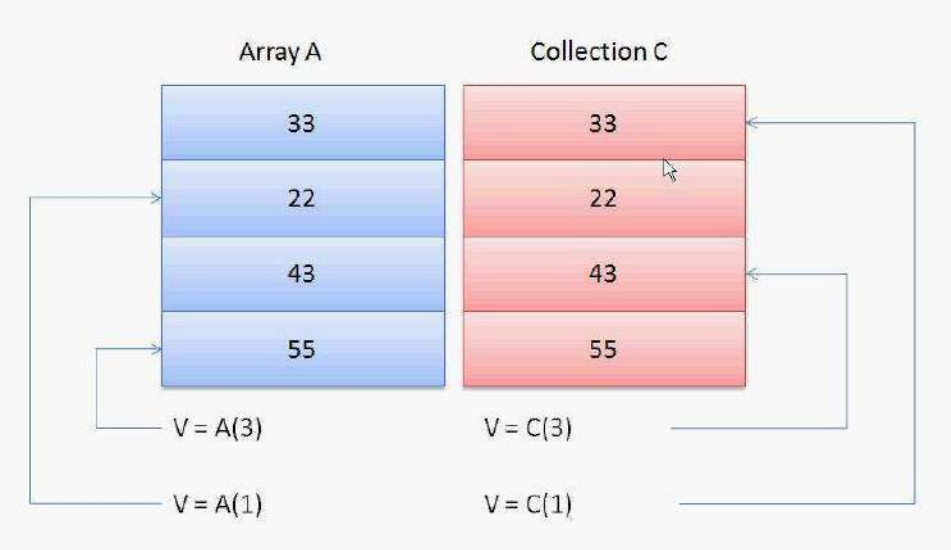
Collections

Last time we spoke about arrays, and saw how to work with them. Today we check out something similar and easier to use, and that is collections.

Collections are very similar to arrays. They are used to store a number of values (or variables), so that you can process them all.



There are a number of differences between arrays and collections. First difference is that the indexing for arrays starts with zero, while for a collection it starts with 1. To understand this, consider the image below:

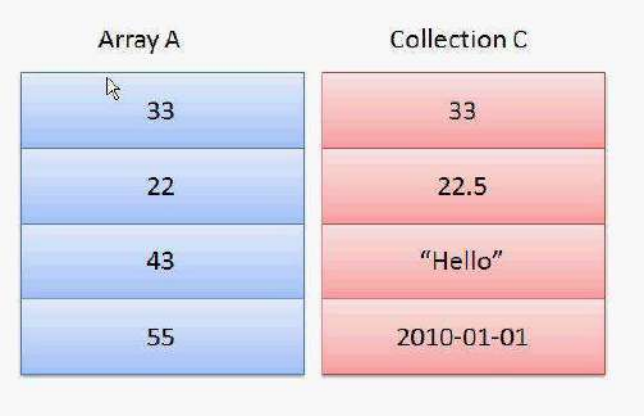


Now the statement V = A(3) will place the value 55 in V because the indexing start at zero in arrays. However the similar statement V = C(3) will place 43 instead because collection indexing is different. The same applies for the second statement.

Another important difference is the data type. All Array elements has the same data type. So if you have an array:

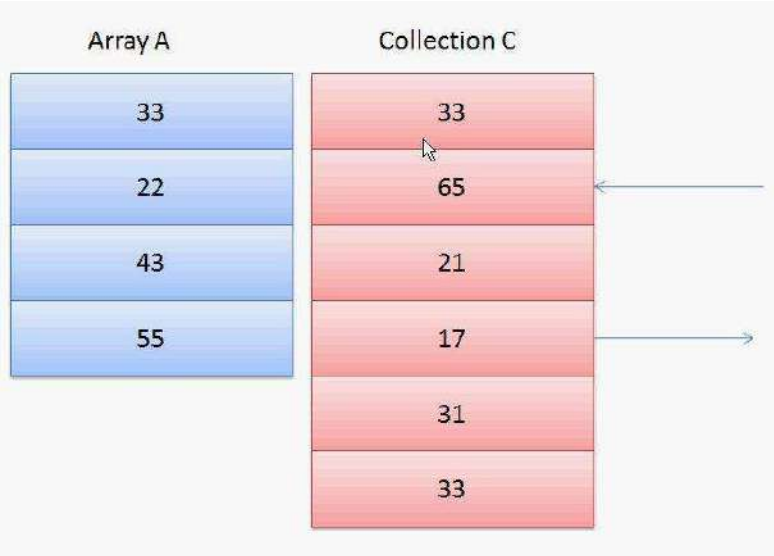
```
Dim A(0 To 9) As Integer
```

then A(0), A(1), A(2)… A(9) are all Integers. Collections on the other hand do not require this. You can store integers, reals, strings, bytes,… etc. in the same collection. This can be illustrated below:

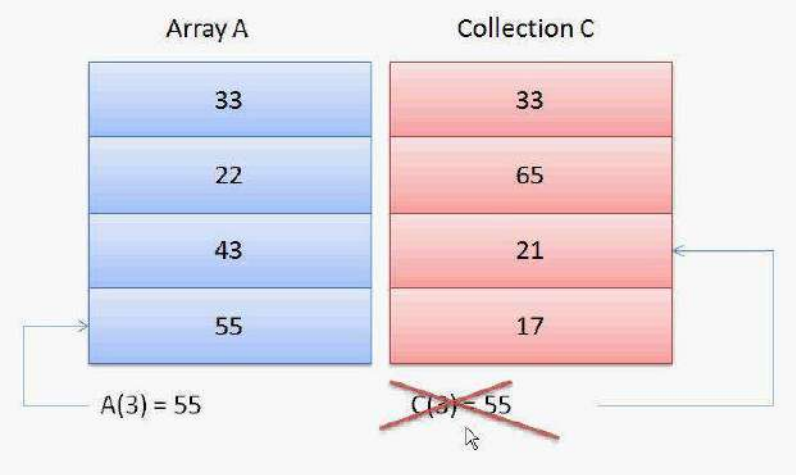


Usually you will use the same data type for all elements of the collection; however you still have the option to use different data types whenever you need to.

Another difference between arrays and collections is that collections can add elements and remove elements directly and change in size without any need for some kind of processing, while arrays are fixed in size, and you cannot insert values at specific locations at random.



Finally array elements can be updated and overwritten, while collections are not. To make this clear check out this:



In this example if you write C(3) = 55 you get an error, that is because collection does not allow you to update or overwrite the content of an element. However there is a way to overcome this. We will discuss this later.

Now let us see how to define a collection. There are two ways to define a collection:

```
Dim C As New Collection
```

In this example you create a collection object that is ready to be used. So you can add, remove elements, get the number of items, or do whatever you want with the collection directly. Another way is:

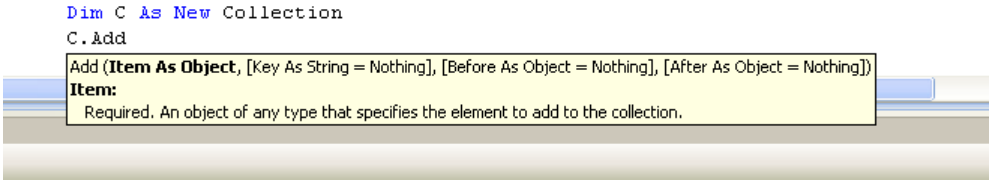
```
Dim C As Collection
```

here C is not ready yet to be used. It does not point to a collection yet. You can never use it. To be able to use it later on in the code you should write:

```
C = New Collection
```

This will allow you to use the collection without any problem. The first method of defining a collection is the one you will probably use the most.

Now in order to add elements to the collection you use the add method.



When you write C.Add the compiler shows you the parameters that you should provide. The ones in the square brackets are optional. The parameters are:

- Item: is the value you want to store
- Key: you can provide a text value to quickly access the elements of collection instead of providing numbers to access them. For example the code below will store the value 40 in V:

```
Dim C As New Collection
C.Add(33, "Smith")
C.Add(40, "Michel")
C.Add(77, "John")

Dim V As String
V = C("Michel")
```

- Before: is the index of the item you want the new element to be inserted before.
- After: is the index of the item you want the new element to be inserted after.

Next is removing elements. This simple, you just provide the index of the element you want to remove:

```
C.Remove(3)
```

This removes the 3rd element from the collection.

Also getting the number of elements is as easy. You use the count function.

```
I = C.Count
```

Finally we describe how to simulate the functionality replacing or updating an item in the array using collection.

If we write :

```
C.Add(N, , , 7)
C.Remove(7)
```

This will have exactly the same effect as:

```
A(6) = N
```

So this is a basic introduction about collections. Next is a simple application demonstrating the use of collections. Create a windows form, and make it similar to what you see below:

Next write the code below:

```
Public Class Form1

    Dim MyCollection As New Collection

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        ' this method clears all the elements in the collection
        MyCollection.Clear()
    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
```

```
        ' read a name
        Dim Name As String
        Name = InputBox("enter a name")

        ' add the name into the list
        MyCollection.Add(Name)
    End Sub

    Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button3.Click
        MsgBox("the number of items is:" & MyCollection.Count)
    End Sub

    Private Sub Button4_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button4.Click
        ' clear old content
        ListBox1.Items.Clear()

        ' insert the items into the list box
        Dim I As Integer
        For I = 1 To MyCollection.Count
            ListBox1.Items.Add(MyCollection(I))
        Next

    End Sub

    Private Sub Button5_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button5.Click
        ' get element position
        Dim I As Integer
        I = InputBox("enter the element number you want to remove:")
        MyCollection.Remove(I)
    End Sub

    Private Sub Button6_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button6.Click
        ' get element position
        Dim I As Integer
        Dim N As String
        I = InputBox("enter the element number:")
        N = InputBox("enter the new name:")
        MyCollection.Add(N, , , I)
        MyCollection.Remove(I)
```

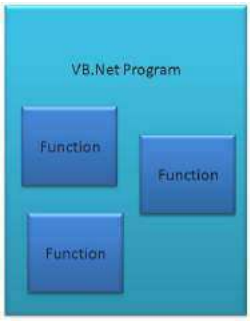
```
    End Sub

End Class
```

The code is very simple and at your level you should understand it very easily. If you have any problem with it just send me notes about it. Next chapter we will start working with functions, and see how it makes coding much easier for us.

Chapter 11: Functions

Functions

Up to this point, when you want to write an application you would simply write all your statements into one single block and try to solve all the problems in that simple block. However there is a better approach to write applications by dividing the application into a number of blocks that form together one application. These what functions are. Think about functions as small or mini-programs that each is designed to address a simple problem.



Functions make your applications easier to write and easier to understand. Now let us consider this example: you need to find the factorial for three variables, A,B and C. (The factorial of 4 is 4x3x2x1, and factorial of 7 is 7x6x5x4x3x2x1).

Usually to solve this issue you write something like:

```
A = 10
B = 5
C = 4

FA = 1
For I = 1 To A
    FA = FA * I
Next

FB = 1
For I = 1 To B
    FB = FB * I
Next

FC = 1
For I = 1 To C
    FC = FC * I
```

```
    Next
```

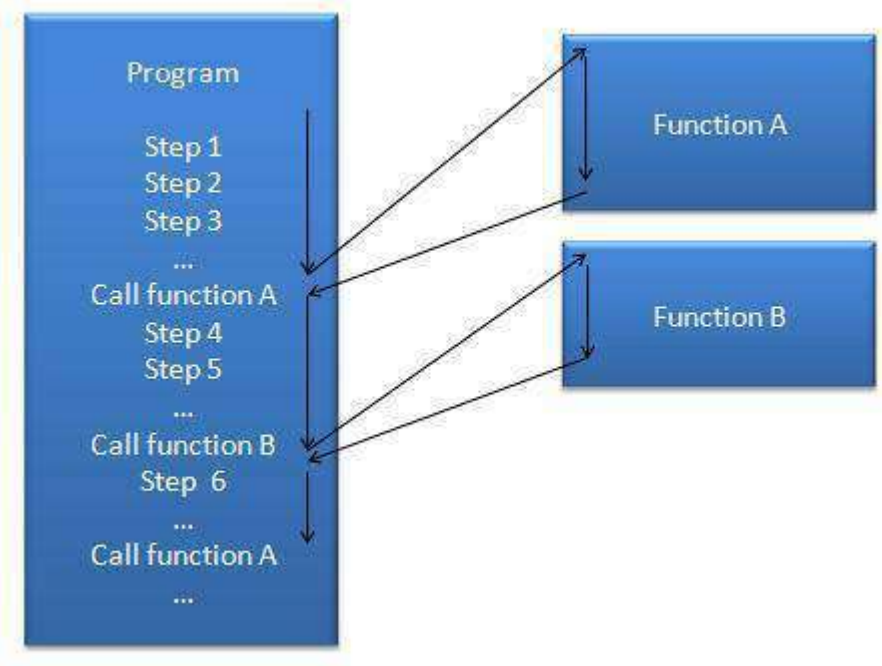
So if you have for example 30 variables, you might need to rewrite the code 30 times (assuming you are not using arrays). However if you think about this in a different way, that is: **Could I have a statement that get me the factorial?** Then the code would be something like:

```
A = 10
B = 5
C = 4

FA = Factorial(A)
FB = Factorial(B)
FC = Factorial(C)
```

Which is very easy to write and understand. All you need is to tell the computer what Factorial really is:

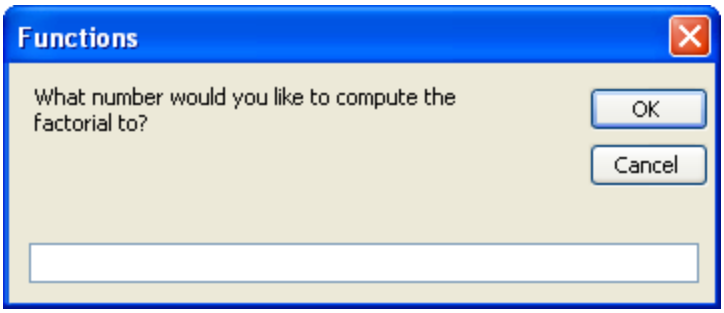
```
Function Factorial(ByVal N As Integer) As Double
    Dim F As Double          ' the factorial total
    Dim I As Integer         ' the counter
    F = 1                    ' the initial value of F
    For I = 1 To N           ' this loop to calculate the factorial
        F = F * I
    Next
    Return F                 ' return the result
End Function
```

Now no matter how many times you need to calculate the factorial, you can just call it whenever you need. You don't have to worry about the loop or initializing F or anything else. You do it only once and you can use it anywhere.



Consider the above figure. It shows how the program behaves when it encounters a function call. The program executes normally, until it reaches a function call. At that time it will transfer its execution to function A. It will execute the function, get the result, and return back to the main program. The execution continues from the function call, and again, when there is another function call the process in repeated.

The functions you use could be built in functions, which means they are already written and available in the compiler, or user defined functions which are the ones you write. Built in functions are used to do general operations needed in most programs, and to make life easier for programmers. Examples of these are the **InputBox** function which shows a small window to read values from the display.



Other examples are math functions like **Math.Pow** - which finds the power of a number - and **Math.Abs** - which finds the absolute value of a number - and string functions like UCase and LCase.

As for user defined functions, you can define a function as follows:

```
' the factorial function
Function Factorial(ByVal N As Integer) As Double
    Dim F As Double          ' the factorial total
    Dim I As Integer         ' the counter
    F = 1                    ' the initial value of F

    For I = 1 To N           ' this loop is used to calculate the
                             ' factorial
        F = F * I
    Next

    Return F                 ' return the result
End Function
```

Function, End Function parts define where the code of the function starts and when it ends.

Factorial is the name of the function. You can choose any name however you might want to choose a meaningful name to remind you what this function does.

ByVal N As Integer is the parameter of the function. For now ignore the ByVal part. The parameter is a value passed from your program to the function so that the function could work on it. You define all the parameters between the brackets.

As Double which comes after the brackets tells the compiler what data type this function returns. When a function finish its processing it should return the result in the specified data type.

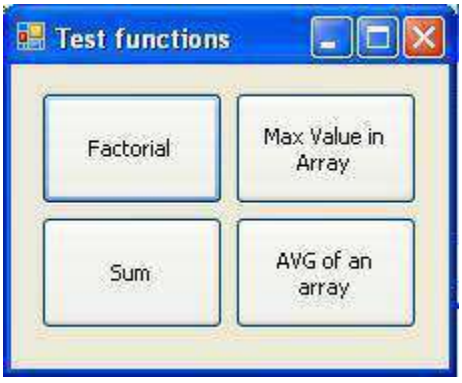
Return is used to tell the compiler what is the result of executing this function.

The following is an example of a function:

```
    ' the sum function
    Function Sum(ByVal V1 As Integer, ByVal V2 As Integer, ByVal V3 As
Integer) As Integer
        Return V1 + V2 + V3 ' calculate the sum and return the result in
```

```
                                ' one single step
    End Function
```

Now let us check the application included with the tutorial and see how it works:



It allows you to compute the factorial of a number, find the sum of 3 numbers, read and find max value or average of an array.

The factorial function is:

```
    ' the factorial function
    Function Factorial(ByVal N As Integer) As Double
        Dim F As Double          ' the factorial total
        Dim I As Integer         ' the counter
        F = 1                    ' the initial value of F

        For I = 1 To N           ' this loop is used to calculate the
factorial
            F = F * I
        Next

        Return F                 ' return the result
    End Function
```

and the code of the factorial button is:

```
        ' read the number from screen
        Dim MyNumb As Integer
        Dim R As Double
        MyNumb = InputBox("What number would you like to compute the
factorial to?")
        R = Factorial(MyNumb)
        MsgBox("the result is:" & R)
```

Check out how to call the function, you use its name and pass the parameter value. You can even call the function in the following ways:

```
R = Factorial(MyNumb + 2)
R = Factorial(6)
R = Factorial(77/2)
R = Factorial(A - B)
```

and the value of the expression between the brackets is evaluated and passed to the function and placed in the variable N.

The rest of the code is straight forward. However I would like to highlight the functions that process functions:

```
' the max value in an array
Function GetMax(ByVal A() As Integer) As Integer
    Dim I As Integer
    Dim Max As Integer

    Max = A(0)                      ' assume the max value is the
                                    ' first value
    For I = 1 To A.Length - 1       ' loop over all other values in
                                    ' an array
        If Max < A(I) Then          ' if we find another value
                                    ' larger than max then
            Max = A(I)              '    update max
        End If
    Next
    Return Max                      ' return the result
End Function

' the avg value of an array
Function GetAVG(ByVal A() As Integer) As Integer
    Dim I As Integer
    Dim sum As Integer

    sum = 0                         ' assume the sum value is 0
    For I = 0 To A.Length - 1       ' loop over all other values in
                                    ' an array
        sum = sum + A(I)
```

```
    Next

    Return sum / A.Length           ' return the result
End Function

' read array
Function ReadArray() As Integer()
    ' the counter
    Dim I As Integer

    ' the number of elements
    Dim N As Integer

    ' read the number of elements from the display
    N = InputBox("how many elements in the array")

    ' create the array
    Dim A(0 To N - 1) As Integer

    ' read the elements of the array
    For I = 0 To N - 1
        A(I) = InputBox("enter the element:" & I.ToString)
    Next

    ' return the result
    Return A
End Function
```

When you pass array to a function you use brackets () to tell the function that it is going to receive the array. In ReadArray() When the function returns an array, you add brackets after the return data type.

