Dialog Boxes

- There are many built-in dialog boxes to be used in Windows forms for various tasks like **opening and saving files, printing a page, providing choices for colors, fonts, page setup, etc.,** to the user of an application.
- The main purpose of built-in dialog boxes is to **reduce** the developer's **time** and **workload**.
- The following lists are the commonly used dialog box controls.
  - **ColorDialog**
  - **FontDialog**
  - **OpenFileDialog**
  - **SaveFileDialog**
  - **PrintDialog**
- All of these dialog box control classes inherit from the **CommonDialog** class and override the **RunDialog()** function of the base class to create the specific dialog box.
- The **RunDialog()** function is automatically invoked when a user of a dialog box calls its ShowDialog() function.
- The **ShowDialog** method is used to display all the dialog box controls at run-time. It returns a value of the type of **DialogResult** enumeration. The values of DialogResult enumeration are described in below table:

Value	Description
Abort	Returns DialogResult.Abort value, when user clicks an Abort button.
Cancel	Returns DialogResult.Cancel, when user clicks a Cancel button.
No	Returns DialogResult.No, when user clicks a No button.
None	Returns nothing and the dialog box continues running.
OK	Returns DialogResult.OK, when user clicks an OK button
Retry	Returns DialogResult.Retry , when user clicks an Retry button
Yes	Returns DialogResult.Yes, when user clicks an Yes button

1. ColorDialog

- It represents a common dialog box that displays available colors along with controls that enable the user to define custom colors.
- It lets the user select a color.
- The main property of the ColorDialog control is Color, which returns a **Color** object.

Properties:

Property Name	Description
AnyColor	It gets or sets a value indicating whether the dialog box displays all available colors in the set of basic colors.
Color	It gets or sets the color selected by the user.
CustomColors	It gets or sets the set of custom colors shown in the dialog box.
FullOpen	It gets or sets a value indicating whether the controls used to create custom

	colors are visible when the dialog box is opened
ShowHelp	It gets or sets a value indicating whether a Help button appears in the color dialog box.

Methods:

Method Name	Description
Reset	Resets all options to their default values, the last selected color to black, and the custom colors to their default values.
RunDialog	When overridden in a derived class, specifies a common dialog box.
ShowDialog	Runs a common dialog box with a default owner.

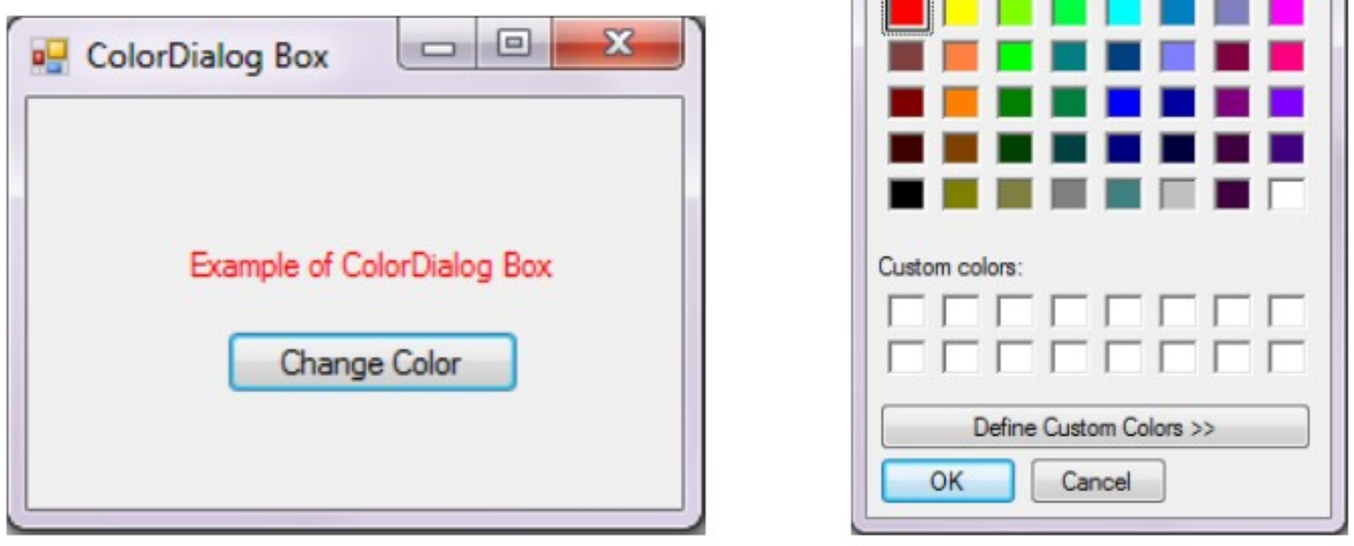
Events:

Method Name	Description
HelpRequest	Occurs when the user clicks the Help button on a common dialog box.

