UNIT 4

ADVANCE FEATURES OF VB .NET

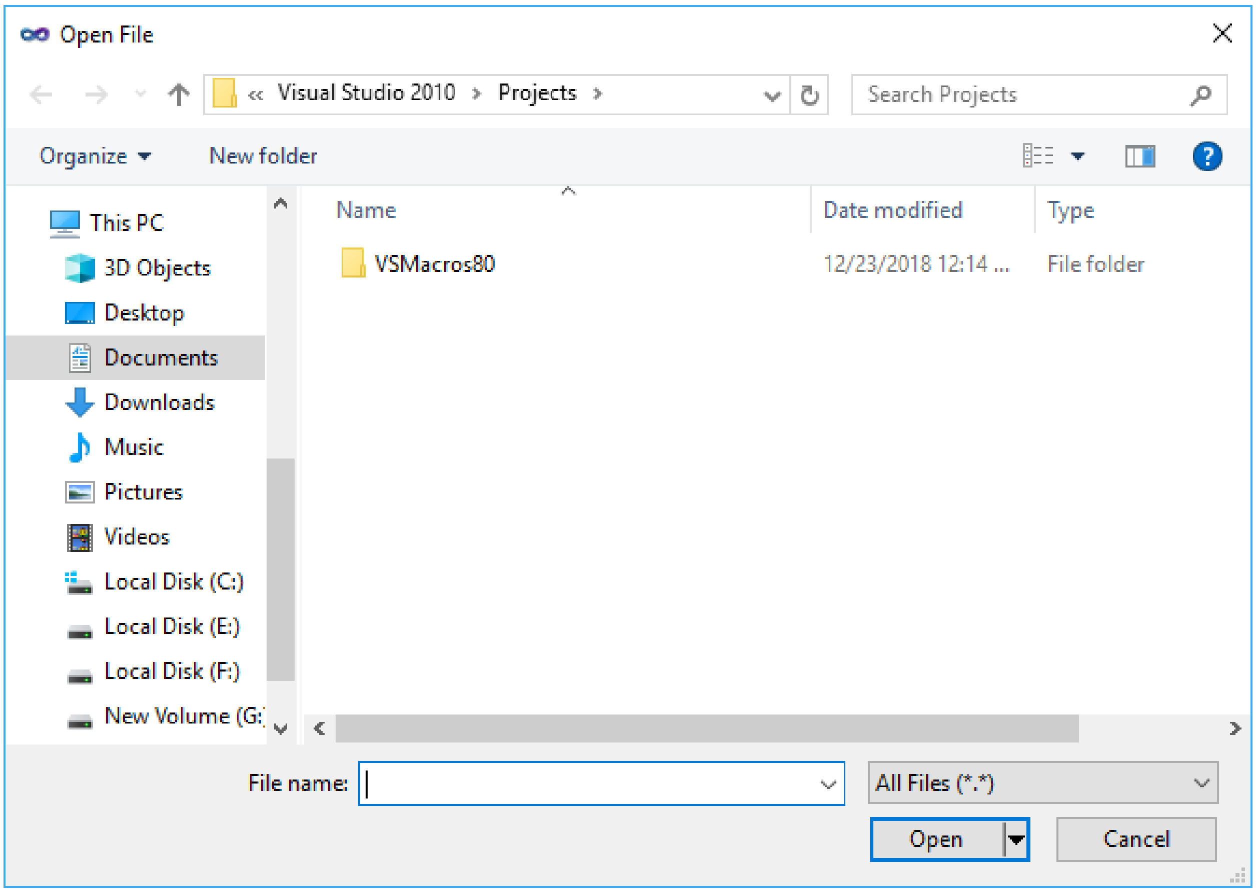
Outline of the Chapter

1. OVERVIEW OF DIALOGBOX IN VB.NET:

- The common dialog used for perform various tasks like opening and saving files, printing a page, providing choices for colors and fonts, page setup, etc., to the user of an application.
- These dialog boxes are used in many applications for using various functionalities. 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
- These **dialog boxes** are available in the Design Group in ToolBox except PrintDialog.
- PrintDialog is available in Printing Group of ToolBox.
- To display a common dialog box from within your code, you simply call the control's **showdialog method**, which is common for all controls.
- Note that all common dialog controls can be displayed only modally and they don't expose a Show method. Modally dialog box means, until you do not response that dialog box, parent application cannot get control back.
- As you called the showdialog method, the corresponding dialogbox appears on the screen and till that the **execution of the program is suspended until the box is closed.**
- Using the Open, Save and FolderBrowser dialog boxes, user can travers the entire structure of their drives and locate the desired filename or folder. When user click on Save or Open button, the dialog box closed and execution of the program is resumes.

1.1 OpenFileDialog and SaveFileDialog control:

- 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**.



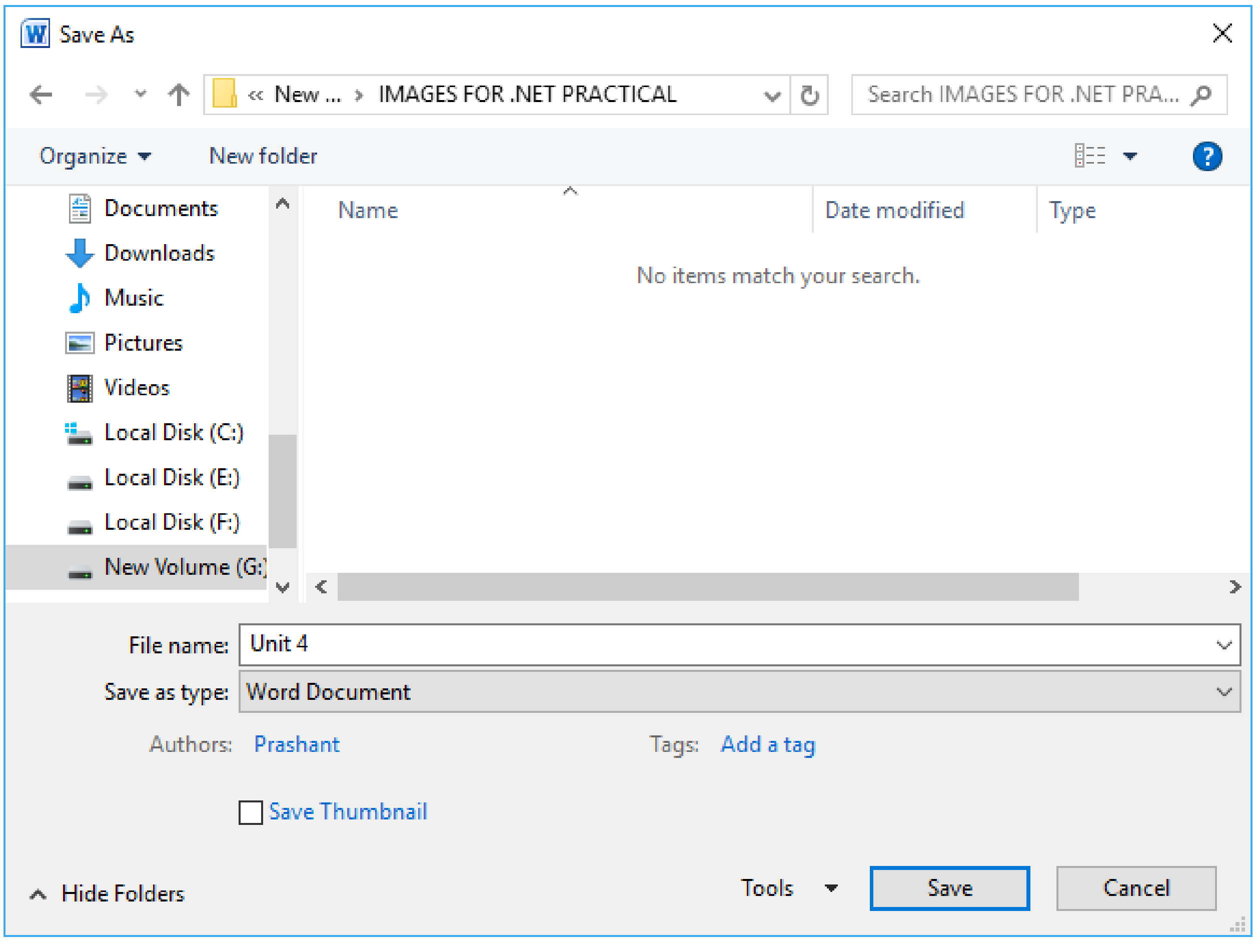
Properties:

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.**
DefaultExt	It gets or sets a **default file extension**.
FileName	It gets or sets a **string containing the file name** selected in the file dialog box.
FileNames	It gets file names of **all selected files** in the file dialog box.
Filter	It gets or sets current file name filter string, which determines the **choices that appears in the “Save as file type” or “Files of Type”** box in the dialog box.
FilterIndex	Gets or sets the **index of the filter currently selected** in the file dialog box.
InitialDirectory	Gets or sets the **initial directory displayed by the file dialog box**.
RestoreDirectory	Gets or sets a value indicating **whether the dialog box restores the current directory before closing**.
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.**

1.2 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.