Next chapter we discuss functions more.

Chapter 12: ByVal & ByRef

ByVal & ByRef

One of the useful properties of functions is that it allows you to propagate changes in a parameter back to its original variable. For example consider this simple functions:

```
Function TestByVal(ByVal N As Integer) As Integer
    N = 0
    Return N
End Function

Function TestByRef(ByRef N As Integer) As Integer
    N = 0
    Return N
End Function
```

Now if you call the functions like this:

```
    Dim K1 As Integer = 100
    TestByVal(K1)
    MsgBox(K1)

    Dim K2 As Integer = 100
    TestByRef(K2)
    MsgBox(K2)
```

The first MSGBOX will display the value 100, while the second will show the value zero. This can be explained as follows:

When the computer sees the ByVal keyword it will create an independent copy of the variable you are passing to the function and work on it. In our example TestByVal, the function sets the variable N to zero. Since it is passed by value this won't affect the orginal parameter K1.

However when you use ByRef keyword the computer will create a copy of the variable that is linked to the original parameter. In other words the variable N in the example is another name for the variable K2, so when you change N in the second

function, K2 changes as well. So this is why you get the value zero from the second MSGBOX.

Now let us create a useful example to swap two numbers:

```
' swap function
Function Swap(ByRef V1 As Integer, ByRef V2 As Integer)
    Dim Tmp As Integer
    Tmp = V1
    V1 = V2
    V2 = Tmp
End Function
```

Here you are passing two numbers into the function and the function swaps them. Now to call the function you can try something like this:

```
    Dim N1 As Integer
    Dim N2 As Integer
    N1 = InputBox("Enter N1")
    N2 = InputBox("Enter N2")
    Swap(N1, N2)
    MsgBox("N1:" & N1)
    MsgBox("N2:" & N2)
```

So whenever you call the function the numbers are changed. Try to modify the code of the function to have some errors, for example let it be like this:

```
' swap function modified
Function Swap(ByVal V1 As Integer, ByVal V2 As Integer)
    Dim Tmp As Integer
    Tmp = V1
    V1 = V2
    V2 = Tmp
End Function
```

Try the same code and you will see it will not work as we clarified before. Also try modifying the function to be like this:

```
' swap function modified
Function Swap(ByVal V1 As Integer, ByRef V2 As Integer)
    Dim Tmp As Integer
    Tmp = V1
    V1 = V2
    V2 = Tmp
End Function
```

Here you are going to get strange result, one of the values will be changed, while the other not simply because one of the parameters if by reference and the other is by value. So just in case you are getting unexpected results and your function code appear to be correct, just make sure that the ByVal and ByRef keywords are used correctly.

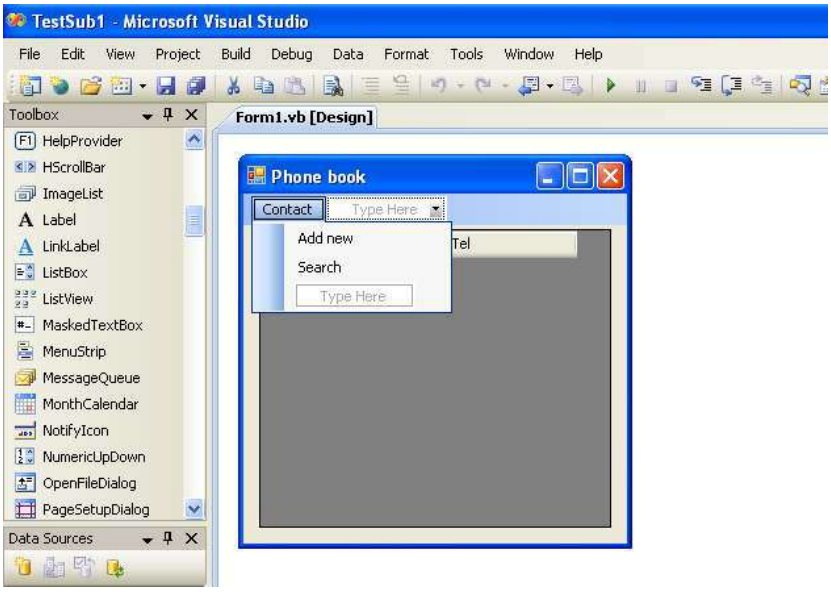
Next chapter deals with subroutine, and try to create a useful sort program.

Chapter 13: Subroutines

Subroutines

In vb.net you can write other type of coding blocks called subroutines. Subroutines are almost exactly similar to function except they don't return a value. In this chapter we are going to create a simple application that stores person's name and telephone number and allows you to search the names. If you did not read the chapter about working with functions, then you won't be able to understand this one.

Open a new project, and then create the interface similar to the below:



You just add a DataGridView control and a menustrip. Set fields for the DataGridView control.

Next go to the code window and write the following:

```
' define the main variables
Dim Names As New Collection
Dim Tels As New Collection
```

These two collections are going to store the names and telephone numbers, next write:

```
' add new person
Sub AddContact(ByVal CName As String, ByVal CTEL As String)
    Names.Add(CName)
    Tels.Add(CTEL)
End Sub
```

This one is a subroutine that will add the name and telephone numbers to the collections. As you can see there is no return value, and the definition is very similar to the functions. The next subroutine is used to view the content of collections in the DataGridView control.

```
    ' display the names in the grid
    Sub ViewContacts(ByVal DGV As DataGridView)
        DGV.Rows.Clear()
        Dim I As Integer
        For I = 1 To Names.Count
            DGV.Rows.Add(Names(I), Tels(I))
        Next
    End Sub
```

Now, we need to add the code for the "Add New" person menu item, go to the user interface, and double click the Add New menu item, and write the code:

```
    ' the handler for the add new contact command
    Private Sub AddNewToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
AddNewToolStripMenuItem.Click
        Dim N As String
        Dim T As String

        N = InputBox("Enter the name of the contact:")
        If N = "" Then
            Exit Sub
        End If

        T = InputBox("Enter the tel number:")
        If T = "" Then
            Exit Sub
        End If

        AddContact(N, T)
        ViewContacts(DataGridView1)
    End Sub
```

If you check the definition of the block:

```
    Private Sub AddNewToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
AddNewToolStripMenuItem.Click
```

you realize that the event handler is actually a subroutine. So we have been using subroutines all the time. Next create a search function and the event handler for the search menu item:

```
    ' the search function
    Function GetTelForName(ByVal Name As String) As String
        Dim I As Integer
        For I = 1 To Names.Count
            If Names(I) = Name Then
                Return Tels(I)
            End If
        Next
        Return ""
    End Function

    Private Sub SearchToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
SearchToolStripMenuItem.Click
        Dim N As String
        Dim T As String
        N = InputBox("Enter the name you are searching for:")
        If N = "" Then
            Exit Sub
        End If

        T = GetTelForName(N)
        If T = "" Then
            MsgBox("Name not found")
        Else
            MsgBox("the tel number is:" & T)
        End If
    End Sub
```

Finally run the application, try to add some names, and perform a search. So this is a quick example about using subroutines.

Chapter 14: Do Loop

Do Loop

In VB.NET, you can use For loop which we saw how to use in one of the previous tutorials. Another type of loop is the Do loop. The format of this loop can be one of the following:

```
Do
      Statement...
      Statement...
      …
Loop While Condition
```

Or

```
Do
      Statement...
      Statement...
      …
Loop Until Condition
```

Or

```
Do While Condition
      Statement...
      Statement...
      …
Loop
```

Or

```
Do Until Condition
      Statement...
      Statement...
      …
Loop
```

The location of the condition tells when the checking of exiting or staying in the loop should be performed. For example:

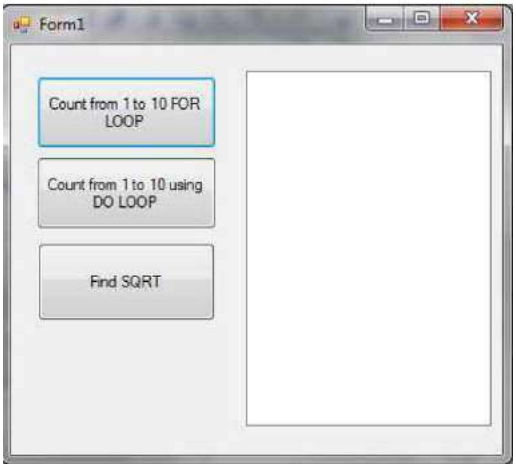
```
Do While K<10
Statement...
Statement...
…
Loop
```

Here the check is performed every time before the loop is executed. It is possible in this example for the loop to never get executed. However if you write it like this:

```
Do
      Statement...
      Statement...
      …
Loop While K<10
```

The check is performed after the loop is executed. So here the loop is executed at least once.

Now let us test the Do Loop with an example:



Create a form similar to the above picture, and then for the first button write the code:

```
Dim I As Integer
ListBox1.Items.Clear()
For I = 1 To 10
      ListBox1.Items.Add(I)
Next
```

This is the standard For loop, Now we will create a similar loop using the Do loop. Add the following code for the second button.

```
Dim I As Integer
I = 1
ListBox1.Items.Clear()
Do
      ListBox1.Items.Add(I)
      I = I + 1
Loop While (I < 11)
```

Run the application and see how both loops give the same result. The thing about the Do loop here is that you much initialize the counter (I) yourself, and increase its value, and check when you should stop the loop. This is something you don't have to do with the For loop.

The Do loop is not used to replace the For loop, but it is used when you don't know how many times you need to execute the code. For example, assume you want to find the square root for an unknown number of values, and you want to stop when you enter a negative number. The way you do it is as follows:

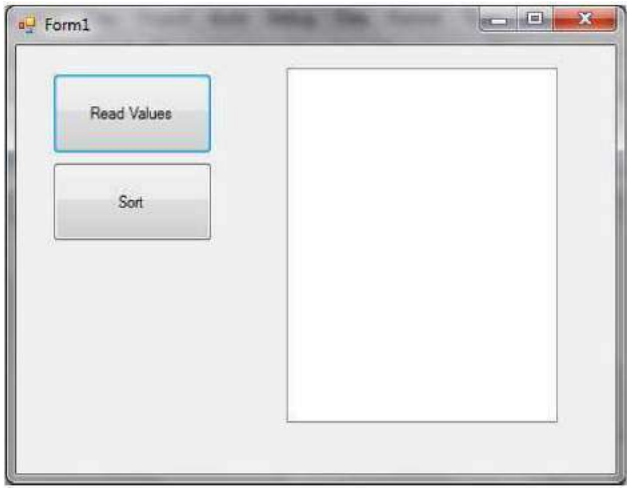
```
Dim I As Double
Do
      I = InputBox("Enter a +ve number:")
      If I >= 0 Then MsgBox("the root of your number is:" &
Math.Sqrt(I))
Loop While I > 0
```

Here you can stop after entering one value, or after entering … say 500 +ve numbers. Try to add the code to the third button in the form above and check it out. The full code for the form should be similar to the following:

```
Public Class Form1
      Private Sub Button1_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs)Handles Button1.Click
      Dim I As Integer
```

```
        ListBox1.Items.Clear()
        For I = 1 To 10
              ListBox1.Items.Add(I)
        Next
        End Sub
        Private Sub Button2_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs)Handles Button2.Click
        Dim I As Integer
        I = 1
        ListBox1.Items.Clear()
        Do
              ListBox1.Items.Add(I)
              I = I + 1
        Loop While (I < 11)
        End Sub
        Private Sub Button3_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs)
Handles Button3.Click
              Dim I As Double
              Do
                    I = InputBox("Enter a +ve number:")
                    If I >= 0 Then MsgBox("the root of your number is:" &
              Math.Sqrt(I))
              Loop While I > 0
        End Sub
End Class
```

Now let us try another example with the Do Loop, this time we use it to perform bubble sort. Create a form similar to the one below:



Next write down the code of the form to be as follows:

```
Public Class Form1
      Dim A() As Integer
      Private Sub Button1_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs)
Handles Button1.Click
      Dim Count As Integer
      Count = InputBox("enter the number of values:")
      ReDim A(0 To Count - 1)
      Dim I As Integer
      For I = 0 To Count - 1
            A(I) = InputBox("Enter the value " & I.ToString)
      Next
      ViewArray(A, ListBox1)
      End Sub
      Public Sub ViewArray(ByVal Ar() As Integer, ByVal L As ListBox)
            L.Items.Clear()
            Dim I As Integer
            For I = 0 To Ar.Length - 1
                  L.Items.Add(Ar(I))
            Next
      End Sub
      Private Sub Button2_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs)
Handles Button2.Click
            ' sort
            Dim I As Integer
            Dim J As Integer
            Dim Flag As Boolean
            Do
                  Flag = False
                  For I = 0 To A.Length - 2
                        If A(I) > A(I + 1) Then
                              Flag = True
                              J = A(I)
                              A(I) = A(I + 1)
                              A(I + 1) = J
                        End If
                  Next
            Loop Until Flag = False
            ViewArray(A, ListBox1)
```

```
      End Sub
End Class
```

Check out the code for the sort operation:

```
' sort
Dim I As Integer
Dim J As Integer
Dim Flag As Boolean
Do
      Flag = False
      For I = 0 To A.Length - 2
            If A(I) > A(I + 1) Then
                  Flag = True
                  J = A(I)
                  A(I) = A(I + 1)
                  A(I + 1) = J
            End If
      Next
Loop Until Flag = False
```

The Flag is used to tell when any two values in the array are exchanged, which tells the computer to perform another loop on array elements. Notice that this bubble sort is not very efficient, and it can be improved. However it is used here to just show you an example of when to use the Do Loop.

Chapter 15: Structures

Structures

In VB.NET, you can combine a number of related variables together and treat them as one unit. This simplifies programming and makes updating the source of your applications easier.

Let us first consider the following example; you want to store the information about a person/employee. The information include name, telephone number and salary. So far we learned that to store such information you should define 3 distinct variables:

```
Dim Name As String
Dim TEL As String
Dim Sal As Decimal
```

Later on you fill these variables with values, and uses them. Now what is you have two employees? Obviously you define another 3 variables:

```
Dim Name2 As String
Dim TEL2 As String
Dim Sal2 As Decimal
```

Now what if you have a 1000 employee? Well a better solution is to use arrays. However now you need to define 3 arrays:

```
Dim Names() As String
Dim TELs() As String
Dim Sals() As Decimal
```

The first array stores the names, the second stores telephone numbers, and last one stores salary. So arrays handles the information for large amount of data pretty well. But what is you need to add another property such as address? The solution is to add another array:

```
Dim Address() As String
```

And if you need to store another property you need to store another array. If you have

14 property for an employee, then you have to store and manage 14 different arrays. In such situations structures are useful. You define a structure to combine the different properties like this:

```
Structure PersonInfo
      Dim Name As String
      Dim Tel As String
      Dim Sal As Decimal
      Dim Address As String
End Structure
```

Now you have a new data type called PersonInfo which contains inside it a name, a telephone, a salary and an address for that particular employee/person.

```
Dim A As PersonInfo
A.Name = "Smith"
A.Tel = "555-22-332"
A.Sal = 400
```

So A here is the name of the variable and it stores all the attributes or properties of employee/person. To access a specific property you use the dot (.) followed by the property. For example A.NAME access the name property of that employee. Now to define another employee/person you write:

```
Dim B As PersonInfo
B.Name = "Micehl"
B.Tel = "111-22-332"
B.Sal = 700
```

So now you have two employees A & B. Now if you want to define an array of such structure you can do so by:

```
Dim Info() As PersonInfo
```

And you can access the elements of the array normally

```
Dim I As Integer
```

```
Dim N As Integer
N = InputBox("Enter the number of people")
ReDim Info(0 To N - 1)
' read info here
For I = 0 To N - 1
      Info(I).Name = InputBox("enter the name of person")
      Info(I).Tel = InputBox("enter the telephone number")
      Info(I).Sal = InputBox("enter the salary")
Next
```

So as you can see structures are used the same way as normal variables do. Now if you want to copy the information of a structure then :

```
A = B
```

This will allow you to copy all the content and attributes of structure B into A. So it does not matter how many attributes you have, they will all be copied in one single step.
Now let us work on an example. Create a simple form containing two buttons and a data grid view. One of the buttons should read the data and the other should sort the data.
Create the following columns in the data grid :

· Name

· TEL

· Sal

· License

Then go to the code page of the form and define the following structure:

```
' this is the structure to store person information
Structure PersonInfo
      Dim Name As String
      Dim Tel As String
      Dim Sal As Decimal
      Dim LincenseNumber As String
End Structure
```

Then define an array of structure:

```
' this is the array to store persons' info
```

```
      Dim Info() As PersonInfo
```

Next create a subroutine to read the information of the array

```
' read the information and store it in an array
Public Sub ReadInfo()
      Dim I As Integer
      Dim N As Integer
      N = InputBox("Enter the number of people")
      ReDim Info(0 To N - 1)
      ' read info here
      For I = 0 To N - 1
            Info(I).Name = InputBox("enter the name of person")
            Info(I).Tel = InputBox("enter the telephone number")
            Info(I).Sal = InputBox("enter the salary")
            Info(I).LincenseNumber = InputBox("enter license number")
      Next
End Sub
```

After that add the following code to display the content of the array

```
' fill the data grid with array info
Public Sub FillDGV(ByVal DAT() As PersonInfo, ByVal DGV As DataGridView)
      DGV.Rows.Clear()
      Dim I As Integer
      For I = 0 To DAT.Length - 1
            DGV.Rows.Add(DAT(I).Name, DAT(I).Tel, DAT(I).Sal,
                         DAT(I).LincenseNumber)
      Next
End Sub
```

Next is to add the code of the read button

```
ReadInfo()
FillDGV(Info, DataGridView1)
```

Next is the sort method:

```
' the sort subroutine
Public Sub Sort(ByRef Arr() As PersonInfo)
```

```
        Dim Flg As Boolean
        Dim I As Integer
        Dim Tmp As PersonInfo
        Do
            Flg = False
            For I = 0 To Arr.Length - 2
                If Arr(I).Name > Arr(I + 1).Name Then
                    Tmp = Arr(I)
                    Arr(I) = Arr(I + 1)
                    Arr(I + 1) = Tmp
                    Flg = True
                End If
            Next
        Loop While Flg
End Sub
```