Example:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
    If ColorDialog1.ShowDialog <> Windows.Forms.DialogResult.Cancel Then
            Label1.ForeColor = ColorDialog1.Color
    End If
End Sub
```

Output:



2. FontDialog

- It prompts the user to choose a font from among those installed on the local computer and lets the user select the font, font size, and color.
- It returns the Font and Color objects.

Properties:

Property Name	Description
Color	It gets or sets the selected font color.
Font	It gets or sets the selected font.

MaxSize	It gets or sets the maximum point size a user can select.
MinSize	It gets or sets the minimum point size a user can select.
ShowEffects	It gets or sets a value indicating whether the dialog box contains controls that allow the user to specify strikethrough, underline, and text color options.
ShowHelp	It gets or sets a value indicating whether the dialog box displays a Help button.

Methods:

Method Name	Description
Reset	Resets all options to their default values, the last selected color to black, and the custom colors to their default values.
RunDialog	When overridden in a derived class, specifies a common dialog box.
ShowDialog	Runs a common dialog box with a default owner.

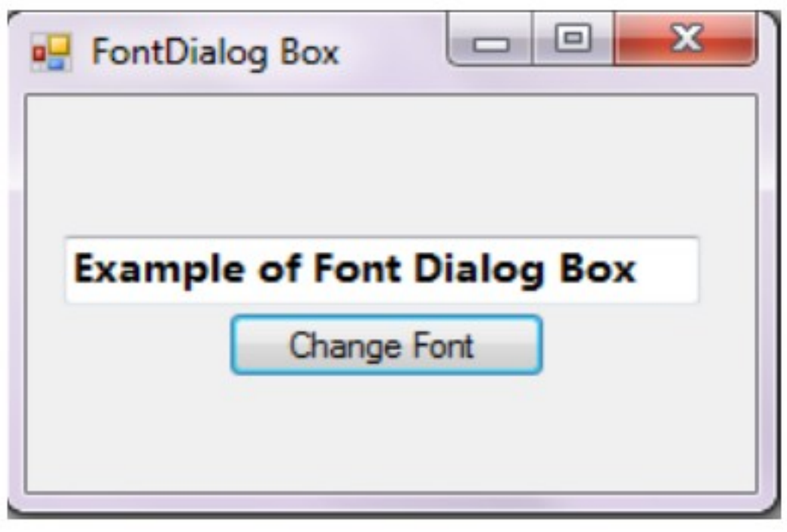
Events:

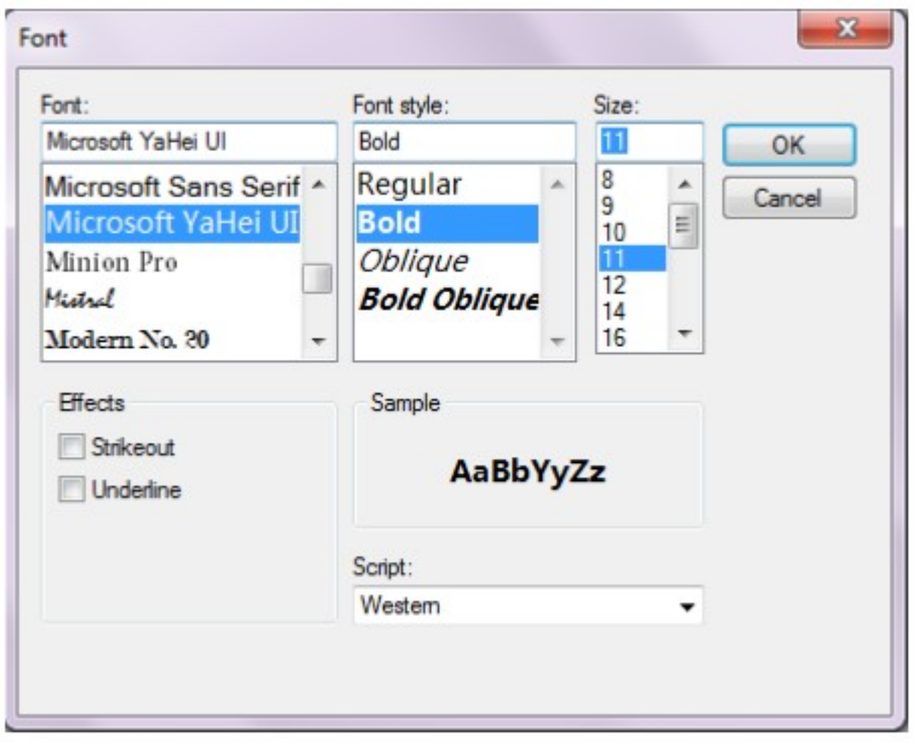
Method Name	Description
Apply	Occurs when the Apply button on the font dialog box is clicked.

Example:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
    If FontDialog1.ShowDialog <> Windows.Forms.DialogResult.Cancel Then
            TextBox1.ForeColor = FontDialog1.Color
            TextBox1.Font = FontDialog1.Font
End If
```

Output:





3. OpenFileDialog

- The **OpenFileDialog** control prompts the user to open a file and allows the user to select a file to open.
- The user can check if the file exists and then open it.
- The **OpenFileDialog** control class inherits from the abstract class **FileDialog**.

Properties:

Property Name	Description
AddExtension	It gets or sets a value indicating whether the dialog box automatically adds extension to a file name if the user omits the extension.
CheckFileExists	It gets or sets a value indicating whether the dialog box displays a warning if the user specifies a file name that does not exist.
CheckPathExists	It gets or sets a value indicating whether the dialog box displays a warning if the user specifies a path that does not exist.
FileName	It gets or sets a string containing the file name selected in the file dialog box.
Multiselect	It gets or sets a value indicating whether the dialog box allows multiple files to be selected.
ShowHelp	It gets or sets a value indicating whether the Help button is displayed in the file dialog box.
Title	It gets or sets the file dialog box title.
ValidateNames	It gets or sets a value indicating whether the dialog box accepts only valid Win32 file names.

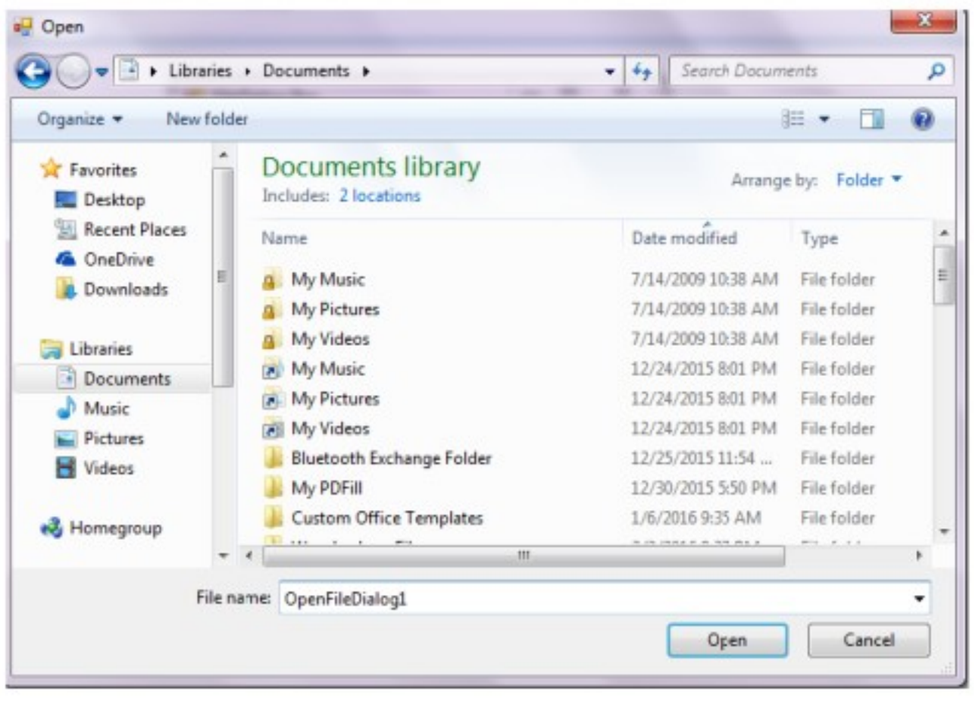
Methods:

Method Name	Description
OpenFile	Opens the file selected by the user, with read-only permission. The file is specified by the FileName property.
Reset	Resets all options to their default value.