File name: Unit 4

Save as type: Word Document

Authors: Prashant

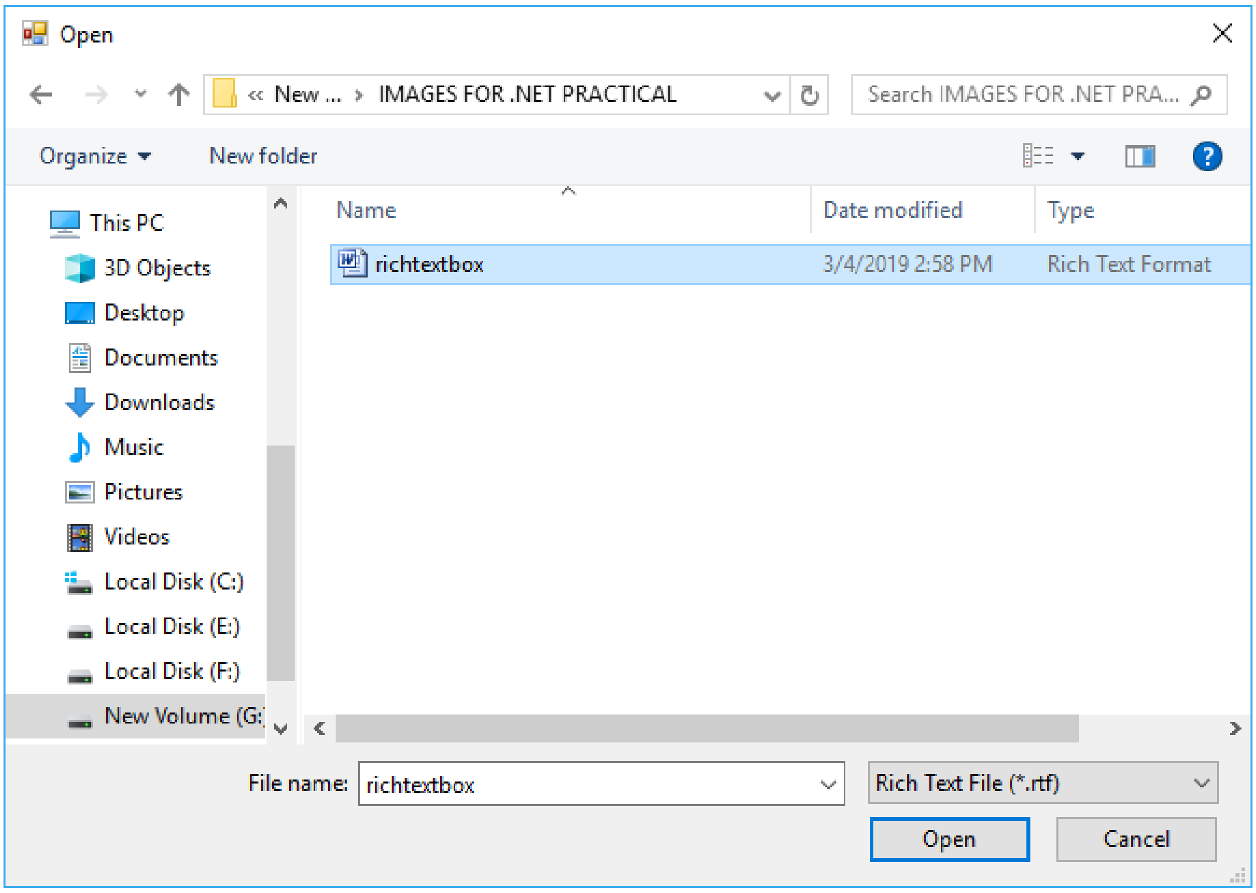
Properties:

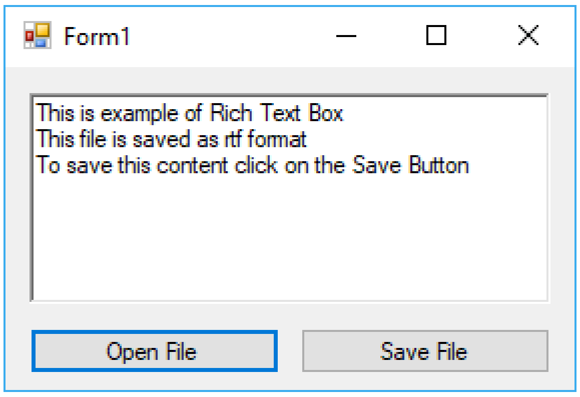
- Properties are same as the Open Dialog.

Example:

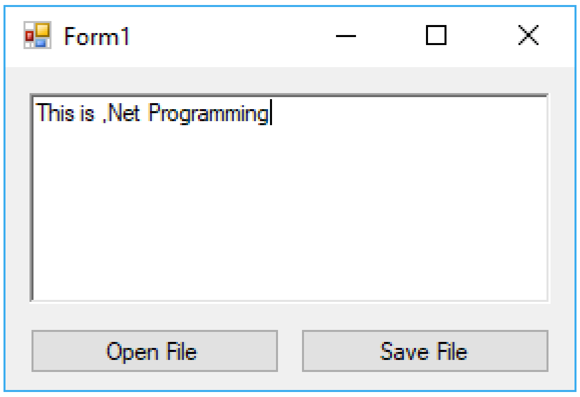
```
Public Class Form1
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles Button1.Click
    OpenFileDialog1.Filter = "Rich Text File (*.rtf)|*.rtf|Word Doc
    (*.doc)|*.doc"
    If OpenFileDialog1.ShowDialog = Windows.Forms.DialogResult.OK Then
        RichTextBox1.LoadFile(OpenFileDialog1.FileName)
    End If
    End Sub
    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles Button2.Click

        SaveFileDialog1.Filter = "TXT Files (*.txt)|*.txt|All Files
        (*.*)|*.*"
        If SaveFileDialog1.ShowDialog = Windows.Forms.DialogResult.OK
        Then
            My.Computer.FileSystem.WriteAllText(SaveFileDialog1.FileN
            ame, RichTextBox1.Text, True)

            MessageBox.Show("File Created Successfully. Please Check
            it")
        End If
    End Sub
End Class
```

Output:

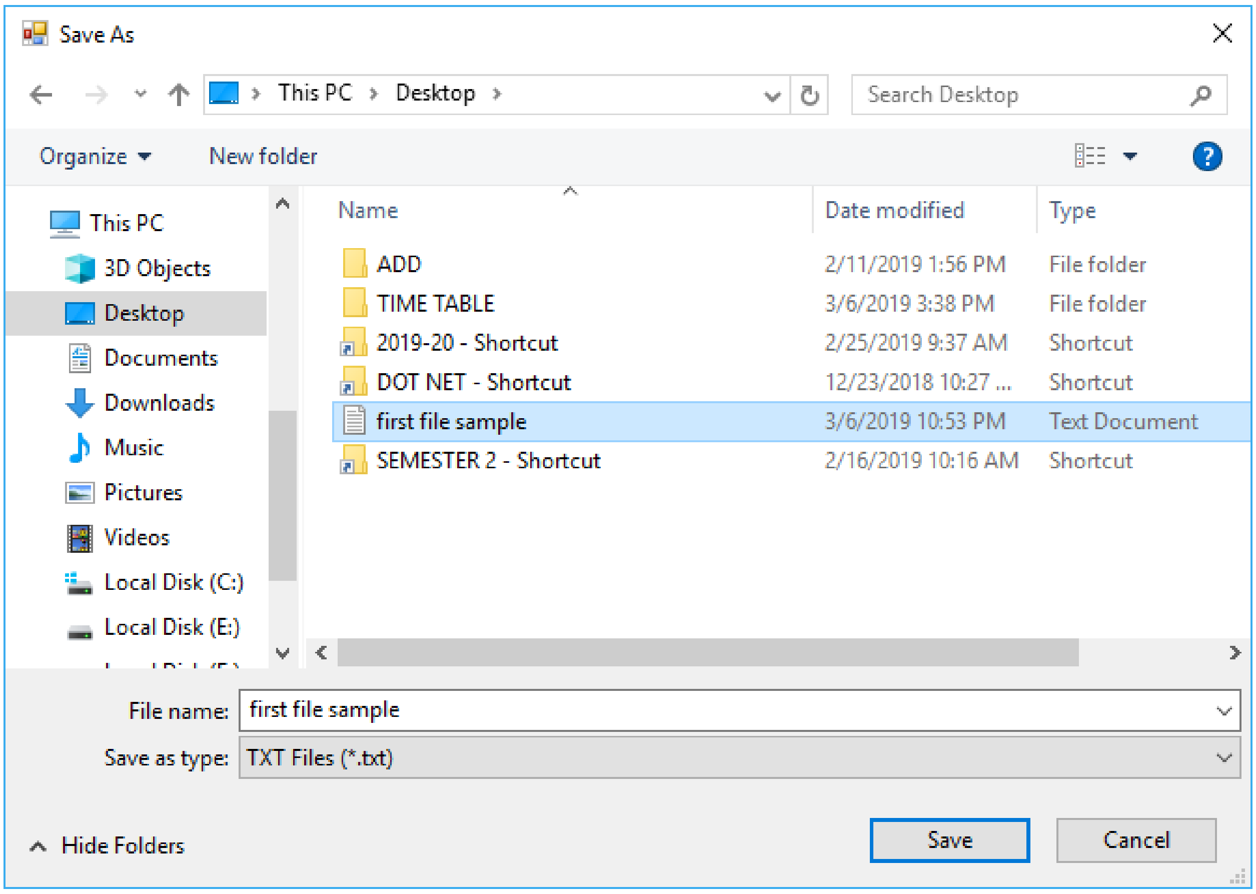




This is example of Rich Text Box
This file is saved as rtf format
To save this content click on the Save Button



This is .Net Programming



Name	Date modified	Type
ADD	2/11/2019 1:56 PM	File folder
TIME TABLE	3/6/2019 3:38 PM	File folder
2019-20 - Shortcut	2/25/2019 9:37 AM	Shortcut
DOT NET - Shortcut	12/23/2018 10:27 ...	Shortcut
first file sample	3/6/2019 10:53 PM	Text Document
SEMESTER 2 - Shortcut	2/16/2019 10:16 AM	Shortcut

File name: first file sample
Save as type: TXT Files (*.txt)

1.3 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.
AllowFullOpen	Gets or sets a value indicating **whether the user can use the dialog box to define custom colors.**

Color	It gets or sets the **color selected by the user.**
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.**
SolidColorOnly	Gets or sets a value indicating whether the dialog box will **restrict users to selecting solid colors only**.

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.

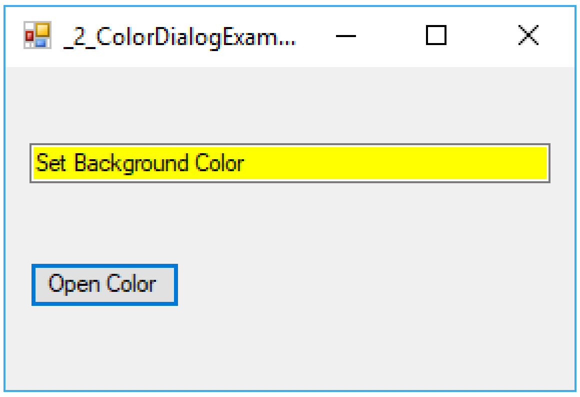
Events:

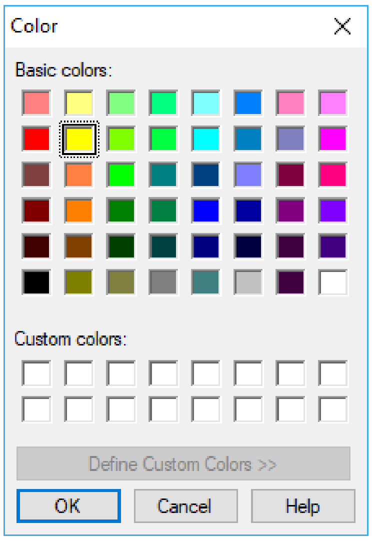
Event Name	Description
HelpRequest	Occurs when the user **clicks the Help button** on a common dialog box.

Example:

```
Public Class _2_ColorDialogExample
      Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
      System.EventArgs) Handles Button1.Click
            ColorDialog1.AllowFullOpen = False
            ColorDialog1.ShowHelp = True
            If ColorDialog1.ShowDialog() = Windows.Forms.DialogResult.OK
            Then
                  TextBox1.BackColor = ColorDialog1.Color
            End If
      End Sub
End Class
```

Output:





1.4 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.**
- User can also select font's color and can also **apply the current settings to selected text on a control** of the form without closing the dialog box, **by clicking Apply Button.**
- In addition to OK button, the font dialog box **may contain the Apply Button**, which reports the current string to your application.

Properties:

Property Name	Description
Color	It gets or sets the selected font color.
Font	It gets or sets the selected font.
FixedPitchOnly	Gets or sets a value indicating **whether the dialog box allows only the selection of fixed-pitch fonts**.
FontMustExist	Gets or sets a value indicating **whether the dialog box specifies an error condition if the user attempts to select a font or style that does not exist**.
ShowApply	Gets or sets a value indicating **whether the dialog box contains an Apply button.**
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.
ShowColor	Gets or sets a value indicating **whether the dialog box displays the color choice.**
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.

Events:

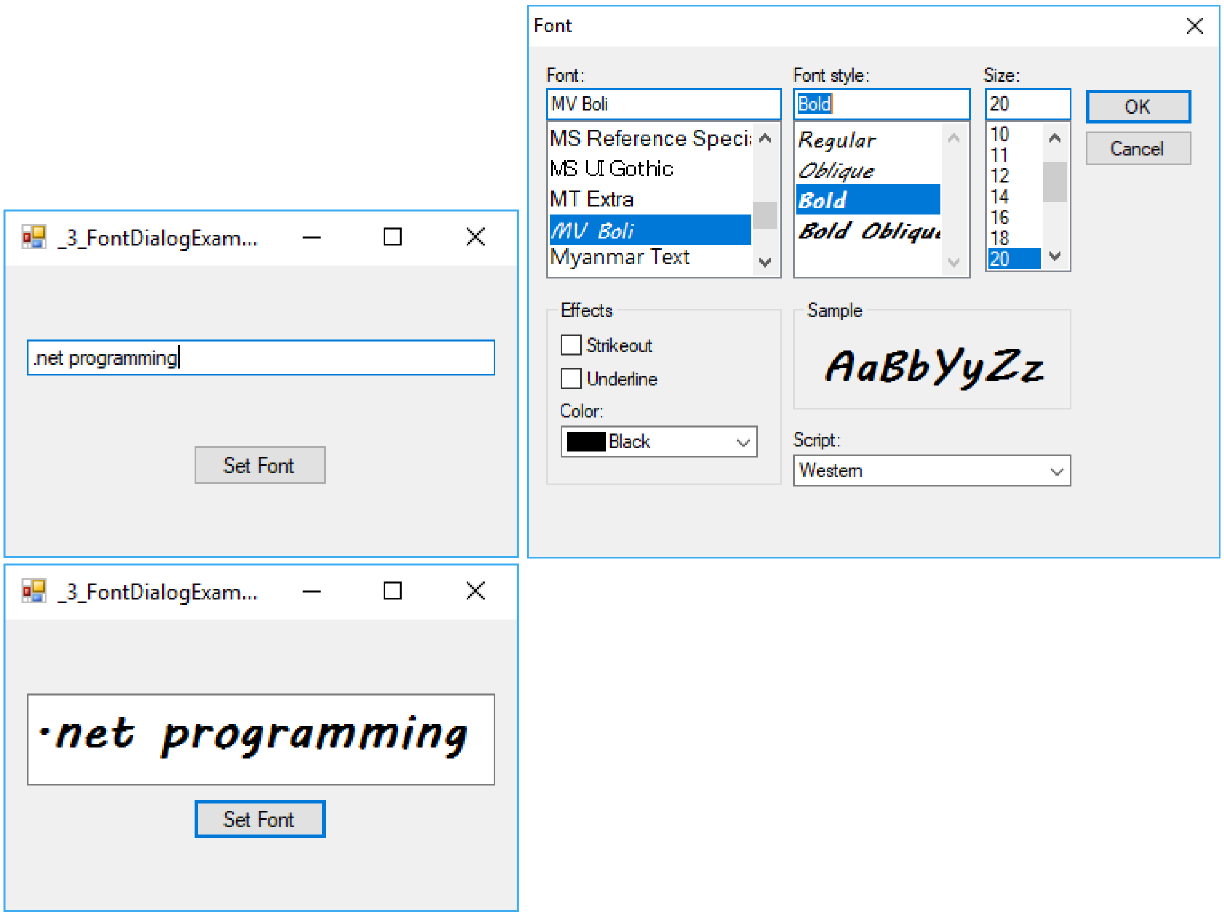
Event Name	Description
Apply	Occurs when the Apply button on the font dialog box is clicked.