Look how simple the swap operation is. No need to swap the 4 attributes, you just use one swap operation. For the second button add the following code:

```
Sort(Info)
FillDGV(Info, DataGridView1)
```

So the code should be:

```
Public Class Form1
    ' this is the structure to store person information
    Structure PersonInfo
        Dim Name As String
        Dim Tel As String
        Dim Sal As Decimal
        Dim LincenseNumber As String
    End Structure
    ' this is the array to store persons' info
    Dim Info() As PersonInfo
    ' read the information and store it in an array
    Public Sub ReadInfo()
        Dim I As Integer
        Dim N As Integer
        N = InputBox("Enter the number of people")
        ReDim Info(0 To N - 1)
```

```
        ' read info here
        For I = 0 To N - 1
              Info(I).Name = InputBox("enter the name of person")
              Info(I).Tel = InputBox("enter the telephone number")
              Info(I).Sal = InputBox("enter the salary")
              Info(I).LincenseNumber = InputBox("enter license
              number")
        Next
End Sub
' fill the data grid with array info
        Public Sub FillDGV(ByVal DAT() As PersonInfo, ByVal DGV As
        DataGridView)
        DGV.Rows.Clear()
        Dim I As Integer
        For I = 0 To DAT.Length - 1
              DGV.Rows.Add(DAT(I).Name, DAT(I).Tel, DAT(I).Sal,
              DAT(I).LincenseNumber)
        Next
End Sub
' the sort subroutine
Public Sub Sort(ByRef Arr() As PersonInfo)
        Dim Flg As Boolean
        Dim I As Integer
        Dim Tmp As PersonInfo
        Do
              Flg = False
              For I = 0 To Arr.Length - 2
                    If Arr(I).Name > Arr(I + 1).Name Then
                          Tmp = Arr(I)
                          Arr(I) = Arr(I + 1)
                          Arr(I + 1) = Tmp
                          Flg = True
                    End If
              Next
        Loop While Flg
End Sub
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs)
Handles Button1.Click
        ReadInfo()
        FillDGV(Info, DataGridView1)
End Sub
```

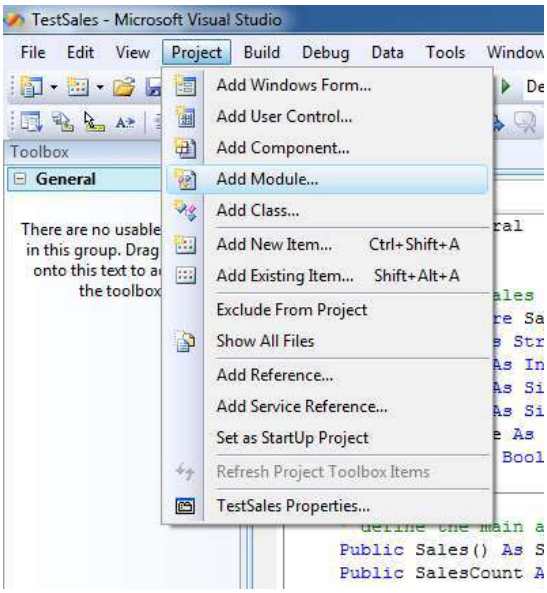
```
    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e
    As System.EventArgs)
    Handles Button2.Click
        Sort(Info)
        FillDGV(Info, DataGridView1)
    End Sub
End Class
```

The source file contains a simple added code, you should try to use that to add more functionality to the example. Try to include car information.

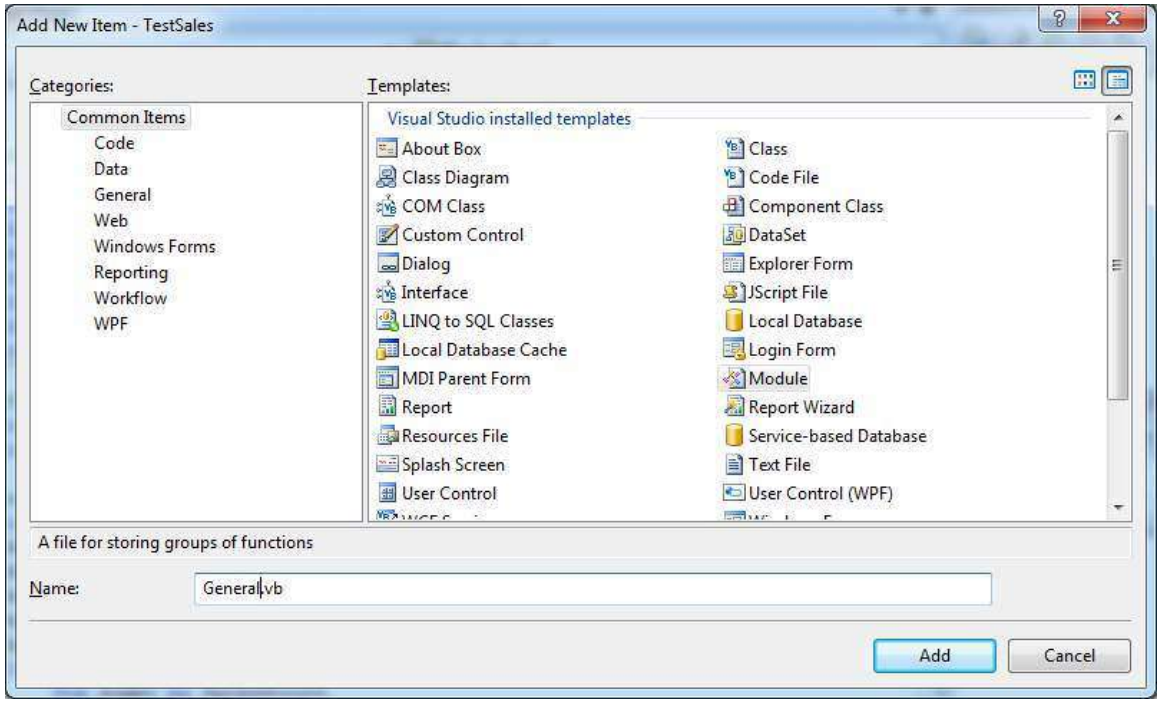
Chapter 16: Modules

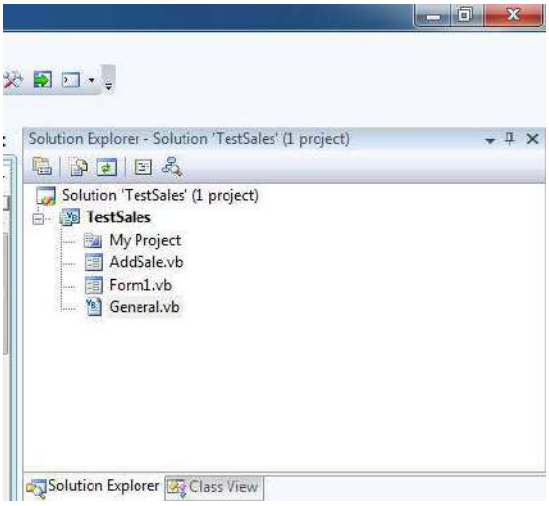
Modules

In vb.net you can write place your code in different files. Some of these files are called modules. A module is a file that contains vb code only (i.e. functions, structures, subroutines…). It does not include GUI like buttons, lists, menus …etc. Now to add a module to your project select project then add module



After that you provide module name:





You can see the module file is added into your project. The code you see is:

```
Module General
End Module
```

Now you can write the functions and subroutines in this module. For example:

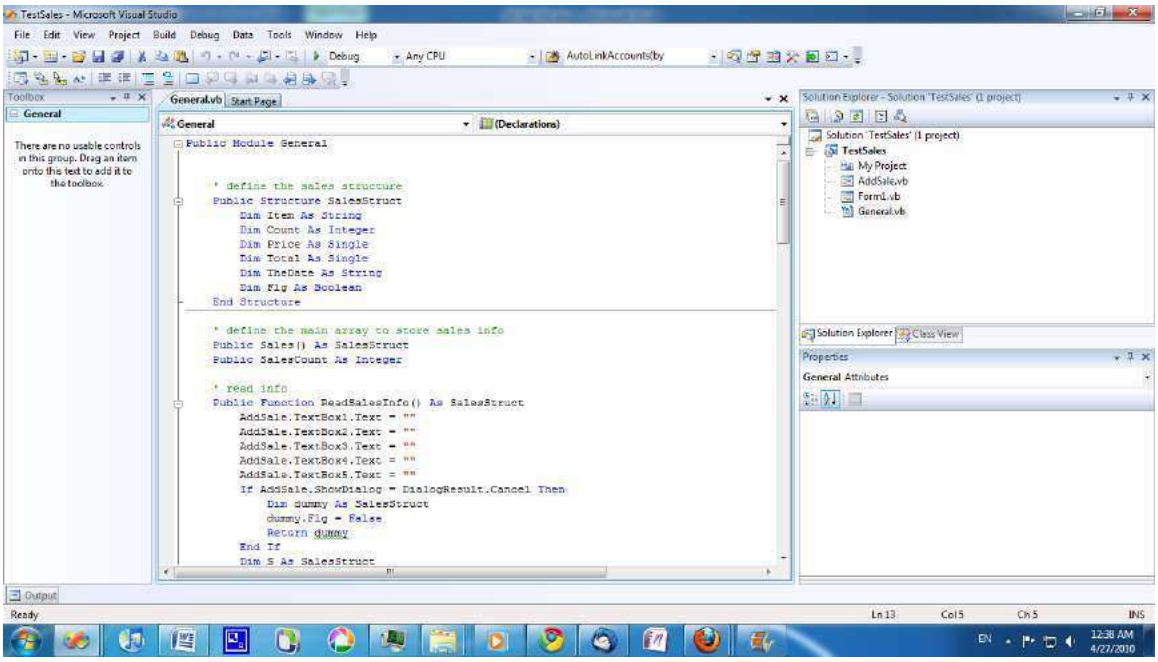
```
Module General
    ' define the sales structure
    Public Structure SalesStruct
        Dim Item As String
        Dim Count As Integer
        Dim Price As Single
        Dim Total As Single
        Dim TheDate As String
        Dim Flg As Boolean
    End Structure
End Module
```

So you might be wondering what difference does modules make in a program? Well modules helps you place related functions, subroutines, and other coding that you make in one place, so that it becomes easier for you to find, and easier for you to work with, and make it possible for other parts of your program to use the code.

To illustrate this consider that you have a two or three forms that require some sort operation. Instead of writing the code in one form which makes it part of that form, you place the code in a module, and sort function/subroutine becomes available to all the

forms. Later on you can even take the code of the sort operation and add that to another project and you will find it works without modification (assuming the code is done correctly).

Now the following example (the sales vb project included on the web site). This project contains one module:



The idea of this project is to add a number of items you were able to sale, and later on you could find the total of sales, save or load the files.

If you check the module you will find the code:

```
' define the sales structure
Public Structure SalesStruct
    Dim Item As String
    Dim Count As Integer
    Dim Price As Single
    Dim Total As Single
    Dim TheDate As String
    Dim Flg As Boolean
End Structure
```

Which defines the main structure, then

```
' define the main array to store sales info
```

```
    Public Sales() As SalesStruct
    Public SalesCount As Integer
```

Which defines the array and its number of elements

```
    ' read info
    Public Function ReadSalesInfo() As SalesStruct
        AddSale.TextBox1.Text = ""
        AddSale.TextBox2.Text = ""
        AddSale.TextBox3.Text = ""
        AddSale.TextBox4.Text = ""
        AddSale.TextBox5.Text = ""
        If AddSale.ShowDialog = DialogResult.Cancel Then
            Dim dummy As SalesStruct
            dummy.Flg = False
            Return dummy
        End If
        Dim S As SalesStruct
        S.Item = AddSale.TextBox1.Text
        S.Count = AddSale.TextBox2.Text
        S.Price = AddSale.TextBox3.Text
        S.Total = AddSale.TextBox4.Text
        S.TheDate = AddSale.TextBox5.Text
        S.Flg = True
        Return S
    End Function
```

This function uses a dialog called AddSale to read the information of an item. The first part just clears the text boxes on the form/dialog, and the if statement part shows the window and tells you if the user canceled the data entry, and the last part fills the structure from the form and returns the result.

```
    ' display the information of the strucutre in the data grid view
    Public Sub DisplayArray(ByVal Arr() As SalesStruct, ByVal DGV As
DataGridView)
        DGV.Rows.Clear()
        Dim I As Integer
        For I = 0 To Arr.Length - 1
            DGV.Rows.Add(Arr(I).Item, Arr(I).Count, Arr(I).Price,
Arr(I).Total, Arr(I).TheDate)
```

```
        Next
    End Sub
```

This part displays the information of the array in a data grid view

```
    ' remove an item from array based on item name
    Public Sub RemoveItemBasedOnName(ByVal Name As String, ByRef Arr()
As SalesStruct, ByRef IC As Integer)
        Dim I As Integer
        Dim J As Integer
        For I = 0 To Arr.Length - 1
            If Name = Arr(I).Item Then
                For J = I + 1 To Arr.Length - 1
                    Arr(J - 1) = Arr(J)
                Next
                ReDim Preserve Arr(0 To Arr.Length - 2)
                IC = IC - 1
                Exit Sub
            End If
        Next
    End Sub
```

This one removes an item based on its name

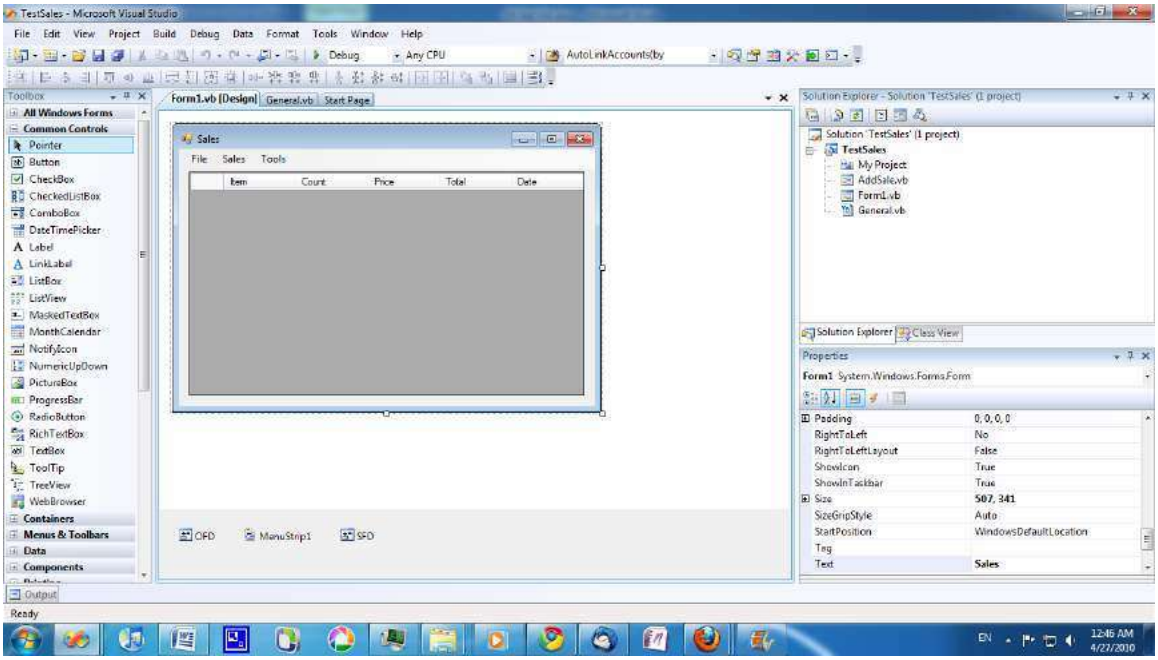
```
    ' save the sales info
    Public Sub SaveFile(ByVal FileName As String, ByVal Arr() As
SalesStruct)
        FileSystem.FileOpen(1, FileName, OpenMode.Output,
OpenAccess.Write)
        Dim I As Integer
        FileSystem.PrintLine(1, Arr.Length)
        For I = 0 To Arr.Length - 1
            FileSystem.PrintLine(1, Arr(I).Item)
            FileSystem.PrintLine(1, Arr(I).Price)
            FileSystem.PrintLine(1, Arr(I).TheDate)
            FileSystem.PrintLine(1, Arr(I).Total)
        Next
        FileSystem.FileClose(1)
    End Sub

    ' load the file info
```

```
    Public Sub LoadFile(ByVal FileName As String, ByRef Arr() As
SalesStruct, ByRef IC As Integer)
        FileSystem.FileOpen(1, FileName, OpenMode.Input,
OpenAccess.Read)
        Dim I As Integer
        IC = FileSystem.LineInput(1)
        ReDim Arr(0 To IC - 1)
        For I = 0 To Arr.Length - 1
            Arr(I).Item = FileSystem.LineInput(1)
            Arr(I).Price = FileSystem.LineInput(1)
            Arr(I).TheDate = FileSystem.LineInput(1)
            Arr(I).Total = FileSystem.LineInput(1)
        Next
        FileSystem.FileClose(1)

    End Sub
```

These two saves and load the information

```
    ' get total sum
    Public Function GetTotalSales(ByVal Arr() As SalesStruct) As Single
        Dim S As Single = 0
        Dim I As Integer
        For I = 0 To Arr.Length - 1
            S += Arr(I).Total
        Next
        Return S
    End Function
```

This last one finds the total. As you can see there is almost no difference in the code that is inside the module. It is exactly the same as the code you use in forms. Now if you open the main form of the application



This one contains a menu strip and a data grid view, with open files dialog and save file dialog. If you check the code of the form

```
    Private Sub AddToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
AddToolStripMenuItem.Click
        Dim SR As SalesStruct
        SR = General.ReadSalesInfo
        If SR.Flg Then
            SalesCount = SalesCount + 1
            ReDim Preserve Sales(0 To SalesCount - 1)
            Sales(SalesCount - 1) = SR
            DisplayArray(Sales, DGV)
        End If
    End Sub


    Private Sub RemoveSaleToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
RemoveSaleToolStripMenuItem.Click
        If DGV.SelectedRows.Count = 0 Then
            Exit Sub
        End If
        RemoveItemBasedOnName(DGV.SelectedRows.Item(0).Cells(0).Value,
Sales, SalesCount)
        DisplayArray(Sales, DGV)
```

```
    End Sub


    Private Sub ExitToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
ExitToolStripMenuItem.Click
        End
    End Sub


    Private Sub SaToolStripMenuItem_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles SaToolStripMenuItem.Click
        SFD.Filter = "*.txt|*.txt"
        If SFD.ShowDialog = Windows.Forms.DialogResult.Cancel Then
            Exit Sub
        End If
        SaveFile(SFD.FileName, Sales)
    End Sub


    Private Sub LoadToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
LoadToolStripMenuItem.Click
        If OFD.ShowDialog = Windows.Forms.DialogResult.Cancel Then
            Exit Sub
        End If
        LoadFile(OFD.FileName, Sales, SalesCount)
        DisplayArray(Sales, DGV)
    End Sub


    Private Sub FindTotalToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
FindTotalToolStripMenuItem.Click
        MsgBox("the total sales:" & GetTotalSales(Sales).ToString)
    End Sub
```

You can see the code is much smaller here, because it is just a call to the code in the module. In fact later on if you want to modify the user interface, the code of the module

is not affected. Also since the code is much smaller, it is easier for other programmers to understand you code and update it.