Example:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
    If OpenFileDialog1.ShowDialog <> Windows.Forms.DialogResult.Cancel Then
        PictureBox1.Image = Image.FromFile(OpenFileDialog1.FileName)
    End If
End Sub
```

Output:



4. SaveFileDialog

- The **SaveFileDialog** control prompts the user to select a location for saving a file and allows the user to specify the name of the file to save data.
- The **SaveFileDialog** control class inherits from the abstract class **FileDialog**.

Properties:

Properties are same as the Open Dialog.

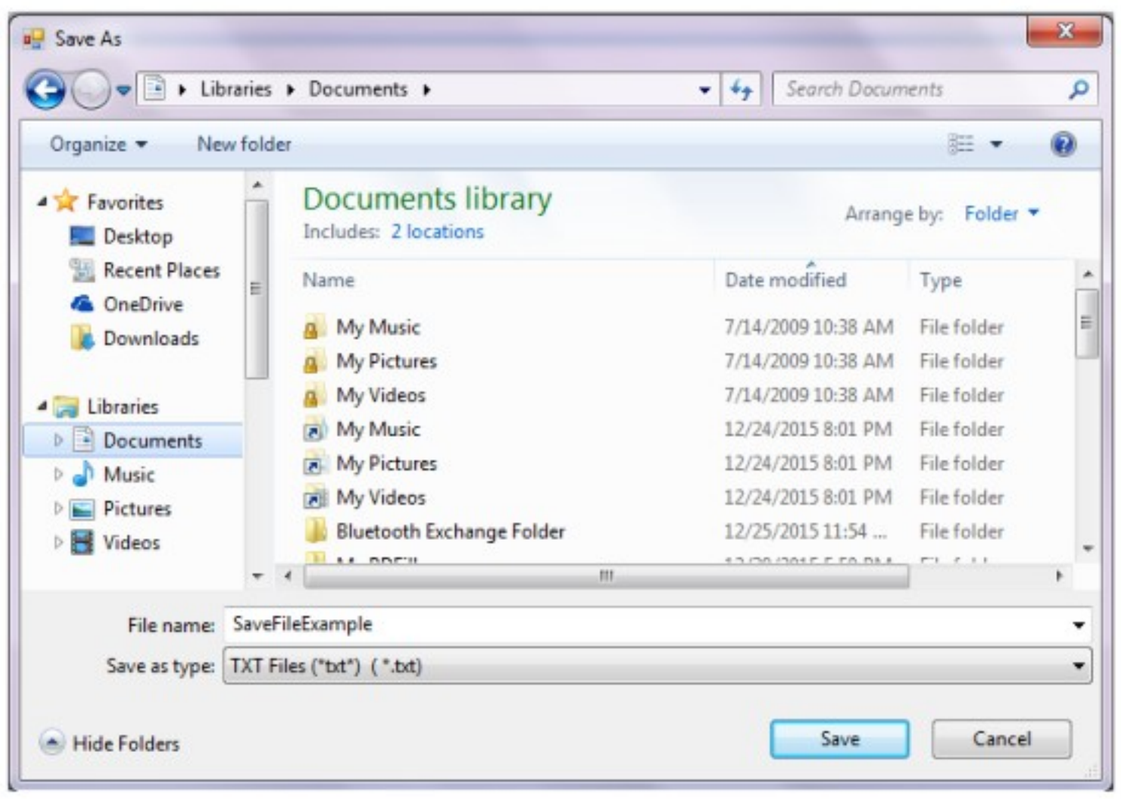
Methods:

Methods are same as the Open Dialog.

Example:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
    SaveFileDialog1.Filter = "TXT Files (*txt*) | *.txt"
    If SaveFileDialog1.ShowDialog = Windows.Forms.DialogResult.OK Then
        My.Computer.FileSystem.WriteAllText(SaveFileDialog1.FileName,
        RichTextBox1.Text, True)
    End If
End Sub
```

Output:



5. PrintDialog

- The **PrintDialog** control lets the user to print documents by selecting a printer and choosing which sections of the document to print from a **Windows Forms application**.
- There are various other controls related to printing of documents. The list of other controls are given below:
  - **PrintDocument control**: It provides support for actual events and operations of printing in Visual Basic and sets the properties for printing.
  - **PrinterSettings control**: It is used to configure how a document is printed by specifying the printer.

- **PageSetUpDialog control**: It allows the user to specify page-related print settings including page orientation, paper size and margin size.
- **PrintPreviewControl control**: It represents the raw preview part of print previewing from a Windows Forms application, without any dialog boxes or buttons.
- **PrintPreviewDialog control**: It represents a dialog box form that contains a PrintPreviewControl for printing from a Windows Forms application.

Properties:

Property Name	Description
AllowCurrentPage	It gets or sets a value indicating whether the Current Page option button is displayed.
AllowPrintToFile	It gets or sets a value indicating whether the Print to file check box is enabled.
AllowSelection	It gets or sets a value indicating whether the Selection option button is enabled.
PrinterSettings	It gets or sets the printer settings the dialog box modifies.
Document	It gets or sets a value indicating the PrintDocument used to obtain PrinterSettings.
PrintToFile	It gets or sets a value indicating whether the Print to file check box is selected.
ShowHelp	It gets or sets a value indicating whether the Help button is displayed.