Example:

```
Public Class _3_FontDialogExample
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles Button1.Click
        FontDialog1.ShowColor = True
```

```
            FontDialog1.Font = TextBox1.Font
            FontDialog1.Color = TextBox1.ForeColor

            If FontDialog1.ShowDialog <> DialogResult.Cancel Then
                'TextBox1.Text = FontDialog1.Font.Name
                TextBox1.Font = FontDialog1.Font
            End If
        End Sub
End Class
```

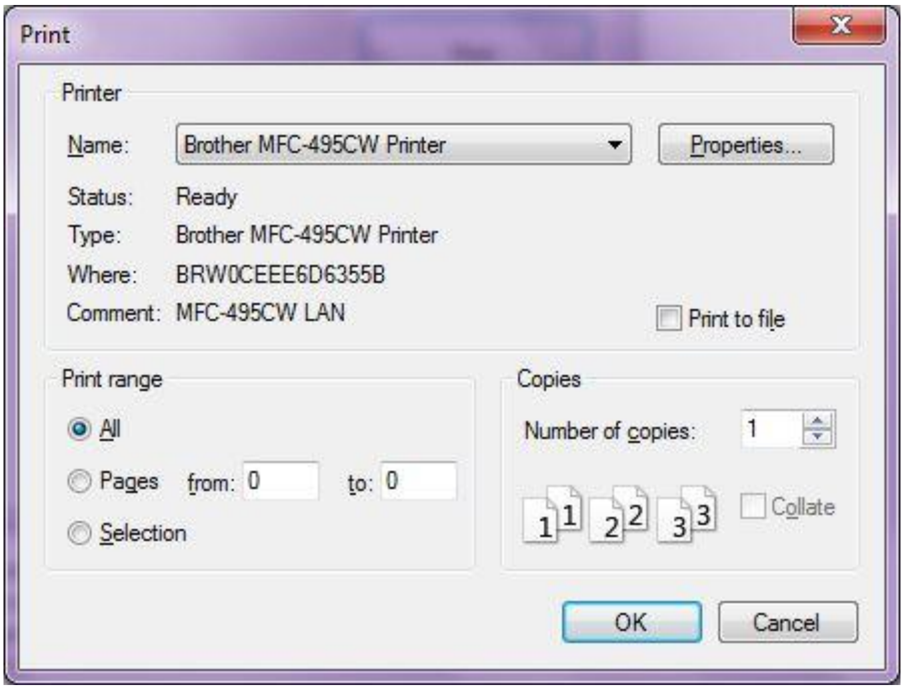
Output:



1.5 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:

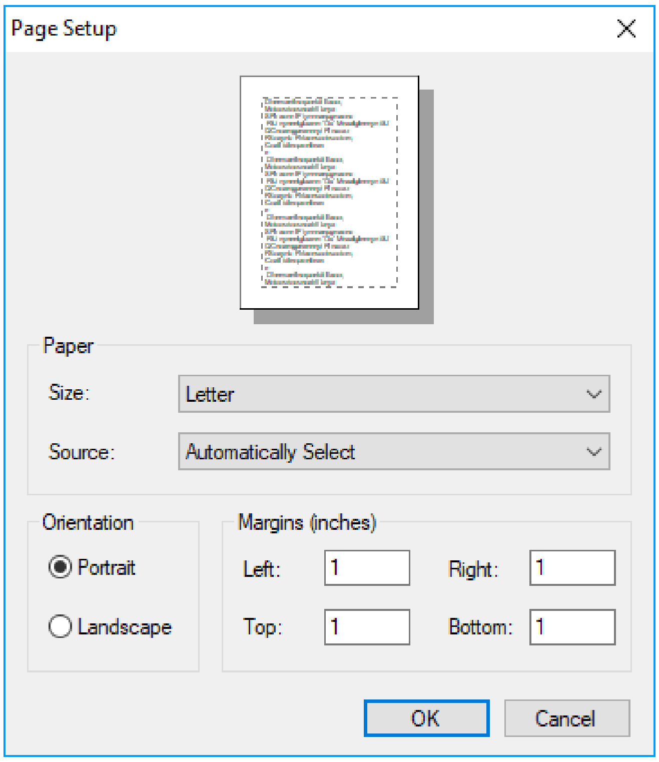
Property Name	Description
AllowCurrentPage	It gets or sets a value indicating **whether the Current Page option button** is displayed.
AllowPrintToFile	It gets or sets a value indicating **whether the Print to file check box is enabled.**
AllowSelection	It gets or sets a value indicating **whether the Selection option button is enabled.**
AllowSomePages	It gets or sets a value indicating whether the **Pages Option Button 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.

Example:

```
Public Class _4_PrintDialogExample
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles Button1.Click
```

```
            PrintDialog1.Document = PrintDocument1
            If PrintDialog1.ShowDialog = Windows.Forms.DialogResult.OK
            Then
                  PrintDocument1.PrinterSettings =
                  PrintDialog1.PrinterSettings
                  PrintDocument1.Print()
            End If
      End Sub
      Private Sub PrintDocument1_PrintPage(ByVal sender As System.Object,
      ByVal e As System.Drawing.Printing.PrintPageEventArgs) Handles
      PrintDocument1.PrintPage
            e.Graphics.DrawString(RichTextBox1.Text, RichTextBox1.Font,
            Brushes.Black, 100, 100)
      End Sub
      Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
      System.EventArgs) Handles Button2.Click
            PageSetupDialog1.Document = PrintDocument1
            PageSetupDialog1.Document.DefaultPageSettings.Color = False
            PageSetupDialog1.ShowDialog()
      End Sub
      Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As
      System.EventArgs) Handles Button3.Click
            PrintPreviewDialog1.Document = PrintDocument1
            PrintPreviewDialog1.ShowDialog()
      End Sub
End Class
```

Output:

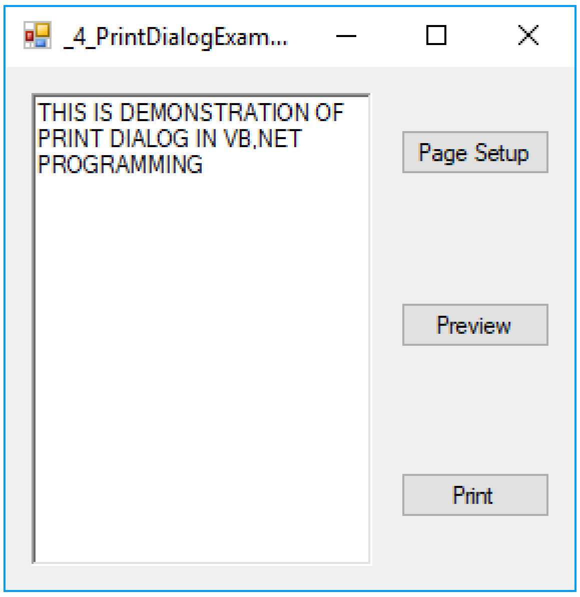


Page Setup

Paper
Size: Letter
Source: Automatically Select

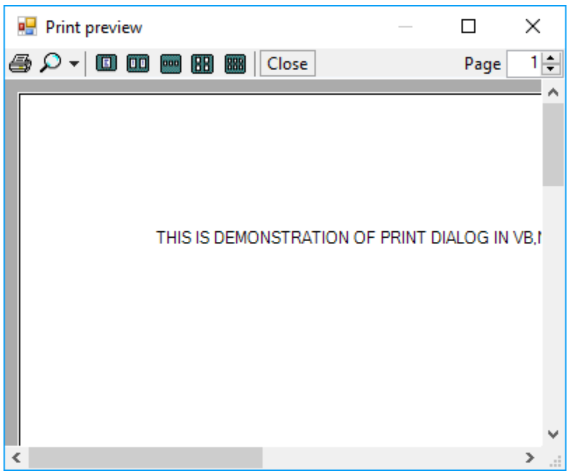
Orientation
[✓ Portrait] Landscape

Margins (inches)
Left: 1 Right: 1
Top: 1 Bottom: 1

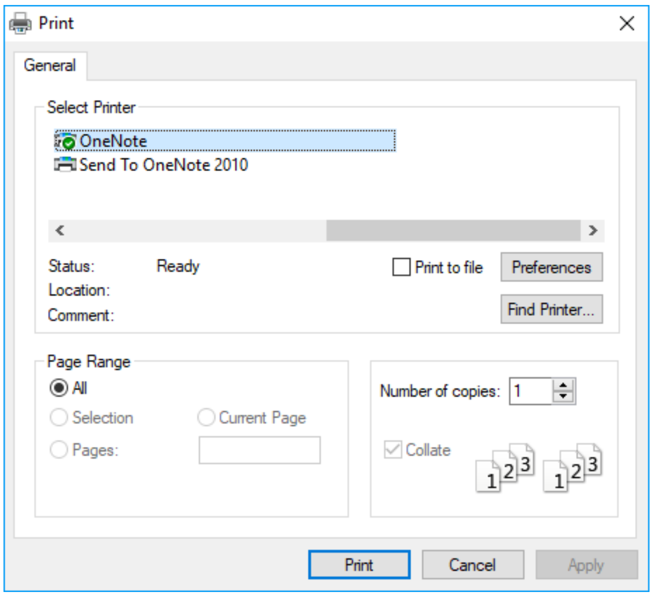


_4_PrintDialogExam...

THIS IS DEMONSTRATION OF PRINT DIALOG IN VB.NET PROGRAMMING



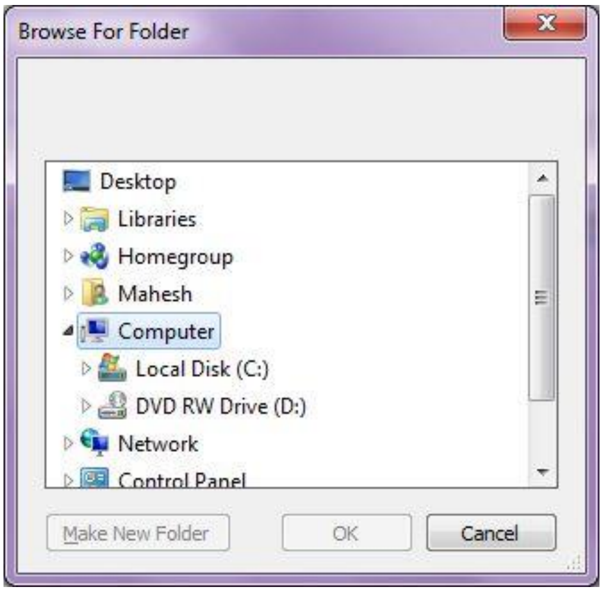
THIS IS DEMONSTRATION OF PRINT DIALOG IN VB.I



Status: Ready
Location:
Comment:
Number of copies: 1

1.6 Folder Browser Dialog Box:

- It prompts the user to **Select Folder**.
- Folder Browser Dialog Box provides a way to prompt the user to **browse, create and select a folder**.
- This is used when the user only need to **select folder, not the file**. Browsing of folder can be done through a tree control.
- Only folder from the File system is selected.



Properties:

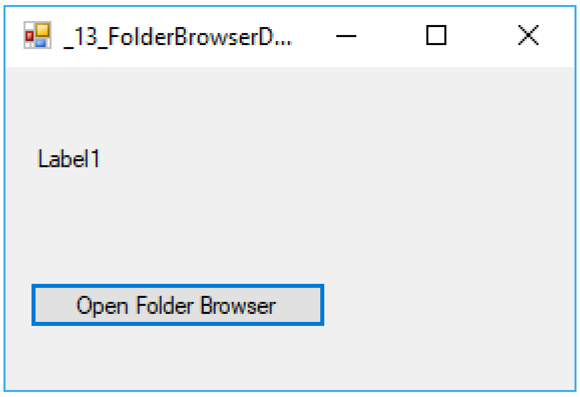
Property Name	Description
RootFolder	Gets or sets the root folder where the browsing starts from.
SelectedPath	Gets or sets the path selected by the user.
Description	Gets or sets the descriptive text displayed above the tree view control in the dialog box.

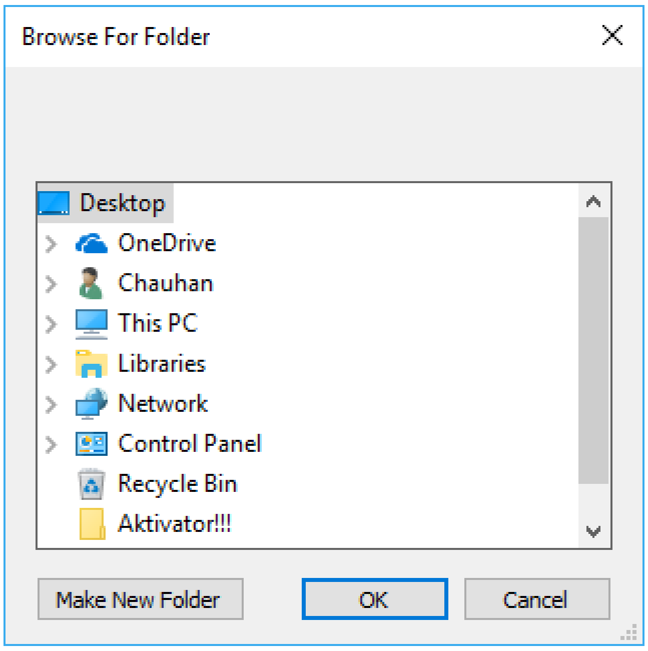
ShowNewFolder Button	Gets or sets a value indicating whether the New Folder button appears in the folder browser dialog box.

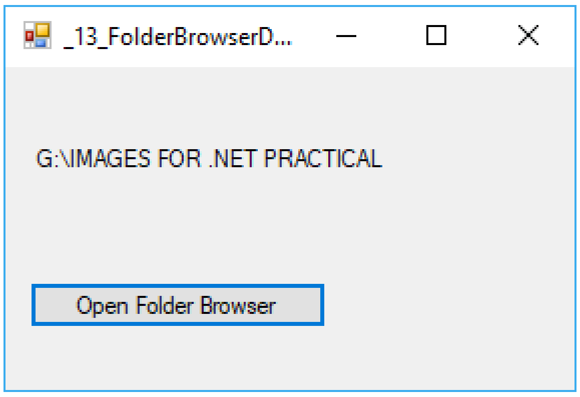
Example:

```
Public Class _13_FolderBrowserDialogExample
      Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
      System.EventArgs) Handles Button1.Click
            If FolderBrowserDialog1.ShowDialog() =
            Windows.Forms.DialogResult.OK Then
                  Label1.Text = FolderBrowserDialog1.SelectedPath
            End If
      End Sub
End Class
```

Output:







```
G:\IMAGES FOR .NET PRACTICAL
```

2. SUB PROCEDURE AND FUNCTIONS:

2.1 Procedure

- 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.

- All executable **code** in VB.NET must be **within some procedure**. We **can invoke the procedure** from some other place in the code. This is **known as procedure call.**
- When the procedure finish running, it **returns control** to the code that invoked it, and is called as **calling code**.
- The **calling code is statement**, or an expression within a statement, that specifies the procedure by name and transfer control to it.

Procedure includes Return Statement:

- With the return statement, **control returns immediately** to the calling code.
- **Statement** comes **after the return** statements are **not executed**.
- You can **use more than one return** statement in the procedure.

Procedure includes Exit Statement:

- With Exit Sub or Exit Function, **control returns immediately to the calling code.**
- Statements after the **Exit Statement are not executed.**
- You can use more than **one Exit statement** in the procedure. And **mixing of Return and Exit** statements are **allowed**.