So to sum things up, modules:

1- Are vb files
2- Used to store functions/subroutines, and other vb coding
3- Makes your program easier to maintin
4- Makes your program easier to understand
5- You can use/not use them, it is up to you
6- You can use any number of modules in a vb project
7- Make it easy to port your code to another application
8- Helps you isolate the interface design from program logic.

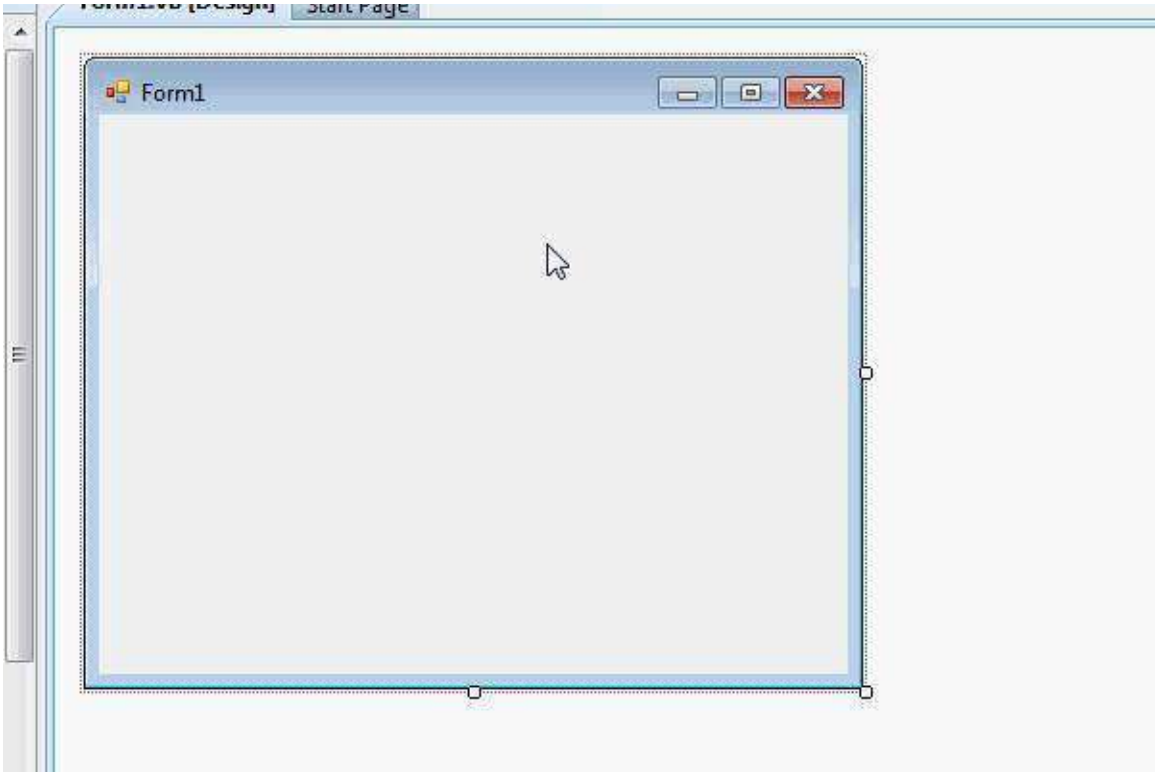
Chapter 17:Classes

Classes

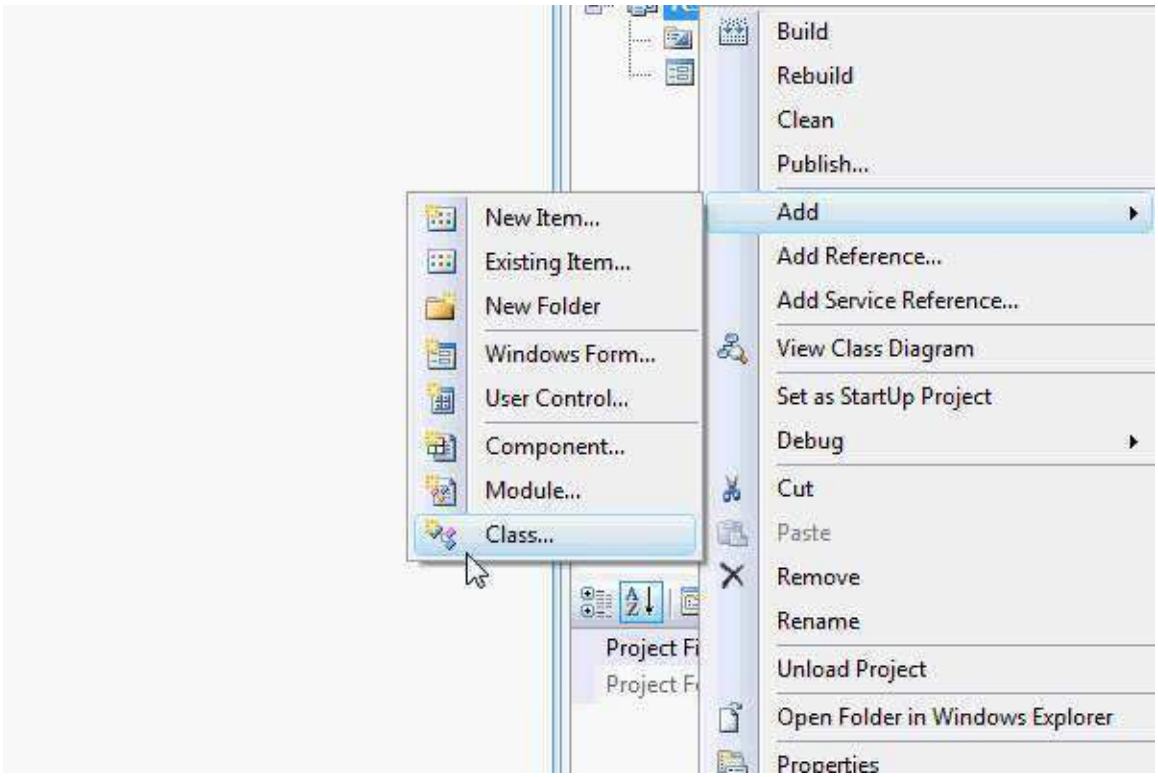
Previously we saw how to work with structures, and we examined how we can combine related information into one logical unit. Classes are very similar to structures, except that they allow you to combine the functions and subroutines that work on your information as well. It also has many other useful features that allows you to create and use frameworks to reuse the code.

To start understand classes we are going to develop a simple address book application. The application will allow you to store user information (name, address and telephone number).

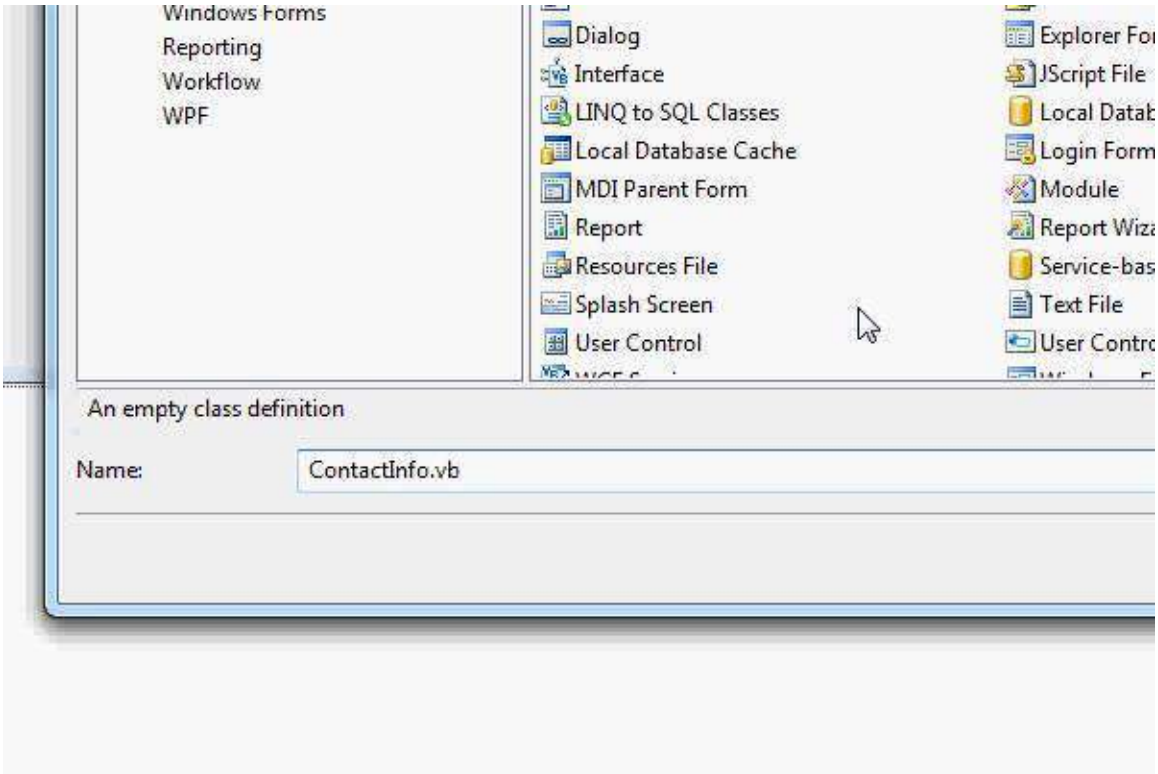
Start visual studio, and create a new project.



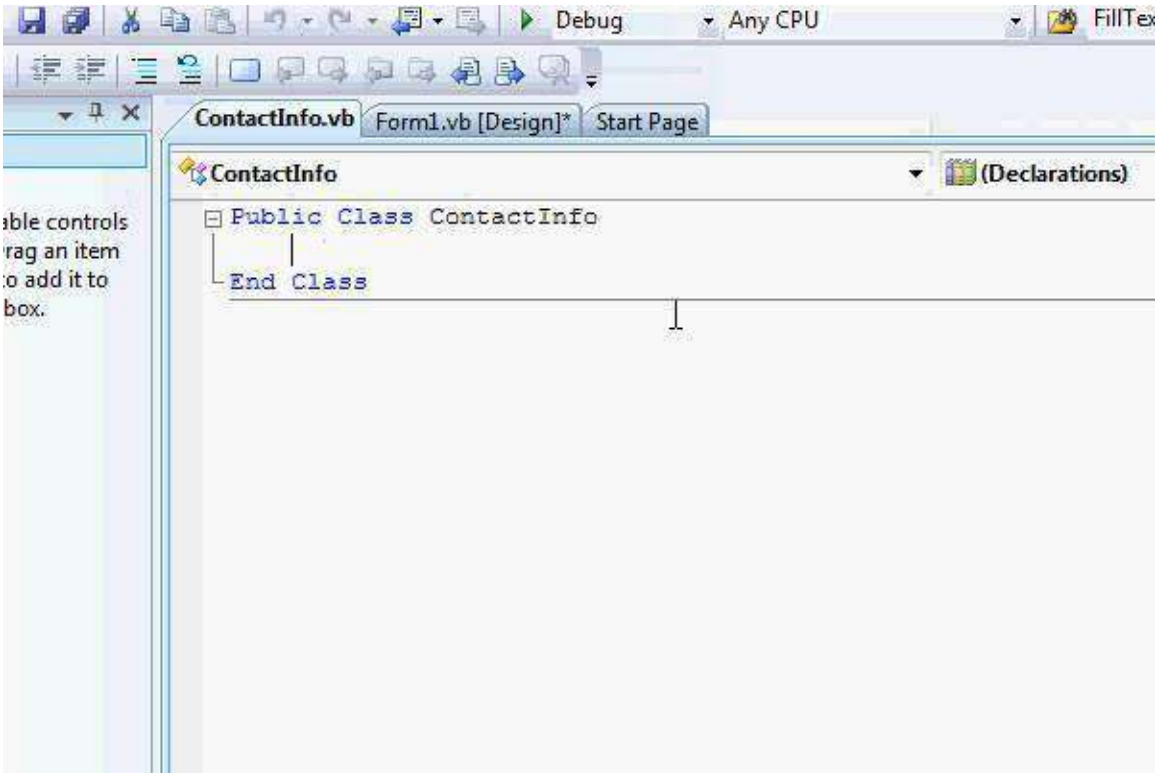
First thing we are going to do is to create a simple class that will describe the information for each contact. Usually each class is placed in a separate file. The process is similar to adding module, or adding another form to your project. Right click your project->Add->Class



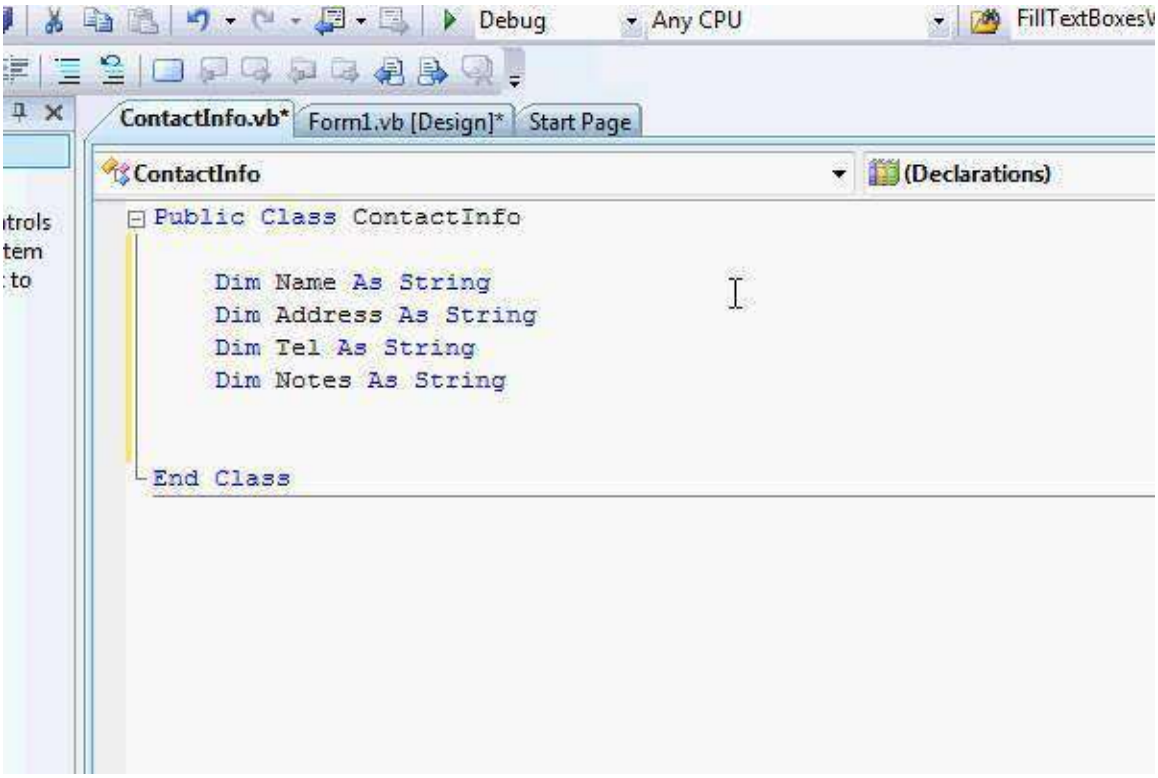
Next provide the name of the class (ContactInfo)



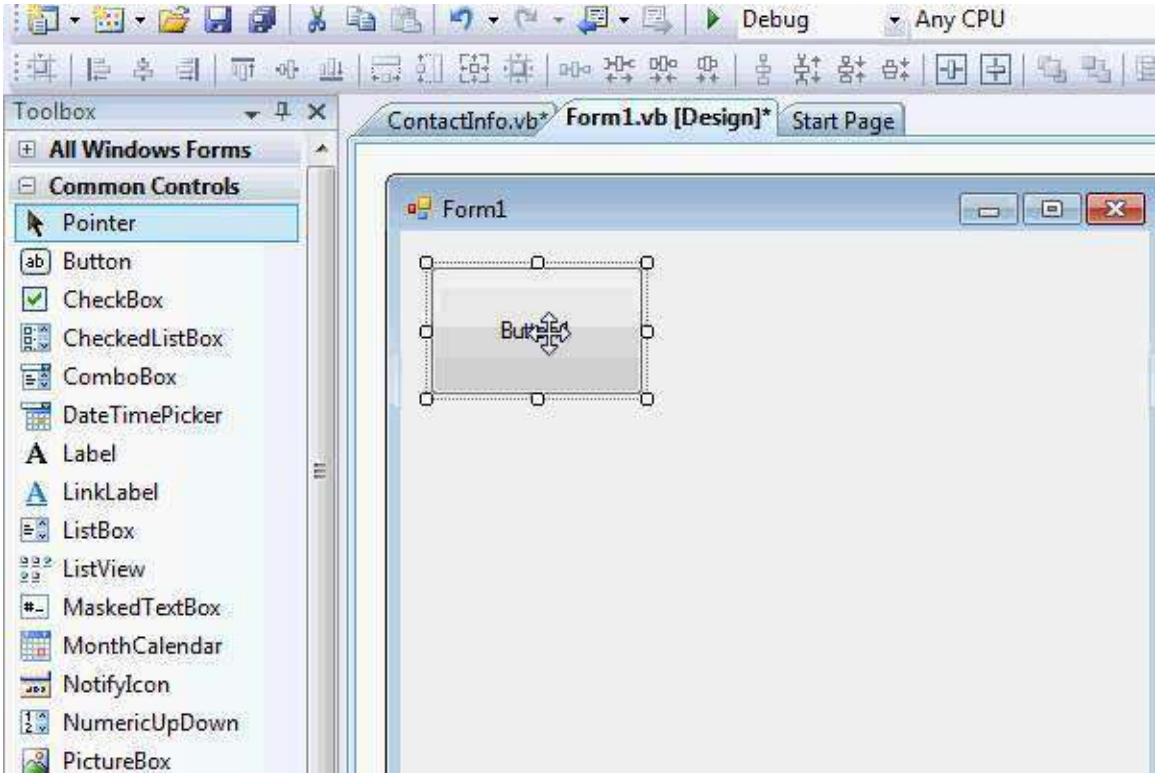
Next you will see the following:



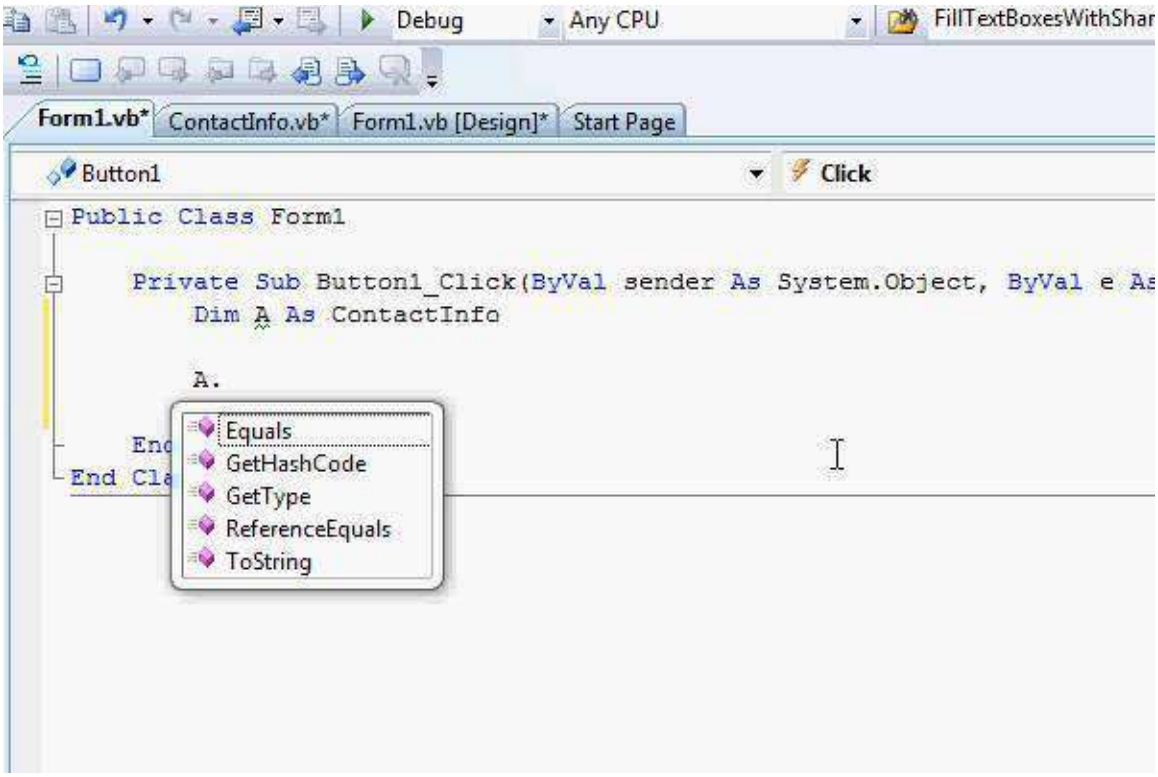
Now you should start writing the code for your class. The contact for each person should include person Name, Address and Telephone, therefore you define three variables as shown below:



To test this go to the form and place a button

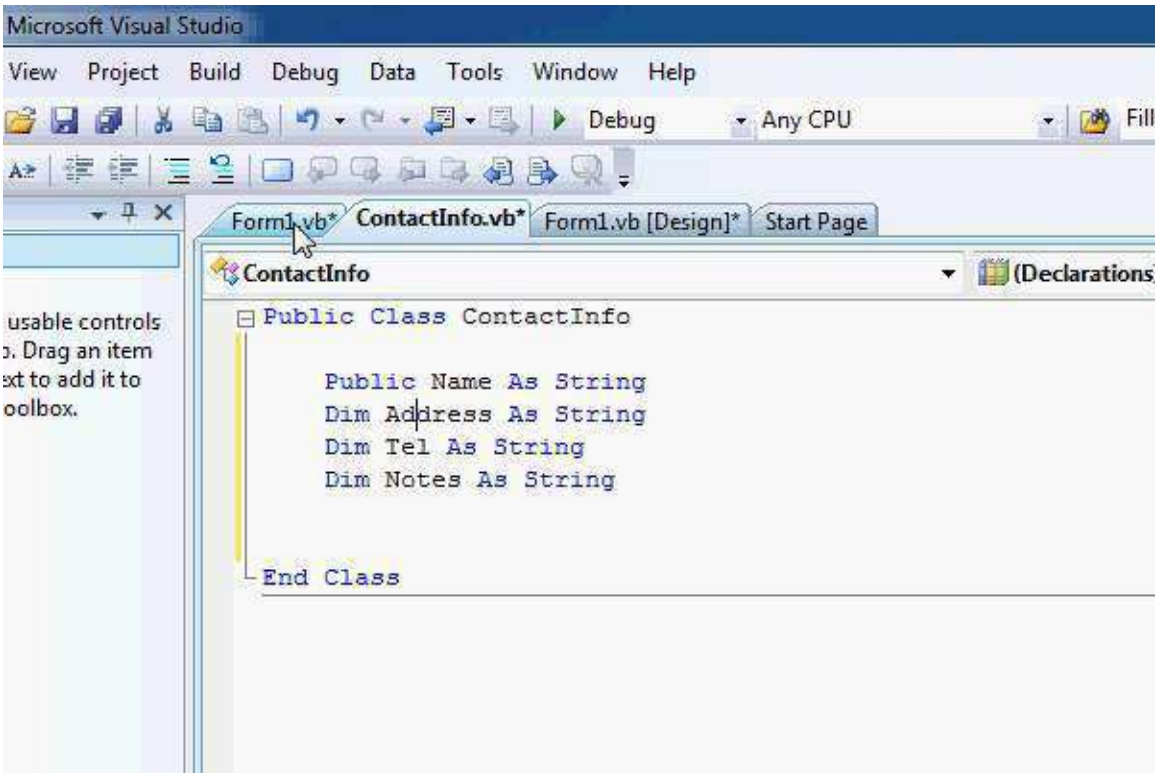


Next try to add the following to the code of the button

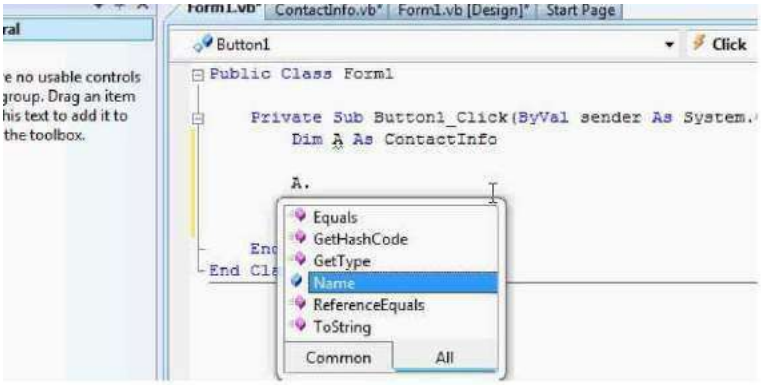


First you define a variable of type ContactInfo in a way similar to what you used to with structures, however, when you want to access the variables of the class you will find that the editor does not list them. Actually even if you write them manually you won't be able to run the program. This is because the variables within the class are protected from access outside the class code. This helps hiding complex code and the variables you don't want to be accessed by mistake.

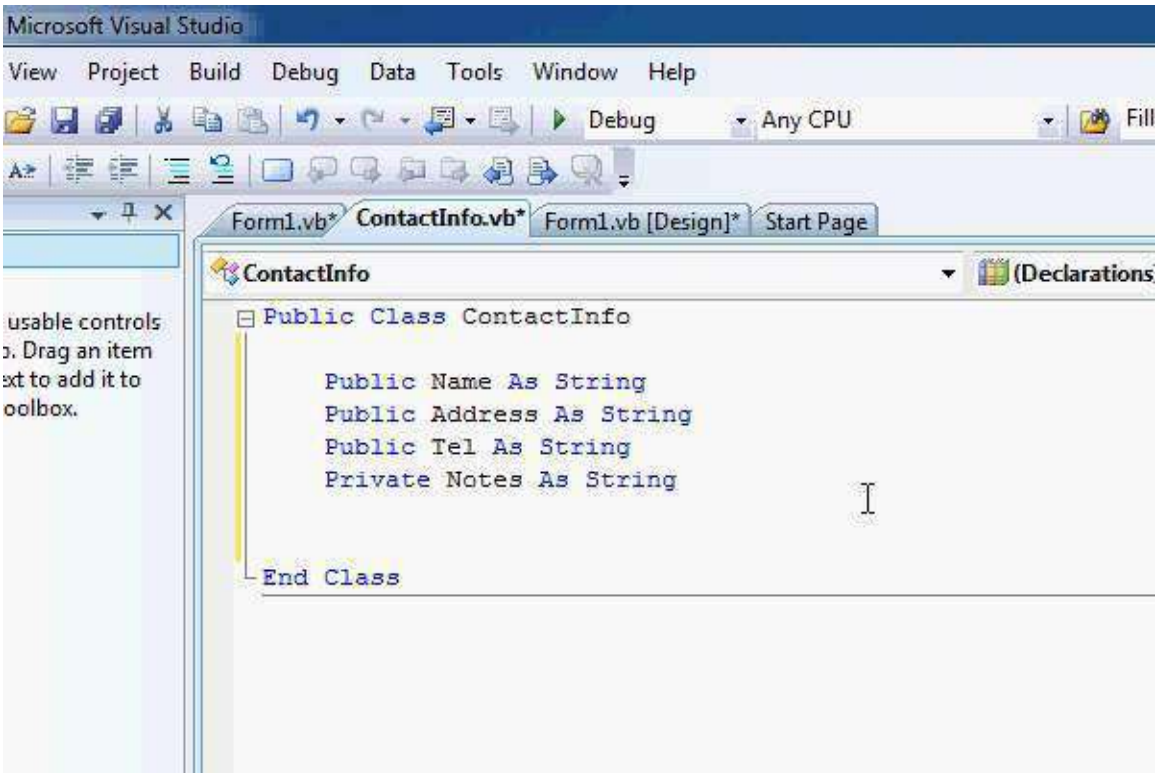
Now to make any variable accessible just change the Dim keyword in front of the variable to Public. This will grant this variable public access from any code within the project.



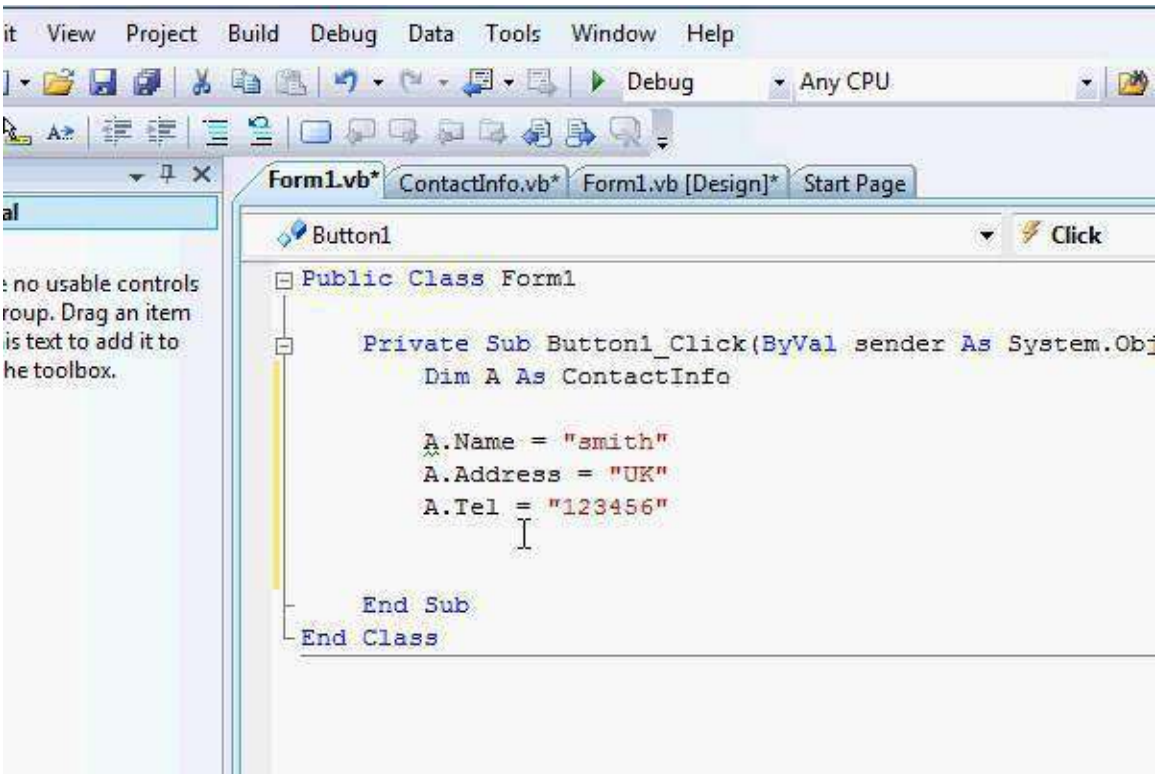
Now if you try to access the name property, you will see that the editor can detect that, and the property is listed when you press the (.) after the variable name.



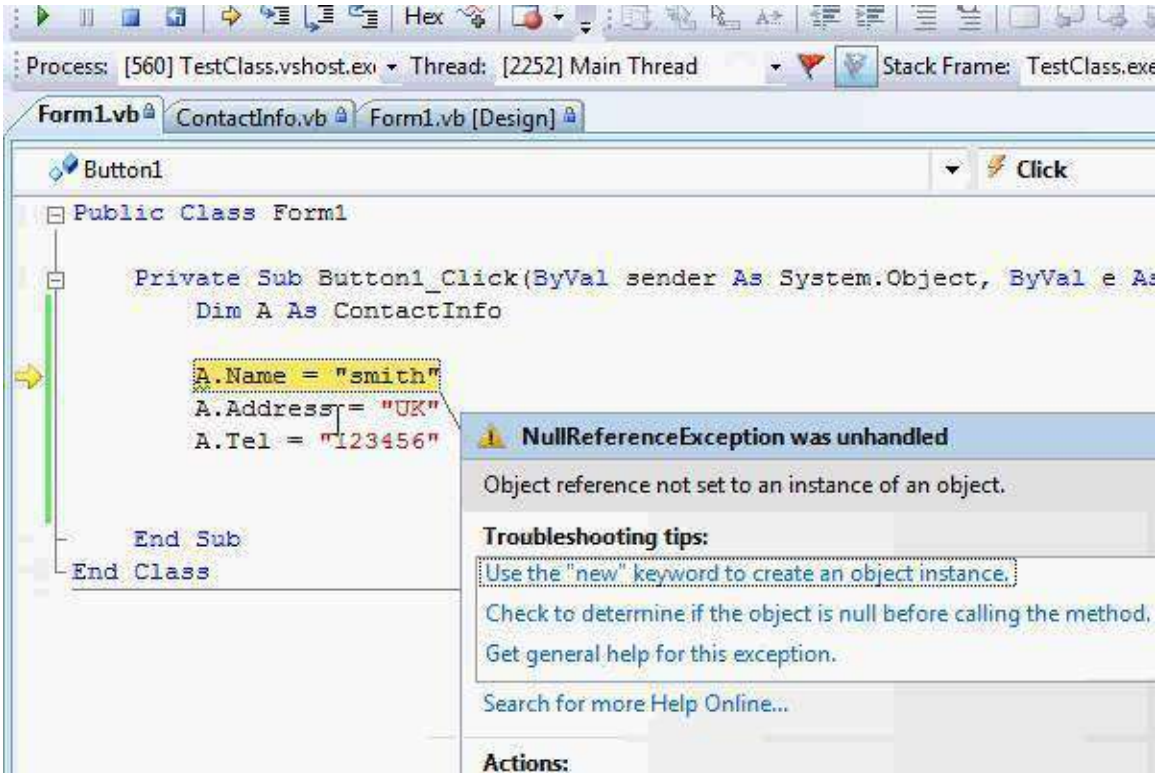
Make the Address and Tel variables within the class public similar to the way below:



Next add the following code to the event handler of the Button1



Now your code is completely correct from syntax point of view, however it will not run correctly. If you run the code and then hit the button, then you get the following error:



This brings us to the second difference of class from structure. The variable A in the example is just a pointer to where the actual data is stored in memory, and there is not memory resources allocated to store the name, address and tel values for A. This is why you are getting the error.