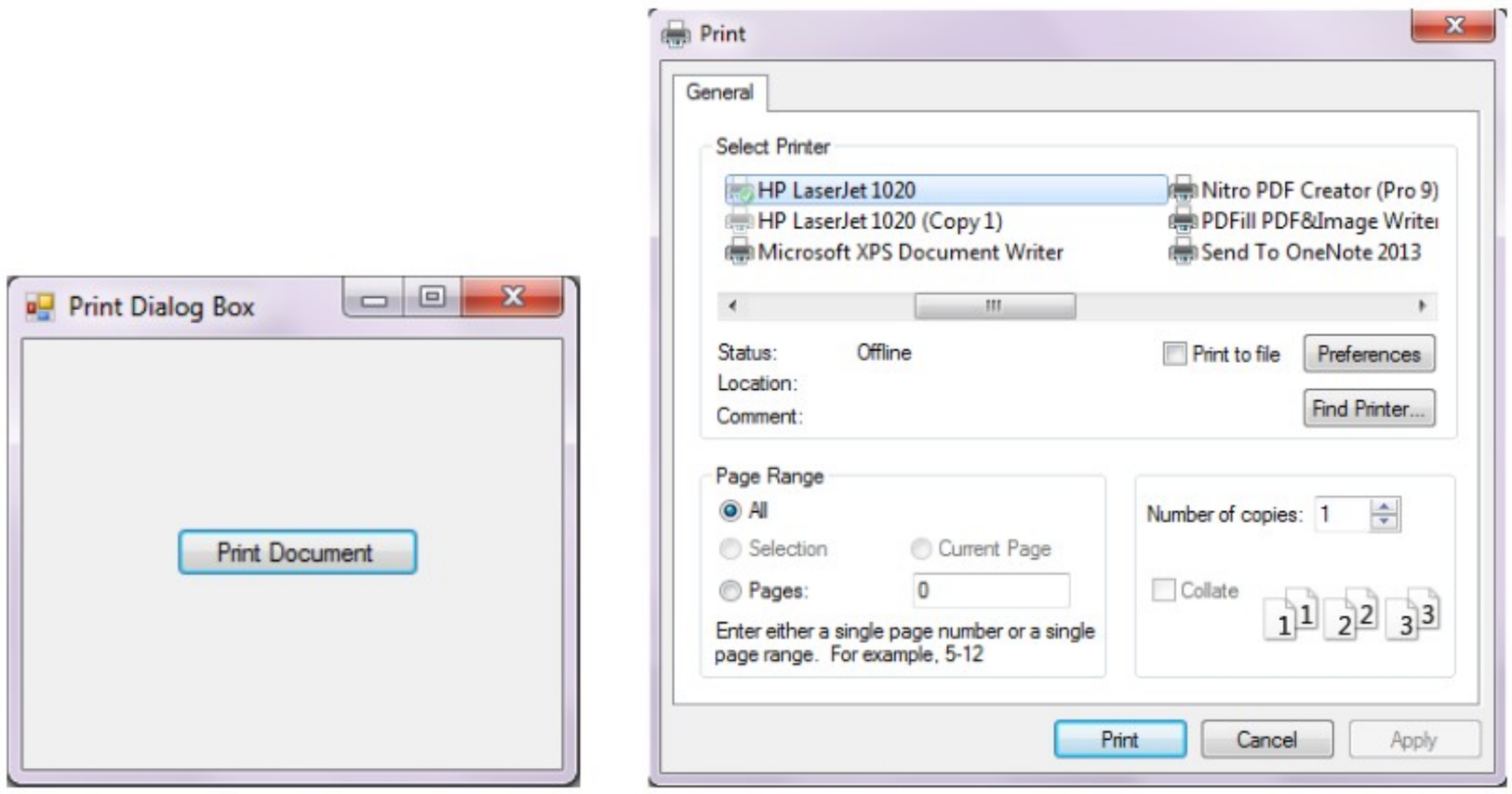
Methods:

Method Name	Description
Reset	Resets all options to their default values.
RunDialog	When overridden in a derived class, specifies a common dialog box.
ShowDialog	Runs a common dialog box with a default owner

Example:

```
Private Sub Button1_Click_1(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    PrintDialog1.Document = PrintDocument1
    PrintDialog1.PrinterSettings = PrintDocument1.PrinterSettings
    PrintDialog1.AllowSomePages = True
    If PrintDialog1.ShowDialog = DialogResult.OK Then
        PrintDocument1.PrinterSettings = PrintDialog1.PrinterSettings
        PrintDocument1.Print()
    End If
End Sub
```

Output:



Sub Procedures and functions

- A **procedure** is a group of statements that together perform a task when called. After the procedure is executed, the control returns to the statement calling the procedure.
- VB.Net has two types of procedures:
  - **Functions**
  - **Sub procedures or Subs**

Functions (Defining a Function)

- The Function statement is used to **declare the name, parameter and the body of a function**.
- Functions return a value, whereas Subs do not return a value.
- The **syntax** for the **Function** statement is:

[Modifiers] Function FunctionName [(parameterlist)] as ReturnType
[statements]
End Function

  - **Modifiers**: Specify the access level of the function; possible values are: Public, Private, Protected, Friend, Protected, Friend and information regarding overloading, overriding, sharing, and shadowing.
  - **FunctionName**: Indicates the name of the function.
  - **ParameterList**: Specifies the list of the parameters.
  - **ReturnType**: Specifies the data type of the variable the function returns.

Function Returning a Value:

- A function can return a value to the calling code in two ways:
  - **By using the return statement**
  - **By assigning the value to the function name**

- In the below example, we write a one function which find out the maximum number from a given two numbers and the return type of that function is Integer.

Example:

```
Module Module1
    Function max_number(ByVal number1 As Integer, ByVal number2 As Integer) As Integer
        Dim result As Integer
        If (number1 > number2) Then
            result = number1
        Else
            result = number2
        End If
        Return result 'By using the return statement
        'max_number = result 'By assigning the value to the function name
    End Function
    Sub Main()
        Dim ans As Integer = max_number(1, 7)
        Console.Write("Maximum Number is : {0}", ans)
        Console.ReadKey()
    End Sub
End Module
```

Output:

Maximum Number is : 7

Sub Procedures or Subs

- **Sub procedures are procedures** that do not return any value. We have been using the Sub procedure Main in all our examples.
- When console applications start, the control goes to the Main Sub procedure, and it in turn, runs any other statements constituting the body of the program.

Defining a Sub Procedures:

- The Sub statement is used to declare the name, parameter and the body of a sub procedure. The syntax for the Sub statement is:

```
[Modifiers] Sub subname [parameterlist]
        [statements]
End Sub
```

  - **Modifiers**: specify the access level of the procedure; possible values are: Public, Private, Protected, Friend, Protected, Friend and information regarding overloading, overriding, sharing, and shadowing.
  - **SubName**: indicates the name of the Sub.
  - **ParameterList**: specifies the list of the parameters.