Procedure not has Return or Exit Statement:

- It completes with and **End Sub or End function** statement, following the last statement of the procedure body.
- The End statement **returns control immediately to the calling code**. You **can have only one End** statement in a procedure.

2.2 Sub Procedure

- Sub procedures are procedures that **do not return any value to the calling code**. A sub procedure is set of VB **statement enclosed within Sub and End Sub.**
- After performing the task, then **control is returned to the calling code**, but it does not return any value to the calling code.
- **Execution** of Sub is **starter from the first statement** after the Sub Statement and ending with the first End Sub, Exit Sub.

Syntax:

```
[Modifiers] Sub SubName [(ParameterList)]
        [Statements]
End Sub
```

Where,

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.

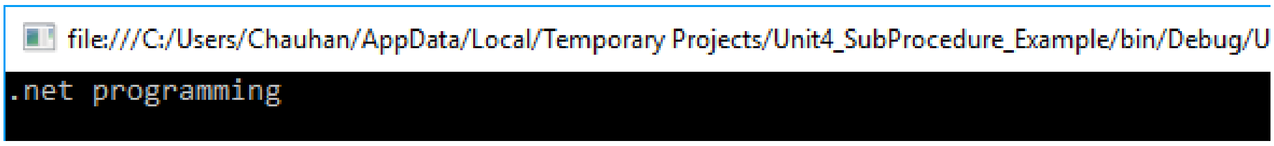
SubName − indicates the name of the Sub

ParameterList − specifies the list of the parameters

Example:

```
Module Module1
      Public Sub Print(ByVal strName As String)
            Console.WriteLine(strName)
            Console.ReadKey()
      End Sub
      Sub Main()
            Call Print(".net programming")
      End Sub
End Module
```

Output:



```
file:///C:/Users/Chauhan/AppData/Local/Temporary Projects/Unit4_SubProcedure_Example/bin/Debug/U
.net programming
```

Calling Sub Procedure:

- We can call a **sub procedure by a standalone calling statement.**
- We cannot call it by using its name in an expression.
- If no argument is passed than, we can omit the parenthesis.

Syntax:

Call Obect.Tostring()

- The use of Call keyword is optional.

2.3 Function Procedure

- Function procedure is **set of VB statement** enclosed within **Function and End Function**.
- After performing the task, then **control is returned to the calling code**, when it returns control, **it return value to the calling code**.
- Each time the **function is called**, its statements run, **starting with the first executable statement** after the Function statement and **ending with the first End Function**, **Exit Function**, or **Return statement** encountered.
- You can define a Function procedure in a module, class, or structure. It is **Public by default,** which means you can **call it from anywhere in your application** that has access to the module in which you defined it.
- A Function procedure **can take arguments,** such as constants, variables, or expressions, which are **passed to it by the calling code.**
- 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.**

Syntax:

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

Where

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.

Example:

```
Function yesterday() As Date
        …….
End Function


Function findSqrt(ByVal ans As Integer) As Integer
        ……..
End Function
```

Return Statement

- The value a Function procedure sends back to the calling code is called its return value.
- It uses the Return statement to specify the return value, and returns control immediately to the calling program. The following example illustrates this.

Syntax:

```
Function FunctionName [(ParameterList)] As ReturnType
        ' The following statement immediately transfers control back
        ' to the calling code and returns the value of Expression.
        Return Expression
End Function
```

Calling Function Procedure:

- We can invoke a Function **Procedure by including its name and arguments** either on the right side of an assignment statement or in an expression.
- You must **provide values for all arguments** that are not optional, and you must enclose the argument list in parentheses.
- If no arguments are supplied, you can optionally omit the parentheses.

Syntax:

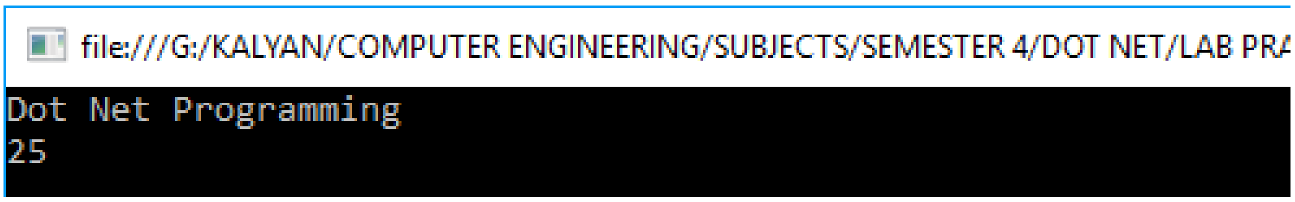
```
lvalue = function_name[( argument_list )]
```

- When you call a Function procedure, you do **not have to use its return value**. If you do not, all the actions of the function are performed, **but the return value is ignored**.

Program:

```
Module Module1
      Sub Main()
            Console.Write(getname())
            Console.WriteLine()
            Console.WriteLine(area(5))
            Console.ReadKey()
      End Sub

      Public Function getname() As String
            Return "Dot Net Programming"
      End Function

      Function area(ByVal side1 As Integer) As Integer
            Return (side1 * side1)
      End Function
End Module
```

Output:



```
file:///G:/KALYAN/COMPUTER ENGINEERING/SUBJECTS/SEMESTER 4/DOT NET/LAB PRA
Dot Net Programming
25
```

2.4 Parameters of Function Procedure and Sub Procedure

- A parameter represents a **value that the procedure expects you to supply** when you call it.
- You can **declare** each procedure parameter **similarly to declare a variable**, specifying the parameter name and data type.
- You can define a procedure with **no parameters, one parameter, or more than one**. The part of the procedure definition that **specifies the parameters is called the parameter list.**
- You declare each procedure parameter similarly to how you declare a variable, specifying the parameter name and data type. You can also **specify the passing mechanism**, and whether the **parameter is optional or a parameter array.**

Syntax:

[Optional] [ByVal | ByRef] [Parameter_Array] parameter_name As datatype

- If the parameter is optional, you must also **supply a default value** as part of its declaration.

Syntax:

Optional [ByVal | ByRef] parameter_name As data_type = default_value

2.5 Passing Arguments to Procedure

- When we use the parameters in the procedure, **we need to pass argument while calling** the function. When a variable is passed to procedure, it is called as argument.
- We can pass argument to the procedure either **Pass by value** or **Pass by reference**. Default argument and optional argument can also be pass to the procedure.

Passing argument by Value:

- Only **copy of a variable is passed** when an argument is passed by value.
- If the procedure changes the value, the change affects only the copy of and not the variable itself.
- By using **ByVal keyword** to indicate an argument passed by value.

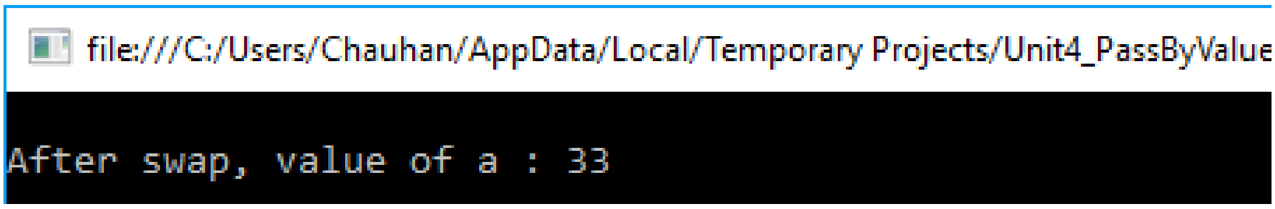
Example:

```
Public Sub print(ByVal variable as Integer)
        …….
End Sub
```

Program:

```
Module Module1
    Dim result As Integer
    Sub sum(ByVal x As Integer, ByVal y As Integer)
        result = x + y
    End Sub
    Sub Main()

        Dim a As Integer = 11
        Dim b As Integer = 22
        sum(a, b)
        Console.WriteLine()
        Console.WriteLine("After swap, value of a : {0}", result)
        Console.ReadLine()

    End Sub
End Module
```

Output:



```
file:///C:/Users/Chauhan/AppData/Local/Temporary Projects/Unit4_PassByValue

After swap, value of a : 33
```

Passing argument by Reference:

- Passing arguments by reference gives the procedure **access to the actual variable in its memory address location**. A reference parameter is a reference to a memory location of a variable.
- The reference parameters represent the same memory location as the actual parameters that are supplied to the method.
- As a result, variables **value can be permanently change by the procedure** to which it passed.
- 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.