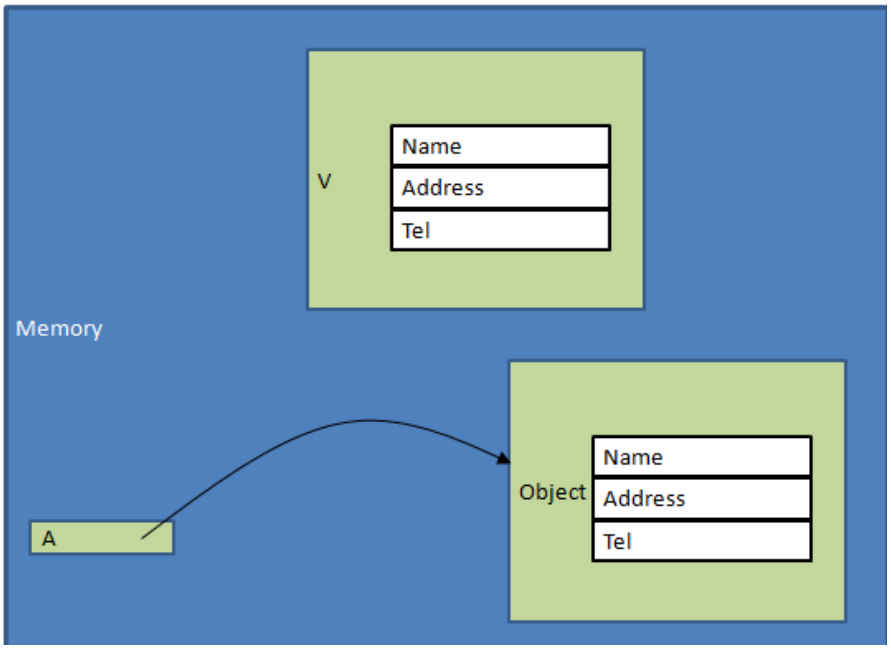
To clarify things more, Let us say we have a structure to store exactly the same information:

```
Structure ContactInfoStruct
    Dim Name As String
    Dim Address As String
    Dim Tel As String
End Structure
```

When you write

```
Dim V As ContactInfoStruct
```

Then what happens in memory is the following:



The variable V is allocated all the required memory resources. Unlike A, it only points to where the actual data are located in memory. So if there are no memory resources allocated, then A cannot point to them, and this is why you get the error. Now if you write:

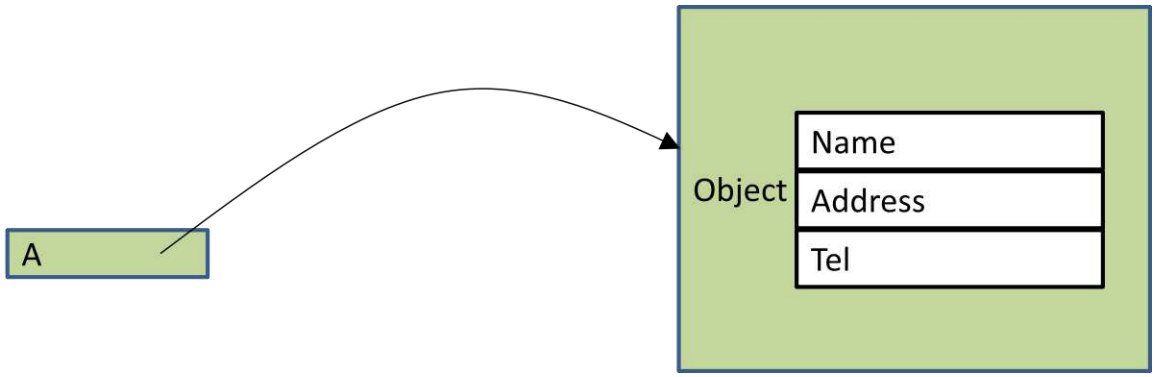
```
Dim A As ContactInfo
```

This creates a variable that points to no actual object (Nothing):

But if you write:

```
Dim A As New ContactInfo
```

Then an object is created, and A points to it:



Now use the New keyword, and test the code, you will see it runs without an error.

```
Dim A As New ContactInfo
```

Another method to do it is by using two steps:

```
Dim A As ContactInfo
A = New ContactInfo
```

This will have exactly the same effect. It is up to you to select which way to use. However in some cases you need to use the second format specially if you want to create and destroy the object linked by the same variable multiple times.
Now go to the class file and write down the following:

```
    Public Sub SetContactInfo(ByVal NME As String, ByVal Addr As String,
ByVal Telephone As String)
        Name = NME
        Address = Addr
        Tel = Telephone
    End Sub
```

This subroutine allows you to fill the variables in the class. It is a normal subroutine except for the Public keyword placed before it. This means that you can call this subroutine form any other code block. It is similar to using Public with variables. This is useful if you want to hide complex functions and subroutines from outside access and provide small number of function to use with your class. Now to test this subroutine, Modify the Button1 event handler to be like this:

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Dim A As ContactInfo
        A = New ContactInfo
        A.SetContactInfo("Smith", "UK", "123456")
    End Sub
```

As you can see it is used the same way variables are accessed. You write the variable name (in this case A), followed by dot (.), followed by the function/subroutine (SetContactInfo). This is interpreted as call the function (SetContactInfo) and use the fields/attributes of A. If you the subroutine code:

```
        Name = NME
        Address = Addr
        Tel = Telephone
```

This is translated to:

```
        A.Name = NME
        A.Address = Addr
        A.Tel = Telephone
```

Now if you are using another object:

```
        Dim B As ContactInfo
        B = New ContactInfo
        B.SetContactInfo("Michel", "US", "123456")
```

The subroutine call will be interpreted as:

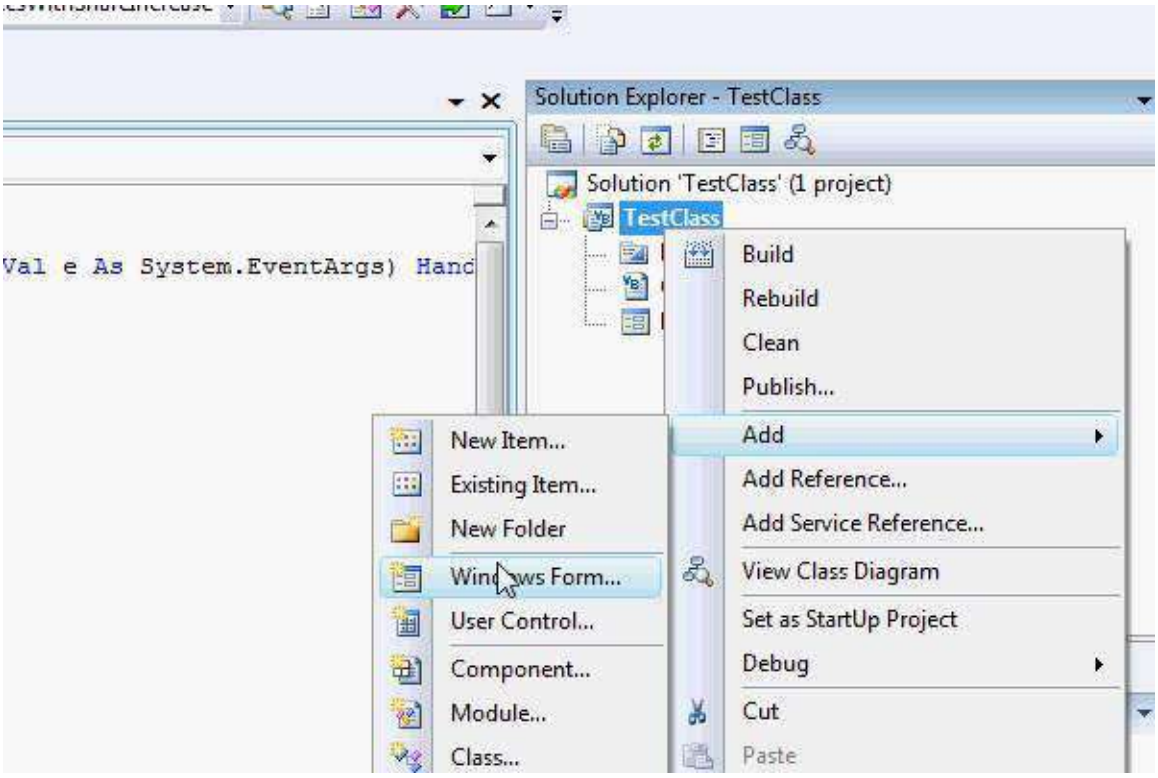
```
        B.Name = NME
        B.Address = Addr
        B.Tel = Telephone
```

and so on.

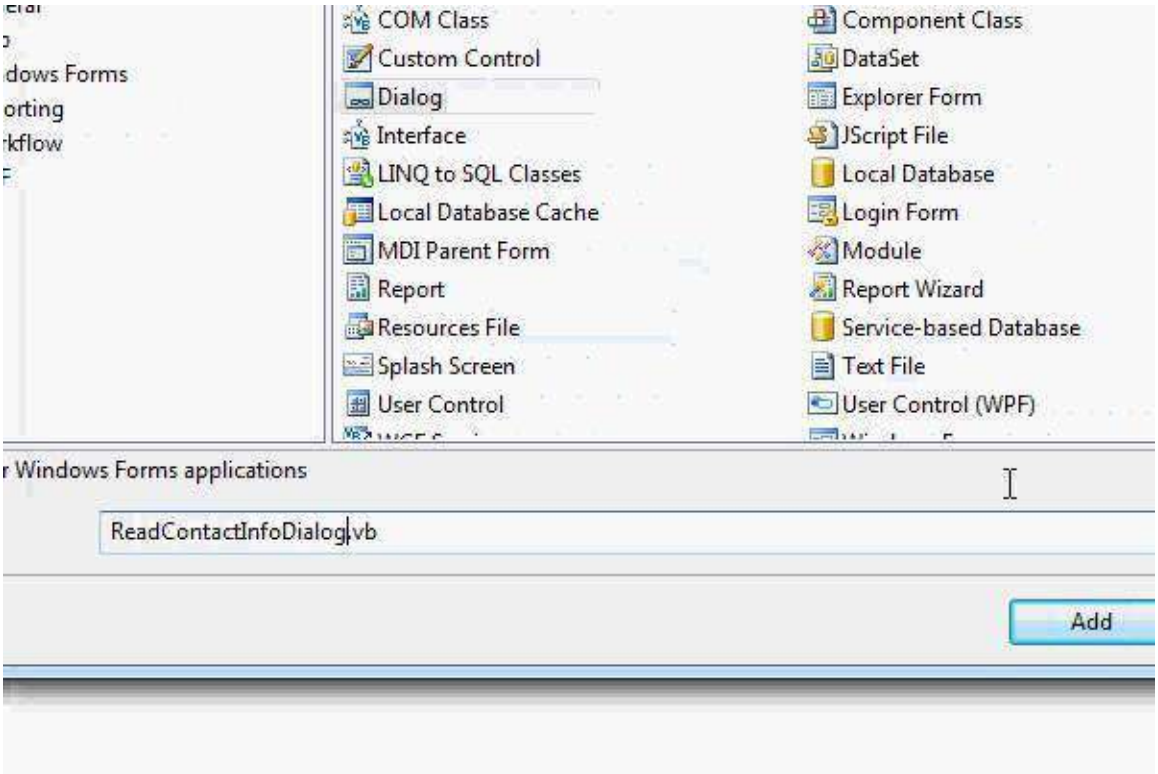
Now write down the following code in the event handler and run it:

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Dim A As New ContactInfo
        Dim B As New ContactInfo

        A.SetContactInfo("Smith", "UK", "123456")
        B.SetContactInfo("Michel", "US", "456789")


        MsgBox(A.Name)
        MsgBox(A.Address)
        MsgBox(A.Tel)

        MsgBox(B.Name)
        MsgBox(B.Address)
        MsgBox(B.Tel)

    End Sub
```

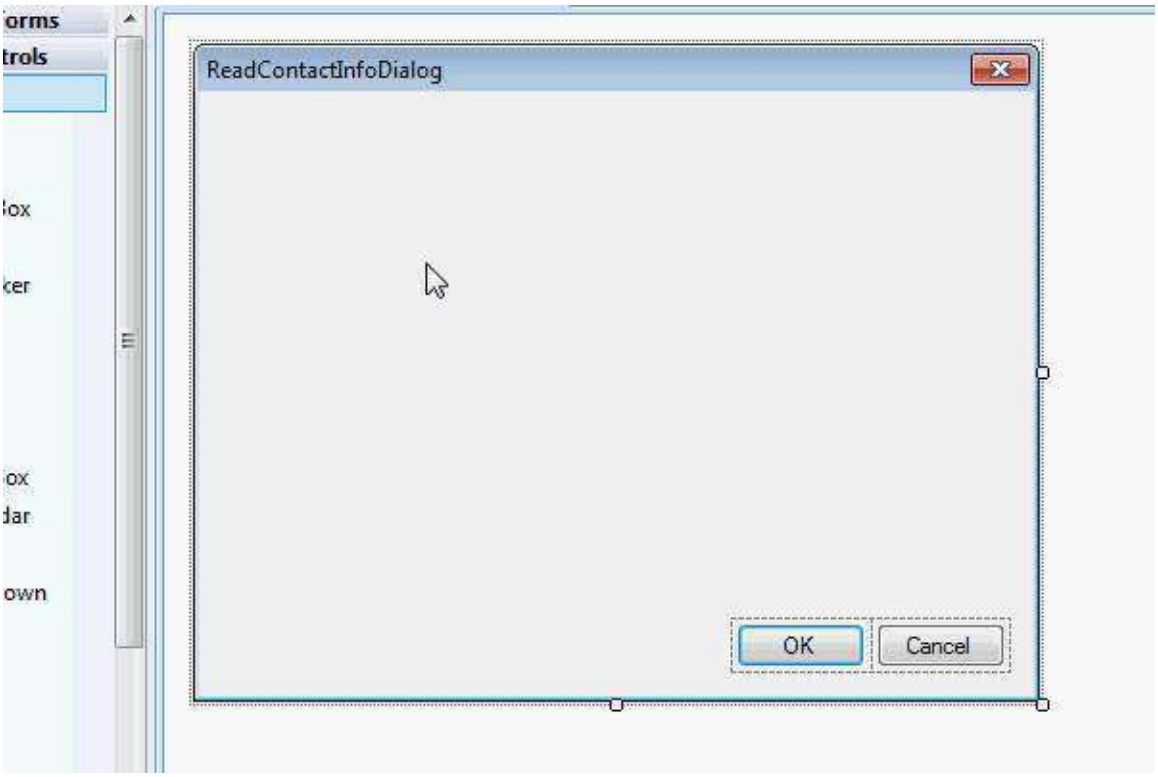
As you can see the code is easier to understand, and you don't have to fill the fields/attributes of the contact one by one. Now we will improve the way we enter the data by reading the information from a dialog. Right click your project and select Add->Windows Form



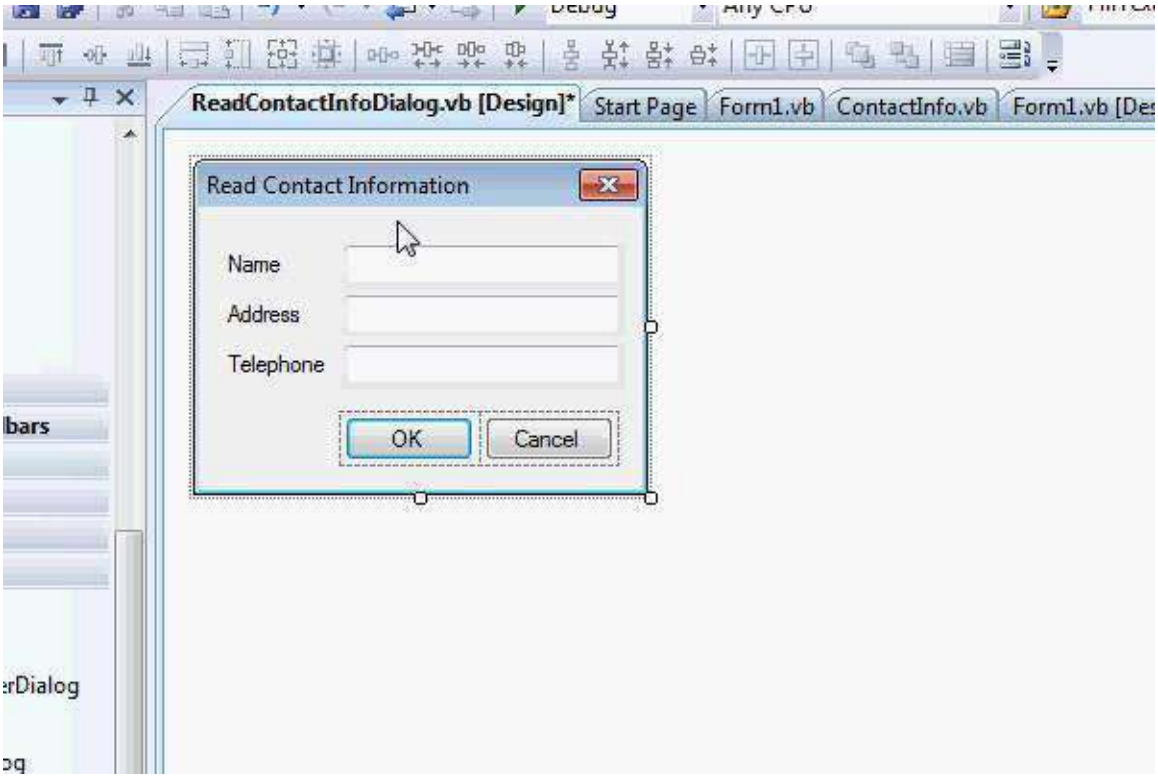
Select Dialog, and assign the name ReadContactInfoDialog:



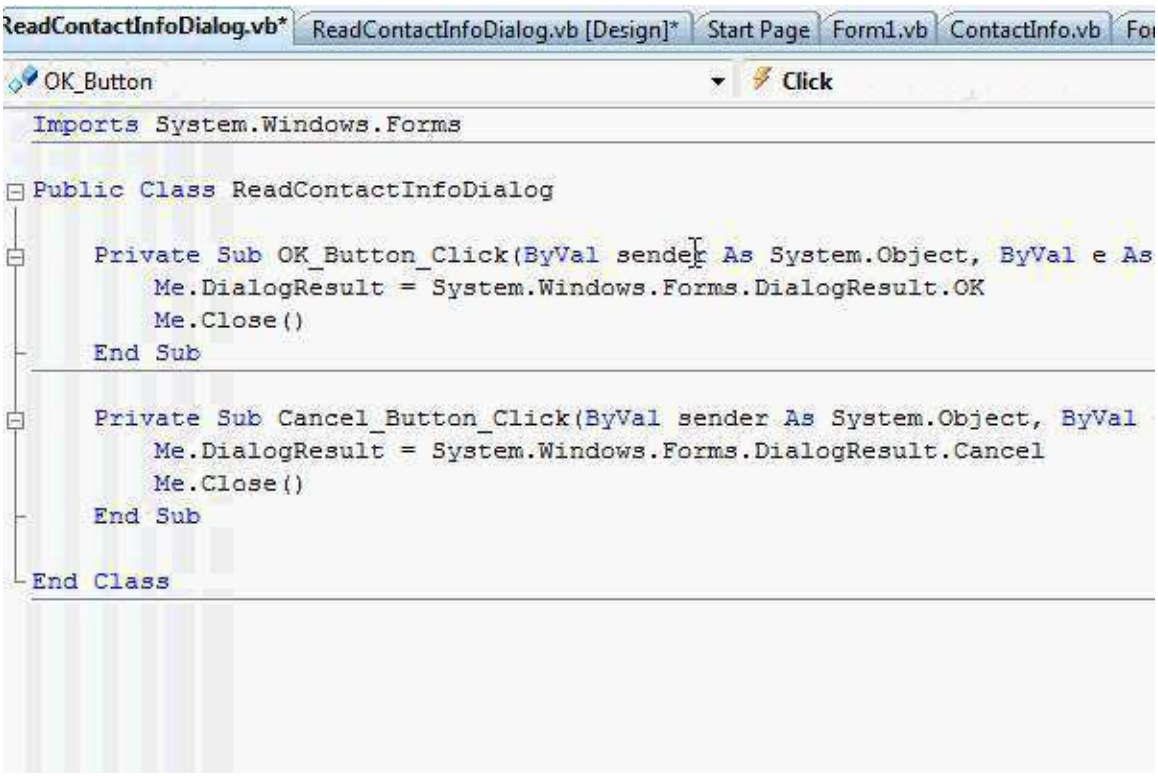
Then select Add, the dialog design appears.



Add three labels, and three text boxes, and make the dialog look like this:



If you check the code of the OK & Cancel buttons, you will find that it is already written. This code is the default behavior for a dialog, so leave it as it is.