Example:

```
Module Module1
    Sub Simple()
        Console.WriteLine("Simple Non Parameterized Procedure")
        Console.WriteLine()
    End Sub
    Sub Addition(ByVal number1 As Integer, ByVal number2 As Integer)
        Dim ans As Integer = number1 + number2
        Console.WriteLine("Addition of 5 and 3 is {0}", ans)
    End Sub
    Sub Main()
        Simple()
        Addition(5, 3)
        Console.ReadKey()
    End Sub
End Module
```

Output:

```
Simple Non Parameterized Procedure
Addition of 5 and 3 is 8
```

- As shown in the above example, we created a one sub procedure named "**Addition**" which takes two integer parameters and we also created another Sub Procedure called "**Simple**" which has no parameter.

Pass by value

- In this mechanism, when a method is called, a new storage location is created for each value parameter.
- The values of the actual parameters are copied into them. So, the changes made to the parameter inside the method have no effect on the argument.
- In VB.Net, you declare the reference parameters using the **ByVal** keyword. The following example demonstrates the concept of pass by value in which we swap the value of two variables:

Example:

```
Module Module1
    Sub swap(ByVal x As Integer, ByVal y As Integer)
        Dim temp As Integer
        temp = x
        x = y
        y = temp
    End Sub
    Sub Main()
```

```
        Dim a As Integer = 11
        Dim b As Integer = 22
        Console.WriteLine("Before swap, value of a : {0}", a)
        Console.WriteLine("Before swap, value of b : {0}", b)
        swap(a, b)
        Console.WriteLine()
        Console.WriteLine("After swap, value of a : {0}", a)
        Console.WriteLine("After swap, value of b : {0}", b)
        Console.ReadLine()
    End Sub
End Module
```

Output:

```
Before swap, value of a : 11
Before swap, value of b : 22

After swap, value of a : 11
After swap, value of b : 22
```

- The above example shows that there is no change in the values even if they had been changed inside the function.

Passing Parameters by Reference

- A reference parameter is a **reference to a memory location of a variable**.
- When you pass parameters by reference, unlike value parameters, a new storage location is not created for these parameters.
- The reference parameters represent the same memory location as the actual parameters that are supplied to the method.
- In VB.Net, you declare the reference parameters using the **ByRef** keyword. The following example demonstrates the concept of pass by reference in which we swap the value of two variables.

Example:

```
Module Module1
    Sub swap(ByRef x As Integer, ByRef y As Integer)
        Dim temp As Integer
        temp = x
        x = y
        y = temp
    End Sub
    Sub Main()
        Dim a As Integer = 11
        Dim b As Integer = 22
```

```
        Console.WriteLine("Before swap, value of a : {0}", a)
        Console.WriteLine("Before swap, value of b : {0}", b)
        swap(a, b)
        Console.WriteLine()
        Console.WriteLine("After swap, value of a : {0}", a)
        Console.WriteLine("After swap, value of b : {0}", b)
        Console.ReadLine()
    End Sub
End Module
```

Output:

```
Before swap, value of a : 11
Before swap, value of b : 22

After swap, value of a : 22
After swap, value of b : 11
```

Exception/Error Handling:

Structured Error Handling

- For Error Handling VB.Net provides Exception Handling, an exception is a problem that arises during the execution of a program.
- An exception is a response to an exceptional circumstance that arises while a program is running, such as an **attempt to divide by zero**.
- Exceptions provide a way to transfer control from one part of a program to another. VB.Net exception handling is built upon four keywords: **Try, Catch, Finally and Throw**.
  - **Try**: A Try block identifies a block of code for which particular exceptions will be activated. It's followed by one or more Catch blocks.
  - **Catch**: A program catches an exception with an exception handler at the place in a program where you want to handle the problem.
  - **Finally**: The Finally block is used to execute a given set of statements, whether an exception is thrown or not thrown.
  - **Throw**: A program throws an exception when a problem shows up. This is done using a Throw keyword.

Syntax of Try/Catch:

```
Try
    [Statements]
Catch ex as [Exception Type]
    [Statements]
Catch ex1 as [Exception Type]
    [Statements]
```

```
Finally
    [Finally statements]
End Try
```

- You can list down multiple catch statements to catch different type of exceptions in case your try block raises more than one exception in different situations.
- The following table provides some of the predefined exception classes derived from the **Sytem.SystemException** class*:*

Exception Class	Description
System.IO.IOException	It handles I/O errors.
System.IndexOutOfRangeException	It handles errors generated when a method refers to an array index out of range.
System.DivideByZeroException	It handles errors generated from dividing a dividend with zero.
System.OutOfMemoryException	It handles errors generated from insufficient free memory.
System.NullReferenceException	It handles errors generated from deferencing a null object.

Example:

```
Module exceptionProg
    Sub division(ByVal num1 As Integer, ByVal num2 As Integer)
        Dim result As Integer
        Try
            result = num1 / num2
        Catch e As DivideByZeroException
            Console.WriteLine("Exception caught: {0}", e)
        Finally
            Console.WriteLine("Result: {0}", result)
        End Try
    End Sub
    Sub Main()
        division(25, 0)
        Console.ReadKey()
    End Sub
End Module
```

Output:

```
Exception caught: System.DivideByZeroException: Attempted to divide by zero.
at ...
Result: 0
```

Unstructured Error Handling (On error go to line, goto 0, goto -1, resume next)

- **On Error GoTo statements** is used to handling Unstructured Exception.
- Error GoTo redirect the flow of the program in a given location.

Syntax On Error Go:

On Error { GoTo [line | 0 | -1 | Resume Next }

- **GoTo line**: Enables the error-handling routine that starts at the line specified in the required line argument. The line argument is any line label or line number. If a run-time error occurs, control branches to the specified line, making the error handler active. The specified line must be in the same procedure as the On Error statement, or a compile-time error will occur.
- **GoTo 0**: Disables the enabled error handler in the current procedure and resets it to Nothing.
- **GoTo -1**: Disables the enabled exception in the current procedure and resets it to Nothing.
- **Resume Next**: Specifies that when a run-time error occurs, control goes to the statement immediately following the statement where the error occurred, and execution continues from that point. Use this form rather than On Error GoTo when accessing objects.