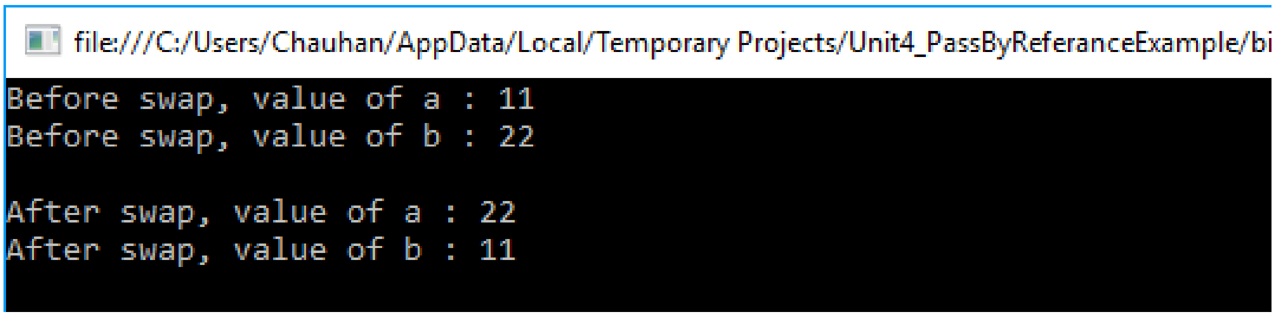
Example:

```
Public Sub print(ByRef variable as Integer)
        …….
End Sub
```

Program:

```
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:



```
file:///C:/Users/Chauhan/AppData/Local/Temporary Projects/Unit4_PassByReferanceExample/bi
Before swap, value of a : 11
Before swap, value of b : 22

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

With Optional Argument:

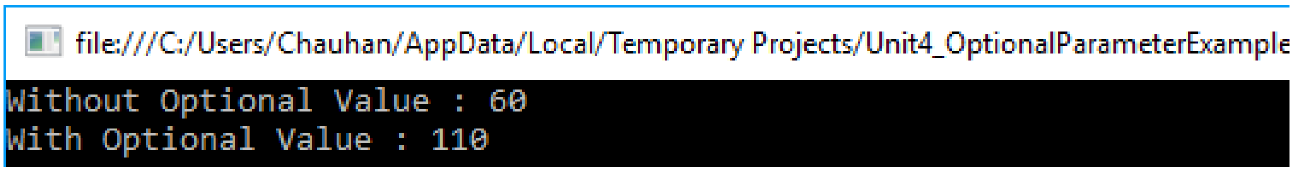
- We can specify argument to a procedure as **optional by placing the OPTIONAL keyword** in the argument list.
- Put OPTIONAL keyword **before the parameter name**, so the parameter is now become optional. After declare one parameter as optional, you must declare every sub sequent parameters as optional.

Program:

```
Module Module1
     Public Function calculate(ByVal x As Integer, Optional ByVal y As
     Integer = 100) As Integer
          x = y + x
```

```
            Return x
        End Function
        Sub Main()
            Dim a As Integer = 10
            Dim b As Integer
            b = calculate(a, 50)
            Console.WriteLine("Without Optional Value : {0}", b)
            b = calculate(a)
            Console.WriteLine("With Optional Value : {0}", b)
            Console.ReadLine()
    End Sub
End Module
```

Output:



file:///C:/Users/Chauhan/AppData/Local/Temporary Projects/Unit4_OptionalParameterExample

```
Without Optional Value : 60
With Optional Value : 110
```

3. EXCEPTION HANDLING IN VB.NET:

- Exceptions are the **occurrence of some condition** that **changes the normal flow of execution**.
- For example, you programme run out of memory (shortage), file does not exist in the given path, network connections are dropped etc.
- More specifically for better understanding, we can say it as **Runtime Errors.**
- **Exception is handled** by saving the current state of execution in a predefined place and **switching the execution to a specific block** of code known as **exception handler**.
- Exception handling **provides a way to run application** with **solving the run time errors** without terminating or exit applications.
- Different type of exception that are commonly occurs in the applications are divide by zero, resource file location changed/deleted, database server shut down etc.
- In VB.Net, when exception is occurs in any method, which does not have exception handler, than program control transferred back to the calling function or to the previous function, and checks that if previous function have the exception handler or not.
- This process repeats till the exception handler is not found.
- If no exception handler is found then, an error message is displayed and the application is terminated or stopped.

3.1 Structured Exception Handling

- In .NET languages, Structured Exceptions handling is a fundamental part of CLR.
- All exceptions in the CLR, are derived from a single base class.
- In Structured Exception Handling, block of code are encapsulated, and with each block exception handlers are associated.
- Each handler specifies some of the filter condition on the type of exception it handles.

- When exception is occur in the code block, set of handlers are searched for handling the exception, and the first one with a matching filter condition is executed.
- One method can have more than one exception handler, and we can also nested the code blocks within each other.
- 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 for Structured Exception Handling.

Keyword	Description
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**. The **catch** keyword indicates the **catching of an exception.**
Finally	The Finally block is used to **execute a given set of statements**, whether an **exception is thrown or not thrown.** That means if you write a finally block, the **code should execute after the execution** of try block or catch block. Example: If you open a file, it must be closed whether an exception is raised or not.
Throw	A program throws an exception **when a problem shows up**. This is done using a Throw keyword.

Syntax:

```
Try
        [Statements]
        [Exit Try]

Catch [Exception [ As type ] ] [when expression]
        [Catch Statements ]
        [Exit Try]
[Catch…]

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

How Try…Catch…Finally handles exception:

- **Try Block** contains the code is monitored for the exception.
- If error is occurred during the execution of the try block, VB checks all **catch statement** until it finds one catch statement with a condition that matches with that error.
- If one **Catch block is found**, **control is transferred** to the first line of code in that catch block.
- If **no matching Catch** is found in that Try…Catch…Finally block, than the **search proceed to the**

catch statements of the outer Try…Catch…Finally block, that contains the block in which the exception is occurred.

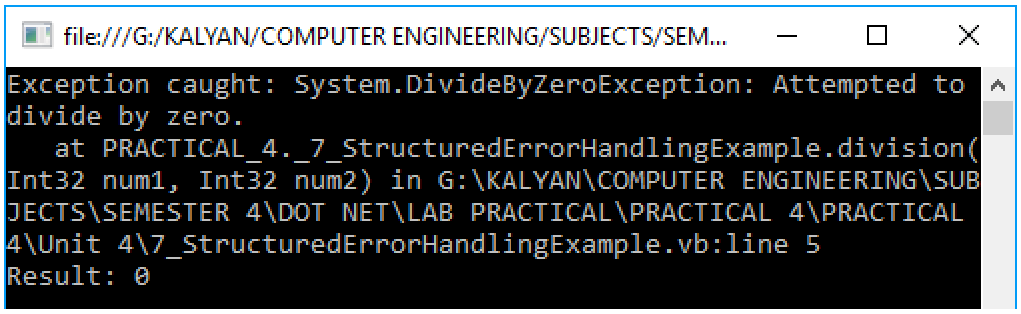
- This process **repeats till the matching catch** is not found in current procedure.
- If **no match** found than **error is produced.**
- You can list down **multiple catch statements** to catch different type of exceptions in case your **try block raises more than one exception** in different situations.
- The code in **Finally** block is always executes last, regardless of whether the code in the catch block has executed. This section is commonly used for clean-up code like closing file, releasing objects etc.
- Below table represents some predefined exception classes which are derived from the System.SystemException class.

Class Name	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.ArrayTypeMismatchException	Handles error generated when type is mismatched with the array type.
System.DivideByZeroException	It handles errors generated from dividing a dividend with zero.
System.OutOfMemoryException	Handles error generated from insufficient free memory.
System.NullReferenceException	It handles errors generated from dereferencing a null object.
System.InvalidCastException	It handles errors generated during typecasting.
System.StackOverflowException	Handles error generated from stack overflow.

Program:

```
Module _7_StructuredErrorHandlingExample
    Sub division(ByVal num1 As Integer, ByVal num2 As Integer)
        Dim result As Integer = 0
        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:



file:///G:/KALYAN/COMPUTER ENGINEERING/SUBJECTS/SEM...

```
Exception caught: System.DivideByZeroException: Attempted to
divide by zero.
   at PRACTICAL_4._7_StructuredErrorHandlingExample.division(
Int32 num1, Int32 num2) in G:\KALYAN\COMPUTER ENGINEERING\SUB
JECTS\SEMESTER 4\DOT NET\LAB PRACTICAL\PRACTICAL 4\PRACTICAL
4\Unit 4\7_StructuredErrorHandlingExample.vb:line 5
Result: 0
```