Next we will add a subroutine to read contact information. Go to the class file and write the following:

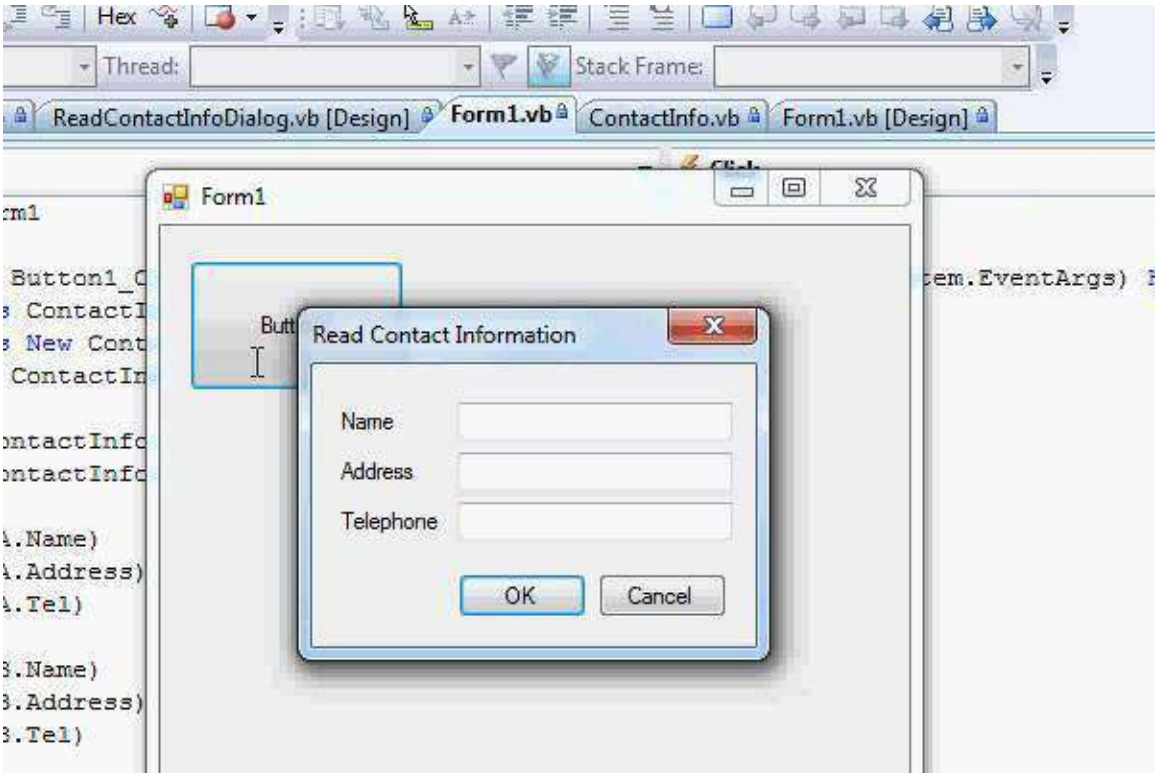
```
Public Sub ReadContactInfo()

    ReadContactInfoDialog.TextBox1.Text = ""
    ReadContactInfoDialog.TextBox2.Text = ""
    ReadContactInfoDialog.TextBox3.Text = ""

    If ReadContactInfoDialog.ShowDialog = DialogResult.Cancel Then
        Exit Sub
    End If

    Name = ReadContactInfoDialog.TextBox1.Text
    Address = ReadContactInfoDialog.TextBox2.Text
    Tel = ReadContactInfoDialog.TextBox3.Text

End Sub
```

The first part clears the text boxes from all previous input. The if statement part checks if the user hit the cancel button, and exits the subroutine if so. If not, the execution

continues to the last part, there the content of the text boxes are copied into the variables of the class. To test it modify the code of the Button1 for the main window (Form1) to be like this:

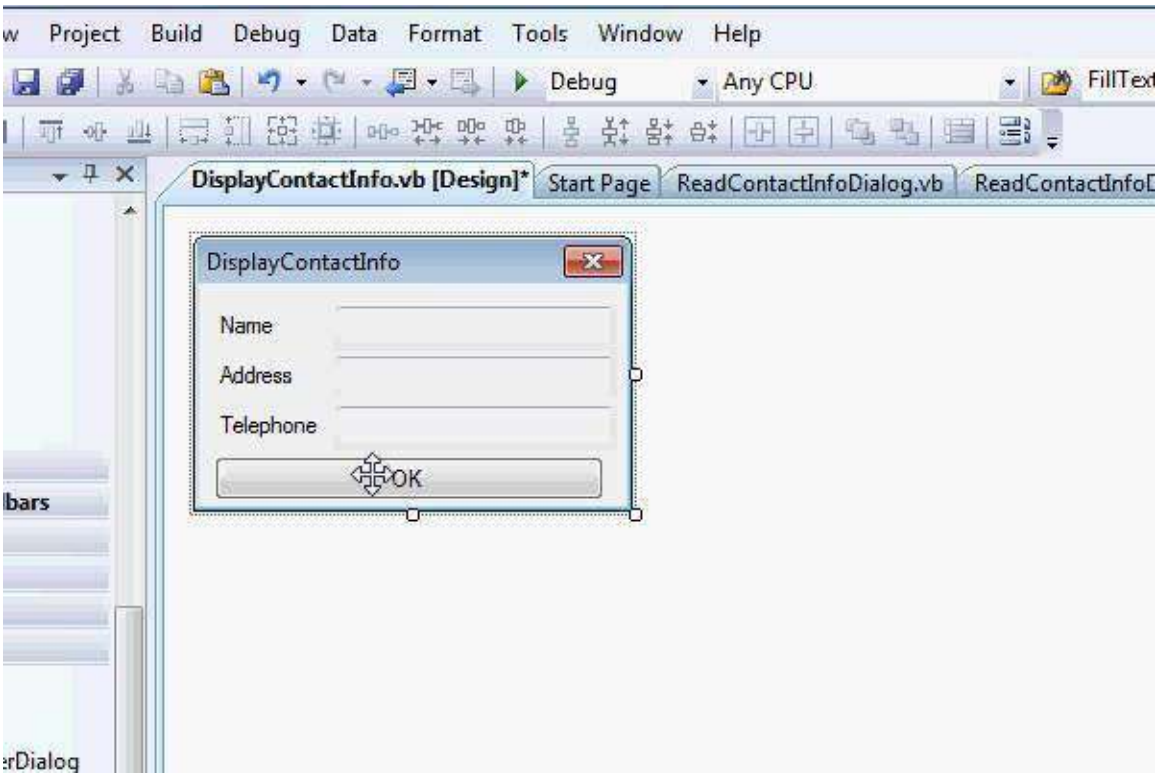
```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Dim A As New ContactInfo
        Dim B As New ContactInfo

        A.ReadContactInfo()
        B.ReadContactInfo()


        MsgBox(A.Name)
        MsgBox(A.Address)
        MsgBox(A.Tel)

        MsgBox(B.Name)
        MsgBox(B.Address)
        MsgBox(B.Tel)

    End Sub
```

Now run the code, and hit the button, you should see something like this:



Enter the information and hit OK. Another window appears, fill the information of the second contact and hit OK. After that you should be able to see the details of each contacts appear in separate message boxes.

Instead of using the message box to display the contact information, we will create another dialog to display such info. Just Add another dialog to the project as we did before and name it DisplayContactInfo. And make it look like the following:



Make sure to only remove the cancel button, and keep the OK button there. Also make sure that all textboxes are read only. Go next to the class file and add the following subroutine:

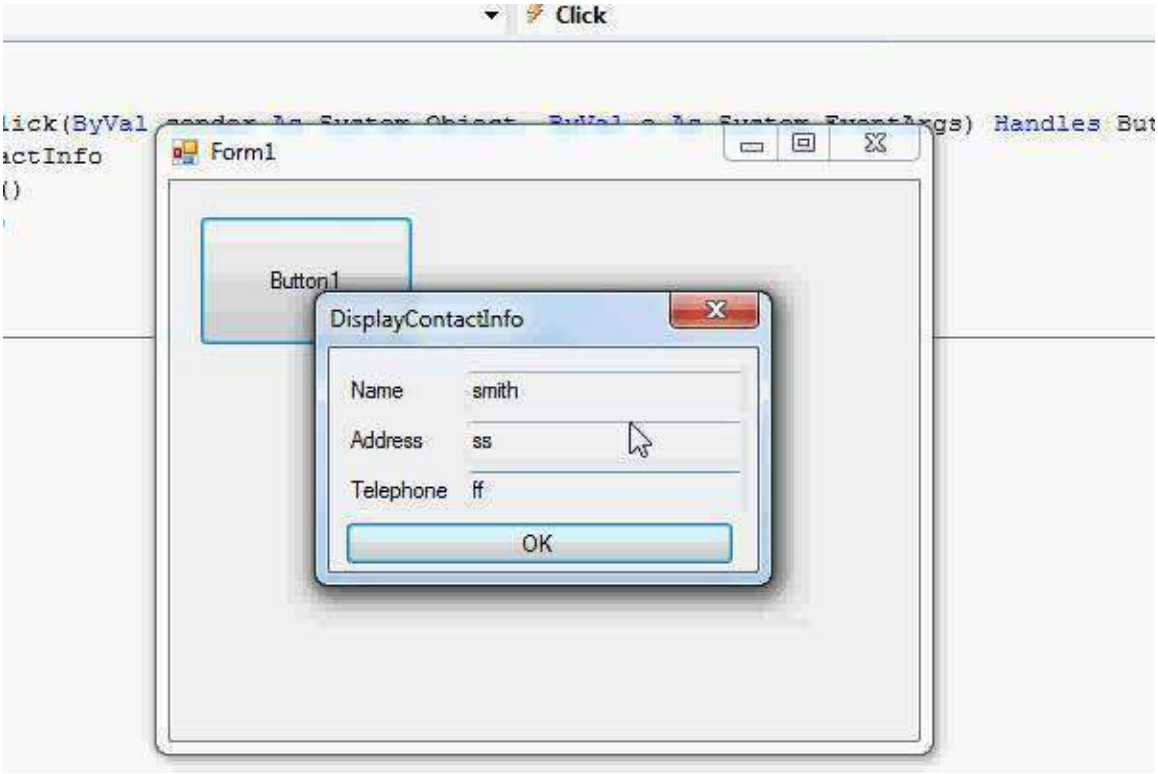
```
    Public Sub DisplayContact()
        DisplayContactInfo.TextBox1.Text = Name
        DisplayContactInfo.TextBox2.Text = Address
        DisplayContactInfo.TextBox3.Text = Tel
        DisplayContactInfo.ShowDialog()
    End Sub
```

This is much smaller code since it just displays the information of the object. To test that, update the code of Button1 in Form1

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Dim A As New ContactInfo
        Dim B As New ContactInfo

        A.ReadContactInfo()
        B.ReadContactInfo()
```

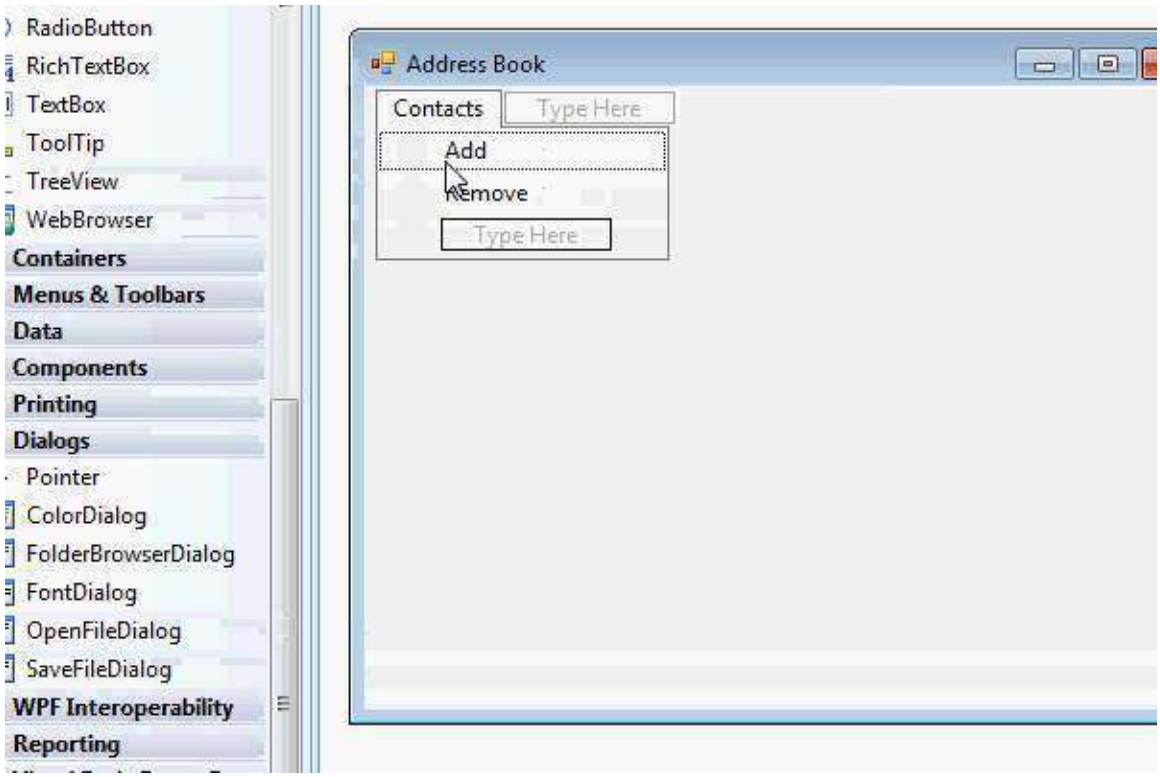
```
        A.DisplayContact()
        B.DisplayContact()

    End Sub
```

Run the code and you will see that you can display the information in the form:

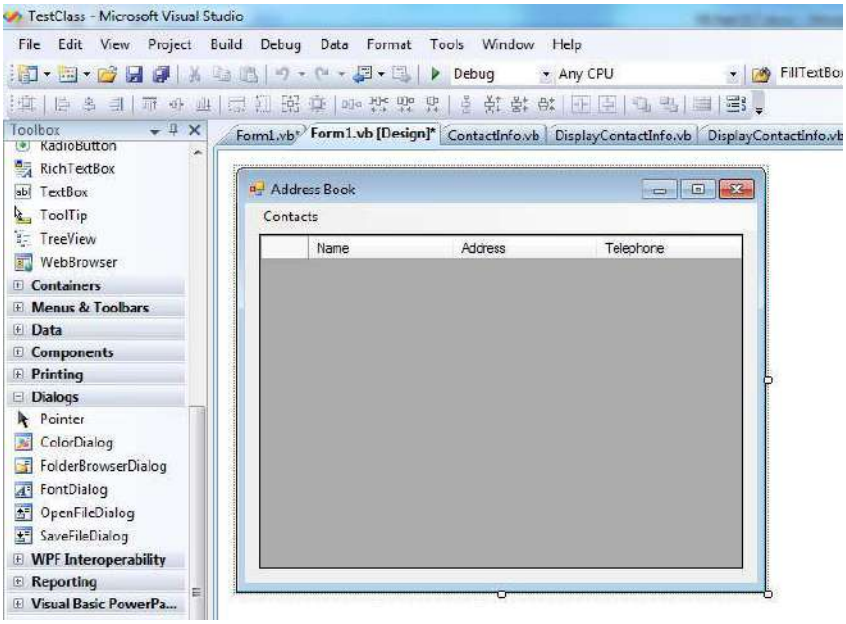


As you can see the code in Button1_Click subroutine is very straightforward and easy to understand. You don't have to worry about the internal details of the class. All you need is to break your problem/your program into a number of logical units/classes each has its own data and functions, and then you combine them together to solve the main problem. Classes makes such thing easier to do.

Now our simple class is almost ready, so we are starting to create the main user interface now. Remove the Button1 from the Form1 window and add a menu strip control. Create the menu entries shown below:



Also add a data grid view, and call it DGV, add three columns to it (one for name, one for address, and one for tel). Disable adding, editing and deletion of rows. You should have something similar to the following:



Double click the form and the editor opens, add the following code after Class Form1

```
Dim ContactList(0 To 999) As ContactInfo
```

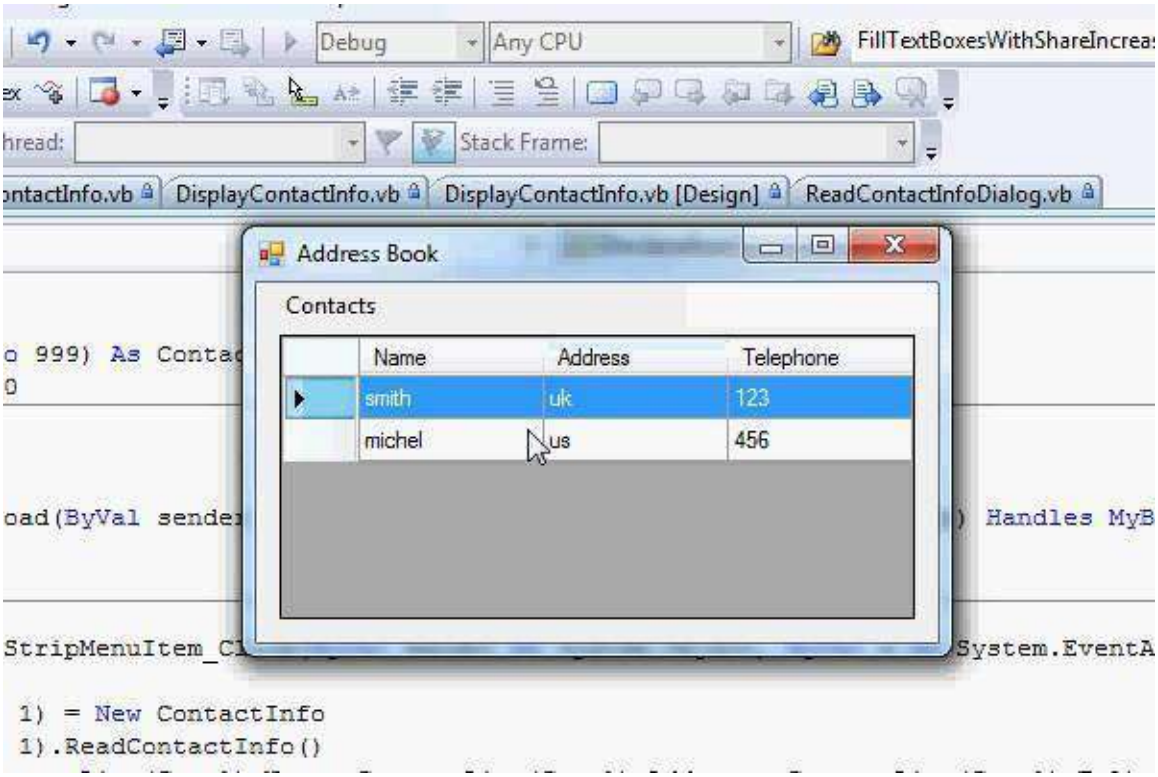
```
    Dim C As Integer = 0
```

The ContactList is an array of type contact info. Each element of this array can point to an object of type ContactInfo, but when the array is created it is pointing to Nothing. C is used to tell how many objects are there in the array. When the program starts the number of elements is Zero.

Next add the following code to the Add menu item:

```
    Private Sub AddToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
AddToolStripMenuItem.Click
        C = C + 1
        ContactList(C - 1) = New ContactInfo
        ContactList(C - 1).ReadContactInfo()
        DGV.Rows.Add(ContactList(C - 1).Name, ContactList(C -
1).Address, ContactList(C - 1).Tel)
    End Sub
```

This subroutine will add new contact, read the information of that contact, and then update the display. Try this out and you should be getting something like this:



Now the remove code should be like this:

```
    Private Sub RemoveToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
RemoveToolStripMenuItem.Click
        If DGV.SelectedRows.Count = 0 Then
            Exit Sub
        End If

        Dim I As Integer
        Dim N As String
        N = DGV.SelectedRows(0).Cells(0).Value
        For I = 0 To C - 1
            If ContactList(I).Name = N Then
                DGV.Rows.Remove(DGV.SelectedRows(0))
                Dim J As Integer
                For J = I + 1 To 999
                    ContactList(J - 1) = ContactList(J)
                Next
                C = C - 1
                Exit Sub
            End If
        Next
    End Sub
```

Notice that we are removing the contact from the display and from the array itself. Try adding and removing few contacts and see how it works. So this concludes the chapter. There will be more about classes in the next chapter. However there is some important things that you must keep in mind. A variable of a class is a pointer only. A good example to understand this is if you write the following code:

```
        Dim A As New ContactInfo
        Dim B As ContactInfo
        A.Name = "Smith"
        B = A
        B.Name = "John"
```

In the end of execution of such code, both A and B will have John as the name value. Any change to A or B will affect the other one because simply they both point to the

same location in memory (point to the same object in memory). But if A & B are structures:

```
Dim A As ContactInfoStruct
Dim B As ContactInfoStruct
A.Name = "Smith"
B = A
B.Name = "John"
```

Then A will be independent of B and changes in A will not affect B and vice versa.

Chapter 18: Classes Initialization and Finalization

Classes Initialization and Finalization

The previous chapter showed how to create a class, and how to add methods, and attributes to it. Today we see how to initialize the objects using the New method. First open the previous class example "testclass". Add a new class to the project, and call it: ContactList. This class will be used to store the contact information in the array and manage it . In the class file add the following code:

```
    Dim ContactArr() As ContactInfo            ' the array of object,
all elements points to nothing
    Dim C As Integer                           ' the number of objects
in the array
```

These are used to store the contact information, and the number of elements in the array used. Next add the following method:

```
    Public Sub AddNewContact()
        C = C + 1                              ' the number of objects
increases by one
        ContactArr(C - 1) = New ContactInfo    ' create the object
        ContactArr(C - 1).ReadContactInfo()    ' read the information
    End Sub
```

This one adds a new contact, then add:

```
    Public Sub RemoveContact(ByVal Name As String)
        ' search for the contact
        For I = 0 To C - 1
            If ContactArr(I).Name = Name Then


                ' next remove the contact from the array by shifting the
other objects
                Dim J As Integer
                For J = I + 1 To 999
                    ContactArr(J - 1) = ContactArr(J)
                Next

                ' the number of elements reduces by one
```

```
                C = C - 1

                ' exit the block
                Exit Sub
            End If
        Next

    End Sub
```

Which will remove a contact based on name. Also, add the following method to fill the data grid view:

```
    Public Sub FillDGV(ByVal DGV As DataGridView)
        ' clear the data grid view
        DGV.Rows.Clear()

        Dim I As Integer

        ' loop over all the contacts
        For I = 0 To C - 1
            ' add contact information
            DGV.Rows.Add(ContactArr(I).Name, ContactArr(I).Address,
ContactArr(I).Tel)
        Next

    End Sub
```

Now comes the constructor, write down the following:

```
    Public Sub New()
        ' first constructor, set the number of elements to zero, and set
array size to 1000
        C = 0
        ReDim ContactArr(0 To 999)
    End Sub
```

The name of this method is: New, and by default, when the compiler sees this, it knows that this method should be called automatically as soon as the object is created. So basically this method tells the computer to set the value of the counter C to zero, and

make the array capable of storing 1000 objects as soon as the ContactList object is created. To test this, Go to the form, and modify the code to be like this:

```
Public Class Form1

    Dim OBJ As ContactList


    Private Sub AddToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs)  Handles
AddToolStripMenuItem.Click

        OBJ.AddNewContact()
        OBJ.FillDGV(DGV)

    End Sub

    Private Sub RemoveToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
RemoveToolStripMenuItem.Click

        ' check if no rows are selected, if so no need to execute
further code, exit the subroutine
        If DGV.SelectedRows.Count = 0 Then
            Exit Sub
        End If

        Dim N As String

        ' get the selected name, it is the first column (cell zero)
        N = DGV.SelectedRows(0).Cells(0).Value

        OBJ.RemoveContact(N)
        OBJ.FillDGV(DGV)

    End Sub


    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

        OBJ = New ContactList()
```

```
        OBJ.FillDGV(DGV)

    End Sub

End Class
```

Check out the code, and specifically the Form1_Load subroutine. When the line:

```
        OBJ = New ContactList()
```

is executed, the New subroutine is called directly. You don't have to do the call yourself, it is automatic. Also the constructor is executed only once.

So basically the constructor helps you prepare your object before using it. To make sure the constructor is being called, try to add a **MsgBox** call in the New method and see how it works.

You can actually create a number of different constructors, and later on you can choose which one to use based on the parameters you pass to it. For example, let us add another constructor to our class:

```
    Public Sub New(ByVal NoOfReads As Integer)
        ' second constructor, set number of elements to zero, and set
array size to 1000
        C = 0
        ReDim ContactArr(0 To 999)

        ' add the contacts
        Dim I As Integer
        For I = 0 To NoOfReads - 1
            Me.AddNewContact()
        Next
    End Sub
```

This constructor allows you to read a number of contacts as soon as you initialize the object without the need of going to the menu and select add contact. In order to call it, simply use it like this:

```
        OBJ = New ContactList(3)
```

When the compiler sees the parameters (3), it searches for the constructor that accepts an integer as a parameter and calls it. You can create as many constructors as you need. The important thing is that the constructor name is always New, and each constructor should have different parameters (either in number or in data type to help the compiler distinguish them). Try the new constructor, and see how it works.

The last thing is the destructor. A destructor or finalizer is a method that is called when an object is destroyed, i.e. its resources are returned into memory. Try adding the code below:

```
Protected Overrides Sub Finalize()
    ' this is how to terminate a class
    Dim I As Integer
    For I = 0 To C - 1
        ContactArr(I) = Nothing
    Next


    MyBase.Finalize()
End Sub
```

Don't worry about the **MyBase**, or **Protected**, or the **Overrides** keywords for now, we will check these in later tutorials, but for now, just keep in mind that this one is being called when the object is destroyed. As you can see what we are doing here is we are looping over all the contactinfo objects and set them to nothing (which means we don't need them anymore, and we want them to be destroyed). Then after that we destroy the object. Try this code out, and see what happens when you place an MsgBox in this method.

Chapter 19: Classes and Inheritance

Classes and Inheritance

In the previous two tutorials, the definition of classes, methods, and their initialization is discussed. This tutorial is about how to perform inheritance. The same example used in the last tutorial is being used here as well.

In many cases you want to take an existing class and extend its functionality. In our previous example the class **ContactList** has **ContatArr()** which is an array used to store the contacts, and the counter **C** which is used to tell how many elements we are using in the array. It also has methods to add a contact, remove a contact, and display the contacts in a **DataGridView**. What we want here is to create a new class that has the same methods and properties as **ContactList**, and also has a **Sort** method which allows you to sort the contacts by name.

To do so simply add another class to your project, and call it **ContactsWithSort**. The first line of code in the class should be:

```
Inherits ContactList
```

The keyword **Inherits** here tells the complier that the class has behave in the same way as **ContactList**. In other words it is like copying the code of ContactList and pasting it to the new class (for now you can think about it like this, it makes things easier).

Now let us try to add the new method to the class…

```
Public Sub Sort()       ' this class has another method called sort.
    Dim I As Integer
    Dim F As Boolean
    Dim Contact As ContactInfo

    Do
        F = False

        For I = 0 To C - 2
            If ContactArr(I).Name > ContactArr(I + 1).Name Then
                F = True
                Contact = ContactArr(I)
                ContactArr(I) = ContactArr(I + 1)
                ContactArr(I + 1) = Contact
            End If
```

```
            Next

        Loop While F

    End Sub
```

Now if you try to run the program (even though you did not use the Sort method or the new class itself) you will get an error. The error is for using **C** and **ContactArr**. The error says that these variables are private. This brings up the issue of the accessibility of variables.

When you define a variable in a class you can set its accessibility level to the following:

Public : this means that the variable can be access inside or outside the class.

Private: this means that the variable can be accessed only inside the original class it is created in.

Protected: this means that the variable can be accessed only in the class and all inherited classes.

So let us check this using the following example:

```
Public Class test
    Dim A As Integer
    Private B As Integer
    Public C As Integer
    Protected D As Integer

    Public Sub SetA(ByVal I As Integer)
        A = I
    End Sub

    Public Sub SetB(ByVal I As Integer)
        B = I
    End Sub

    Public Sub SetC(ByVal I As Integer)
        C = I
    End Sub

    Public Sub SetD(ByVal I As Integer)
        D = I
```

```
    End Sub

End Class
```

In the example, **A** is treated as private. So if you add this method to the class:

```
Public Sub SetA(ByVal I As Integer)
    A = I
End Sub
```

It works perfectly fine. However, if you add the following code into a form or module:

```
Dim Q As New test
Q.A = 10
```

This would trigger an error because **A** should only be accessed from within the class. Now let us check **B** which is private. If this is a method in the class, then it works.

```
Public Sub SetB(ByVal I As Integer)
    B = I
End Sub
```

But if you add the following code into any place other than the class test, you get an error.

```
Dim Q As New test
Q.B = 10
```

So it works exactly like private. Next let us try to work with **C** which is Public.

```
Public Sub SetC(ByVal I As Integer)
    C = I
End Sub
```

This obviously works fine since it is in the same class (test). If you write the following code in any other place other than the class test, then it works perfectly fine.

```
Dim Q As New test
Q.C = 10
```

This works because the variable C here is public which means it can be accessed from any other place. Now let us check the last one D which is protected. The method within the class again has no problem

```
Public Sub SetD(ByVal I As Integer)
    D = I
End Sub
```

If you want to access the variable D from outside the class it is treated like private, but it has some special treatment, which we will see later.

```
Dim Q As New test
Q.D = 10
```

So this triggers an error. Now let us go back to our example and see why we can't access the variable **C** and **ContactArr**. We used (Dim) for these two which means they are treated like private. As we have seen before that private variables in a class can not be accessed from outside the class itself. So we want to make them accessible. Making these variables public means that they will be accessed from any part of the project, which is not a good idea. If you change these variables' visibility to protected, then the classes inherited from them will be able to access these. An access from any other location is denied. To test this try to create a class test2 inherited from test.

```
Public Class test2
    Inherits test

    Public Sub SetAll()
        A = 10      ' error
        B = 20      ' error
        C = 30      ' correct
        D = 40      ' correct
    End Sub
End Class
```

Here A is not accessible in this class simply because it is private in the original class. B is the same so it causes the same problem. C is public in class test, so it is accessible here

and everywhere else. D is protected so it is accessible in test2. The table below summarizes how these work:

Accessibility	Base Class	Inherited Class	Outside the Class
Dim	Accessible	Not Accessible	Not Accessible
Private	Accessible	Not Accessible	Not Accessible
Public	Accessible	Accessible	Accessible
Protected	Accessible	Accessible	Not Accessible

So going back to our example, set each of **C** and **ContactArr** in **ContactList** class to **protected**. You will see the code now is correct.

Next, modify the object in the form to use the new class:

```
    Dim OBJ As ContactsWithSort
```

And modify the code of initialization of OBJ in the load event of the form:

```
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        OBJ = New ContactsWithSort()
        OBJ.FillDGV(DGV)
    End Sub
```

Finally, add a menu item to sort the contacts, and write the following in the event handler:

```
        OBJ.sort()
        OBJ.FillDGV(DGV)
```

Run the code check it out. Below is the full code of the **ContactList** class:

```
Public Class ContactList
```

```
    Protected ContactArr() As ContactInfo       ' the array of object,
all elements points to nothing
    Protected C As Integer                      ' the number of objects
in the array

    Public Sub AddNewContact()
        C = C + 1                               ' the number of objects
increases by one
        ContactArr(C - 1) = New ContactInfo     ' create the object
        ContactArr(C - 1).ReadContactInfo()     ' read the information
    End Sub

    Public Sub RemoveContact(ByVal Name As String)
        ' search for the contact
        For I = 0 To C - 1
            If ContactArr(I).Name = Name Then


                ' next remove the contact from the array by shifting the
other objects
                Dim J As Integer
                For J = I + 1 To 999
                    ContactArr(J - 1) = ContactArr(J)
                Next

                ' the number of elements reduces by one
                C = C - 1

                ' exit the block
                Exit Sub
            End If
        Next

    End Sub


    Public Sub FillDGV(ByVal DGV As DataGridView)
        ' clear the data grid view
        DGV.Rows.Clear()

        Dim I As Integer

        ' loop over all the contacts
```

```
        For I = 0 To C - 1
            ' add contact information
            DGV.Rows.Add(ContactArr(I).Name, ContactArr(I).Address,
ContactArr(I).Tel)
        Next

    End Sub

    Public Sub New()
        ' first constructor, set the number of elements to zero, and set
array size to 1000
        C = 0
        ReDim ContactArr(0 To 999)
    End Sub

    Public Sub New(ByVal NoOfReads As Integer)
        ' second constructor, set number of elements to zero, and set
array size to 1000
        C = 0
        ReDim ContactArr(0 To 999)

        ' add the contacts
        Dim I As Integer
        For I = 0 To NoOfReads - 1
            Me.AddNewContact()
        Next
    End Sub

    Protected Overrides Sub Finalize()
        ' this is how to terminate a class
        Dim I As Integer
        For I = 0 To C - 1
            ContactArr(I) = Nothing
        Next


        MyBase.Finalize()
    End Sub
End Class
```

Next is the code for the **ContactsWithSort** class

```
Public Class ContactsWithSort

    Inherits ContactList
      ' this tells the compiler that this class has the same behaviour
      of ContactList

    Public Sub Sort()           ' this class has another method called
sort.
        Dim I As Integer
        Dim F As Boolean
        Dim Contact As ContactInfo

        Do
            F = False

            For I = 0 To C - 2
                If ContactArr(I).Name > ContactArr(I + 1).Name Then
                    F = True
                    Contact = ContactArr(I)
                    ContactArr(I) = ContactArr(I + 1)
                    ContactArr(I + 1) = Contact
                End If
            Next

        Loop While F

    End Sub

End Class
```

And finally the code of the form:

```
Public Class Form1

    Dim OBJ As ContactsWithSort


    Private Sub AddToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
AddToolStripMenuItem.Click
        OBJ.AddNewContact()
        OBJ.FillDGV(DGV)
    End Sub
```

```
    Private Sub RemoveToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
RemoveToolStripMenuItem.Click
        ' check if no rows are selected, if so no need to execute
further code, just exit the subroutine
        If DGV.SelectedRows.Count = 0 Then
            Exit Sub
        End If

        Dim N As String

        ' get the selected name, it is the first column (cell zero)
        N = DGV.SelectedRows(0).Cells(0).Value

        OBJ.RemoveContact(N)
        OBJ.FillDGV(DGV)
    End Sub

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        OBJ = New ContactsWithSort()
        OBJ.FillDGV(DGV)
    End Sub


    Private Sub SortToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
SortToolStripMenuItem.Click
        OBJ.sort()
        OBJ.FillDGV(DGV)
    End Sub
End Class
```

The rest of the files don't need modification, they are the same. So as you can see inheritance allows us to extend the functionality of an existing class, and add some features to them.

Chapter 20: Try & Catch

Try & Catch

When you develop a vb.net application it is very common to get what is known as Runtime error. These are errors that might happen due to some wrong input for example, or some computational operation during program execution. To demonstrate the idea, create a simple vb.net application that read two values from the display and divide them. The code should be similar to this:

```
Dim A As Integer
Dim B As Integer
Dim C As Integer
A = TextBox1.Text
B = TextBox2.Text
C = A / B
MsgBox("The result is:" & C.ToString)
```

This code is correct, and should work fine. However if the value of B is zero (by entering a value of zero in textbox2), then you will get a runtime error. This is simply because you can not divide any number by zero (C=A/B).

This is just a simple example of the errors that you might get. VB allows you to catch such errors so that your program don't crash, and you can give a friendly message to the end user or treat the error. The way to do it is by using the try statement. It should be similar to the following:

```
Try
     The code that could cause error goes here
Catch ex As Exception
     The treatement of the error goes here
End Try
```

To use this one, you can rewrite the code as follows:

```
Dim A As Integer
Dim B As Integer
Dim C As Integer
A = TextBox1.Text
B = TextBox2.Text
```

```
Try

    C = A / B
    MsgBox("The result is:" & C.ToString)

Catch ex As Exception

    MsgBox("Error")

End Try
```

What happens here is that the statements between Try and Catch are monitored for any errors. If an error happens, then the execution will be interrupted, and a new execution starts in the Catch part. So in the example above if the division is by zero, then a friendly message is displayed telling the end user that there is some kind of problem is there. You program will not crash in this case.

Another thing is that there is an object called **ex**. This one holds details about the error. You can get some details about the error itself. For example:

```
Try

    C = A / B
    MsgBox("The result is:" & C.ToString)

Catch ex As Exception

    MsgBox(ex.Message)

End Try
```

Here the program will give the end user the detail of the error (the message property describes the error here). You can display other error details, or store them for debugging purposes. Some of these are ex.StackTrace which gives you the calls that caused the errors, and where the error exactly happened. So it is very useful.

The try statement can catch different set of errors, so using ex.Message is useful because it can tell you what kind of error you are getting and hence you can identify where the error is.

Last thing is that you can use a finally part with the Try statement

```
Try
     The code that could cause error goes here.
Catch ex As Exception
     The treatement of the error goes here.
Finally
     A number of statements that always get executed.
End Try
```

This part is always executed regardless of the state of execution errors. This part can be eliminated, and you can place your code after the try statement resulting in exactly the same effect.

So this concludes the last chapter in this simple programming book. Hope you enjoyed learning the language and practicing some of the included examples.

Thank you.

mkaatr.