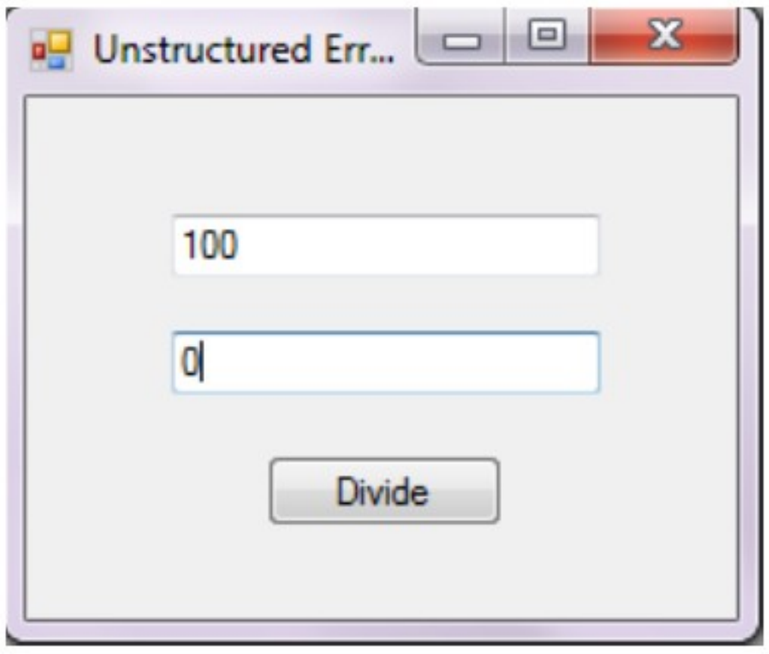
Example:

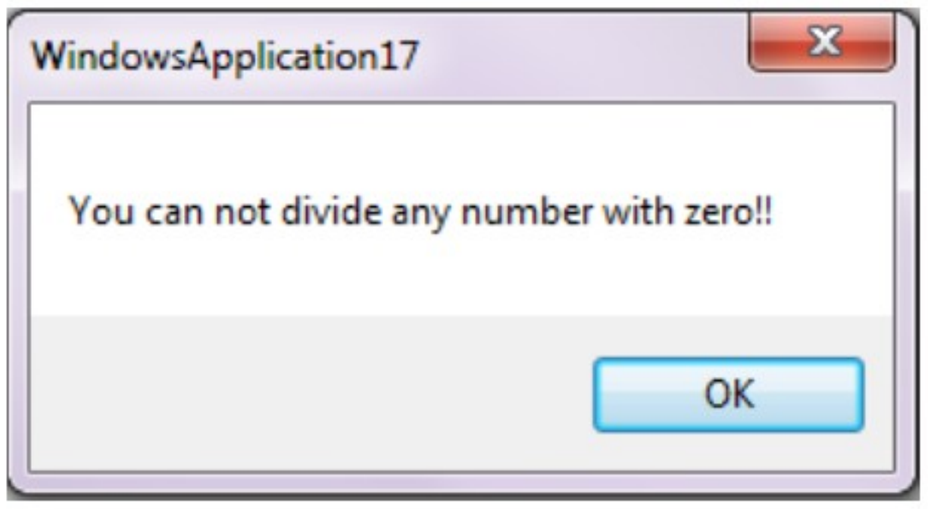
```
Public Class Form1
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
    Handles Button1.Click
        Dim Num1, Num2, Ans As Integer
        Num1= TextBox1.Text
        Num2= TextBox2.Text

        On Error GoTo Handle
            Ans = Num1 /  Num2
            MessageBox.Show("Your answer is:" & Ans)

        On Error GoTo 0
            TextBox1.Text = ""
            TextBox2.Text = ""
    Handle:
        If (TypeOf Err.GetException() Is ArithmeticException) Then
            MsgBox("You can not divide any number with zero!!")
            Resume Next
        End If
    End Sub
End Class
```

Output:





Multiple document interface (MDI)

- The **multiple-document interface (MDI)** allows you to create an application that maintains multiple forms within a single container form.
- Applications such as **Microsoft Excel and Microsoft Word** for Windows have multiple-document interfaces.
- An MDI application allows the user to display multiple documents at the same time, with each document displayed in its own window.
- **Documents or child windows** are contained in a parent window, which provides a workspace for all the child windows in the application.
- For example, **Microsoft Excel** allows you to create and display multiple-document windows of different types. Each individual window is confined to the area of the Excel parent window. When you minimize Excel, all of the document windows are minimized as well, only the parent window's icon appears in the task bar.
- A **child form** is an ordinary form that has its **MDIChild** property set to **True**. Your application can include many MDI child forms of similar or different types.
- Following below are the steps to create **MDI parent** form in vb.net application:
  - Open visual studio then Go to and **create a new window application**→Give proper title "**MDI_Example**"→Click **OK** button.
  - After that click on **the Solution explorer** and right click on the **application name**→Select **Add** option from the opened context menu→Select **New Item (See Fig 4.1).** It will open a one dialog box (See Fig 4.2)
  - From opened dialog box, Select **Windows Form category** from left side in opened dialog box→select **MDI Parent Form** (give proper name of the MDI Form)→Click on the **ADD** button. (See Fig 4.2)
  - Select the **MDI** form and set the following **properties** of the **MDI form**.

    IsMdiContainer = True

    WindowState = Maximize
  - Now set your **MDI form as a startup form** in your application, Do the following steps:

- **Go to Solution Explorer** and select your application name.
- Right click on it and select **properties** from the opened context menu.
- It will open one dialog box which shown in Fig 4.3. Select application tab from the opened dialog box and **set your MDI form as Startup form** which is shown in Fig 4.3.

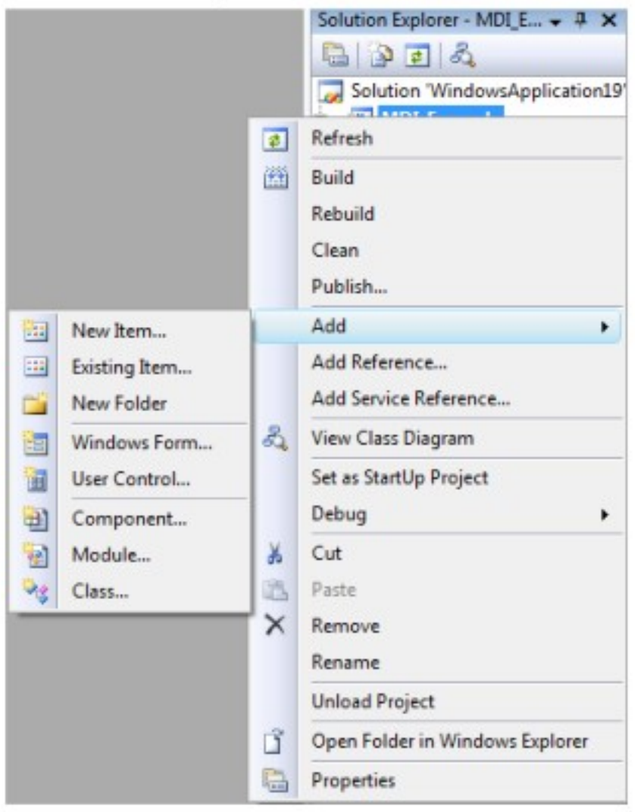


Fig 4.1 add new Item in application

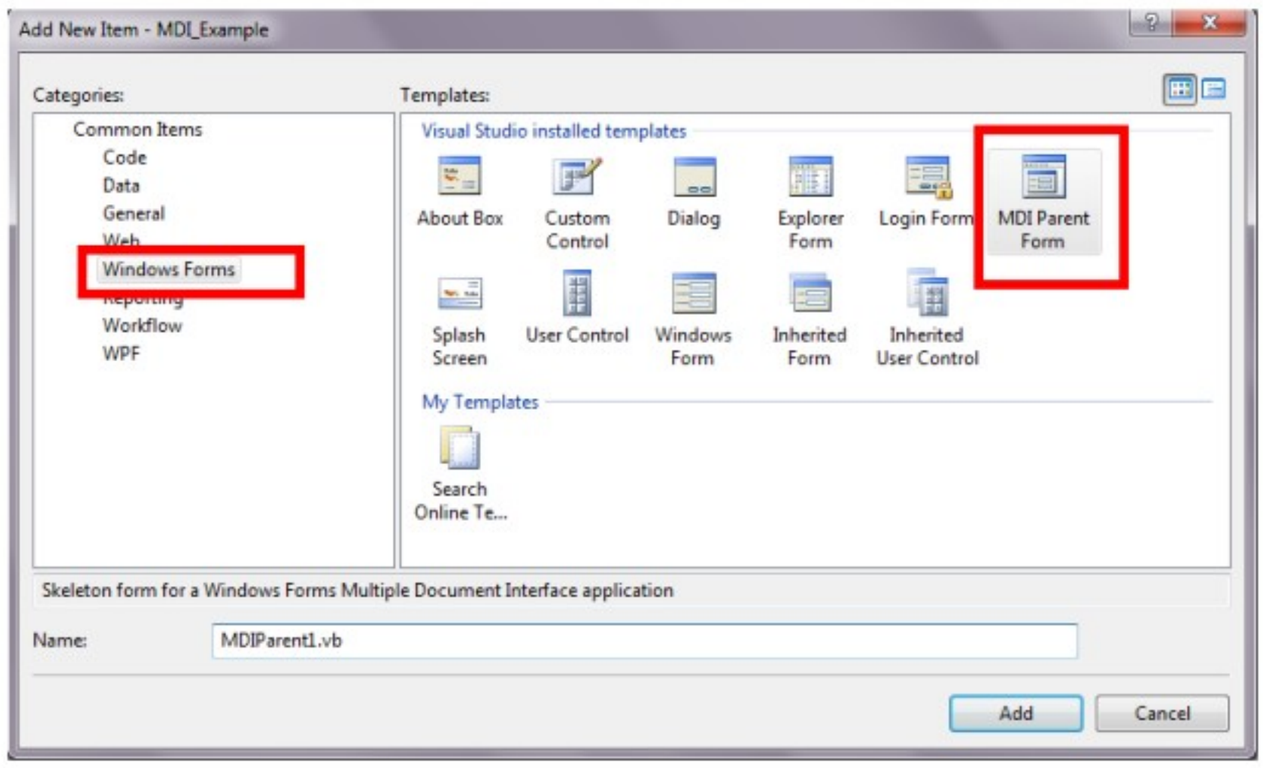


Fig 4.2 Add New Item Dialog Box

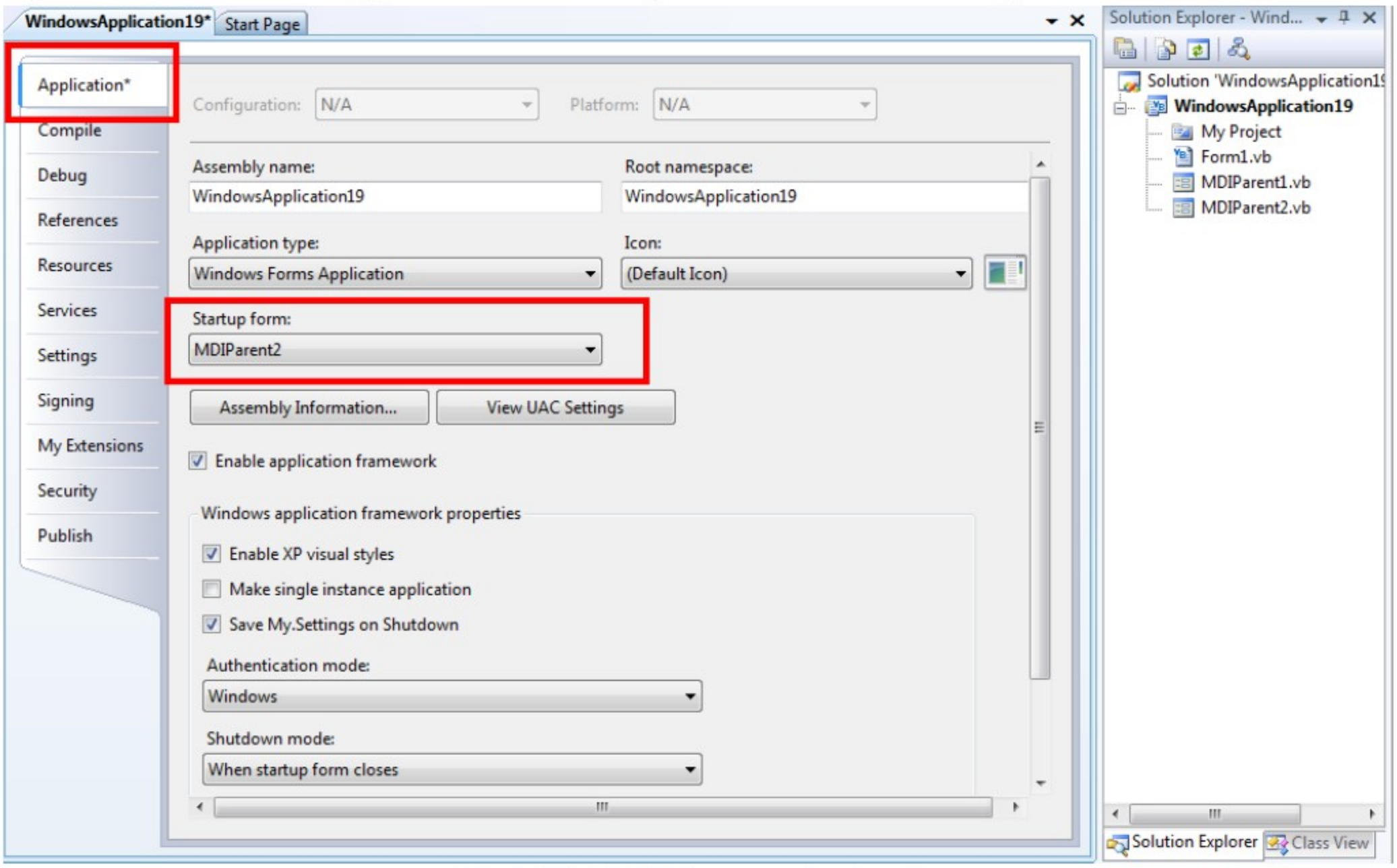


Fig 4.3 Set MDI form as startup Form

- Following below are the steps to create **MDI Child** form in vb.net application:
  - Create a MDI parent form with a MenuStrip which containing the values New and Windows as menu items and also create an Open as sub menu of New.(See Fig 4.4)

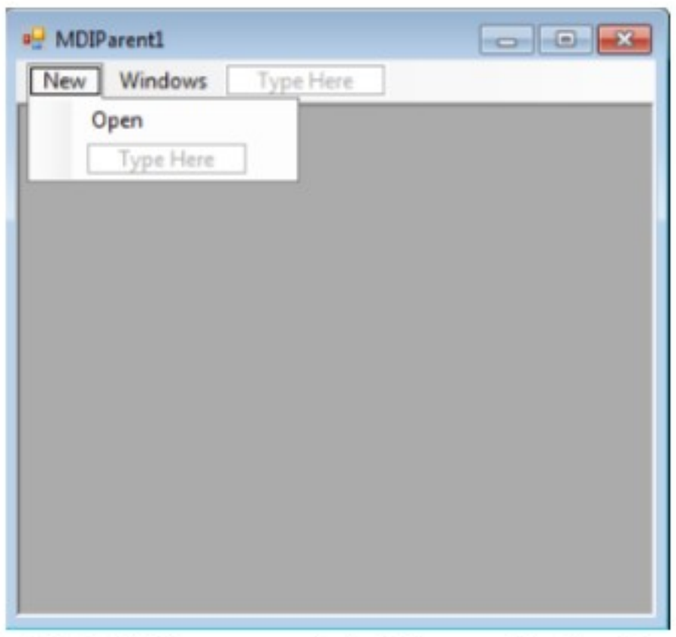


Fig 4.4 MDI Form with MenuStrip Control

- After that click on **the Solution explorer** and right click on the **application name**→Select **Add** option from the opened context menu→Select **New Item.** It will open a one dialog box (See Fig 4.2)
- From opened dialog box, Select **Windows Form category** from left side in opened dialog box→select **Windows Form**→Click on the **ADD** button (See Fig 4.2).
- After that drag and drop a RichTextBox Control to the newly created **Form** from Toolbox.
- After that double click on the submenu "Open" from MDI form and write the following code to add the Child form under this MDI form.

```
Private Sub OpenToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles OpenToolStripMenuItem.Click
    Dim ChildForm As Form1 = New Form1()
    ChildForm.MdiParent = Me
    ChildForm.WindowState = FormWindowState.Normal
    ChildForm.Show()
End Sub
```

- Now run the application, and then you can create a new MDI child form by clicking on Open Submenu Item. When you click on the Open submenu, it will open childform within the MDI parent form which shown in Fig 4.5.

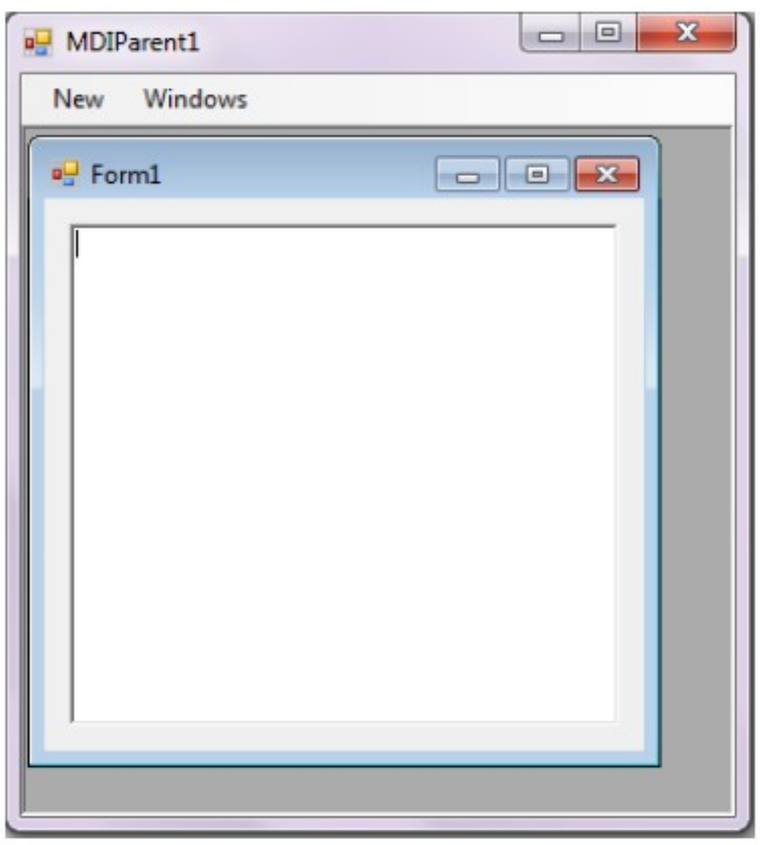


Fig 4.5 ChildForm within MDI parent form