3.2 Un-Structured Exception Handling

- Unstructured exception handling is **implemented using the Err object** and three statements: **On Error, Resume, and Error**.
- The **On Error statement** establishes a single exception handler that **catches all thrown exceptions**; you can subsequently **change** the handler **location**, but you can **only have one handler at a time.**
- The **method keeps track of the most recently thrown exception** as well as the most recent **exception-handler location.**

Property of Err Object:

Property Name	Description
Description	Text message providing a **short description of the error.**
HelpContext	Integer containing the **context ID** for a **topic in a Help file.**
HelpFile	String expression containing the **fully qualified path to a Help file.**
Number	Numeric value specifying an error.
Source	String expression **representing the object** or application **that generated the error.**

- In Unstructured exception handling, you can **place an On Error** statement **at the beginning** of a block of code, and it **handles any error occurring within that block**.
- When an **exception is raised** in a procedure **after the On Error** statement executes, the program control **transferred to the line argument** specified in the On Error statement.
- The line argument, which is **line number** or line label, indicates the **exception handler location**.
- The **On Error** statement is used **specifically for unstructured exception handling**. In unstructured exception handling, **On Error is placed** at the **beginning of a block** of code.
- It then has “scope” over that block, it handles any error occurring within the block.
- If a program encounters another On Error statement, that statement becomes valid and the first statement becomes invalid.
- Sometimes a call is made from the original procedure to another procedure, and an exception is occurred in the called procedure.
- In such cases, if the called procedure does not handle the exception, the exception propagates back to the calling procedure, and execution branches to the line argument.

Syntax:

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

- The On **Error GoTo statement** enables a routine for exception handling and **specifies the location** of that routine within the procedure.
- Used **with a label or line number**, it directs the code to a specific exception handling routine.
- Used with **-1, it disables error handling** within the procedure.
- Used with **0, it disables the current exception**.
- If there is no On Error statement and the exception is not handled by any methods in the current call stack, then any run-time error that occurs is fatal: execution stops and an error message are displayed.

Statement	Description
On Error GoTo -1	Resets Err object to Nothing, **disabling error handling in the routine**
On Error GoTo 0	Disables the enabled error handler in the current procedure Or Resets last exception-handler location to Nothing, disabling the exception.
On Error GoTo <label_name>	Sets the **specified label as the location of the exception handler**. Calls the error-handling code that starts at the line specified at line. Here, line is a line label or a line number. **If a runtime error occurs, program execution goes to the given location**. The specified line must be in the same procedure as the On Error statement.
On Error Resume Next	Establishes the **Resume Next behaviour as the location of the most recent exception handler** or Specifies that when an exception occurs, execution skips over the statement that caused the problem and goes to the statement immediately following. Execution continues from that point.

3.2.1 On Error GoTo Line:

- Enables the error handling routine that starts at line specified in the required line argument.
- The line argument is any line number or line label.
- When runtime error occurs in the code, program control transferred to that line and making the error handler active.
- The specified line must be in the same procedure as the On Error statement, otherwise a compile time error will occur.

Syntax:

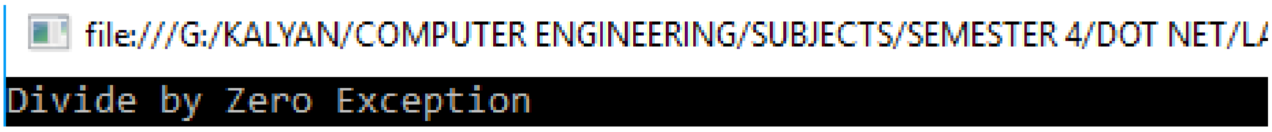
```
Module Module1
    Sub Main()
        On Error GoTo Handler
        .
        .
```

```
                    Exit Sub
                    Handler:
                End Sub
        End Module
```

Program:

```
Module _8_OnErrorGoToLineExample
        Sub Main()
                Dim a As Integer = 100
                Dim b As Integer = 0
                Dim result As Integer
                On Error GoTo errHandler
                result = a / b
                Exit Sub
        errHandler:
                Console.WriteLine("Divide by Zero Exception")
                Console.ReadKey()
        End Sub
End Module
```

Output:



```
file:///G:/KALYAN/COMPUTER ENGINEERING/SUBJECTS/SEMESTER 4/DOT NET/LA
Divide by Zero Exception
```

Note that we've used an Exit Sub statement here so that in normal execution, the procedure stops before reaching the error-handling code that follows the Handler label.

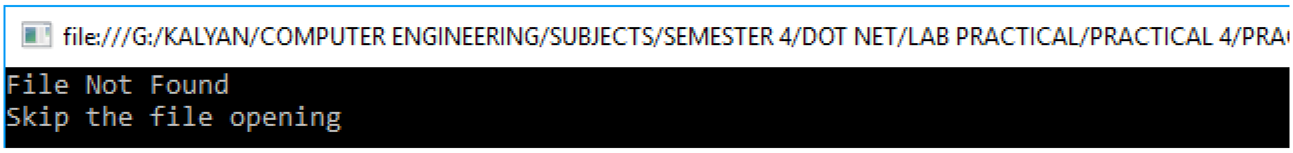
3.2.2 On Error GoTo Resume Next:

- This statement provides a way to resume the execution of program even after the exception is occurred.
- We can use "Resume statement" to resume execution with the statement that cause exception is occurred.
- "Resume Next" to resume execution with the statement after the one that caused the exception. And Resume line, where the line is line number or label that specifies where to resume the execution of program.

Program:

```
Module _9_OnErrorResumeNextExample
        Sub Main()
                On Error GoTo errHandler
                'try to open file which does not exist
                System.IO.File.Open("Invalid File Name", IO.FileMode.Open)
                Console.WriteLine("Skip the file opening")
                Console.ReadKey()
                Exit Sub
        errHandler:
                Console.WriteLine("File Not Found")
                Resume Next
        End Sub
End Module
```

Output:



file:///G:/KALYAN/COMPUTER ENGINEERING/SUBJECTS/SEMESTER 4/DOT NET/LAB PRACTICAL/PRACTICAL 4/PRA

```
File Not Found
Skip the file opening
```

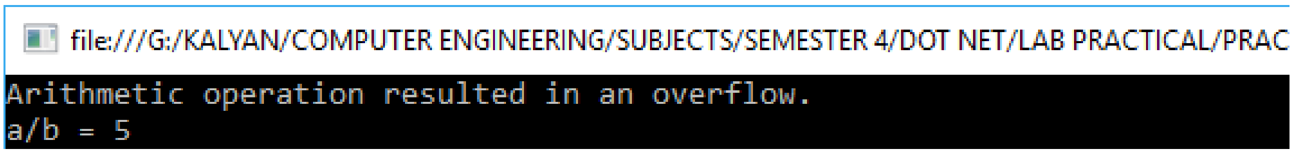
3.2.3 On Error GoTo 0:

- The On Error GoTo 0 Statement disables any error handler in the current procedure.
- If On Error GoTo 0 Statement is not included, the error handler is still disabled when the procedure containing the exception handler ends.

Program:

```
Module _10_OnErrorGoTo0Example
      Sub Main()
            Dim a As Integer = 100
            Dim b As Integer = 0
            Dim result As Integer
            On Error GoTo DivideByZero
            result = a / b
            'this line disables exception handler
            On Error GoTo 0
            Console.WriteLine("a/b = " & result)
            Console.ReadKey()
            'Exit sub routine before error handling conde, if not done
            than result in unexcepted output
            Exit Sub
      DivideByZero:
            Console.WriteLine(Err.Description)
            b = 20
            'Solution to the error, so program execution is resume from
            the point where error is occurred
            Resume
      End Sub
End Module
```

Output:



file:///G:/KALYAN/COMPUTER ENGINEERING/SUBJECTS/SEMESTER 4/DOT NET/LAB PRACTICAL/PRAC

```
Arithmetic operation resulted in an overflow.
a/b = 5
```

3.2.4 On Error GoTo -1:

- The On Error GoTo -1 Statement disables any exception handler in the current procedure.
- If On Error GoTo -1 Statement is not included, the exception is automatically disabled when its procedure ends.

4. MULTIPLE DOCUMENT INTERFACE (MDI):

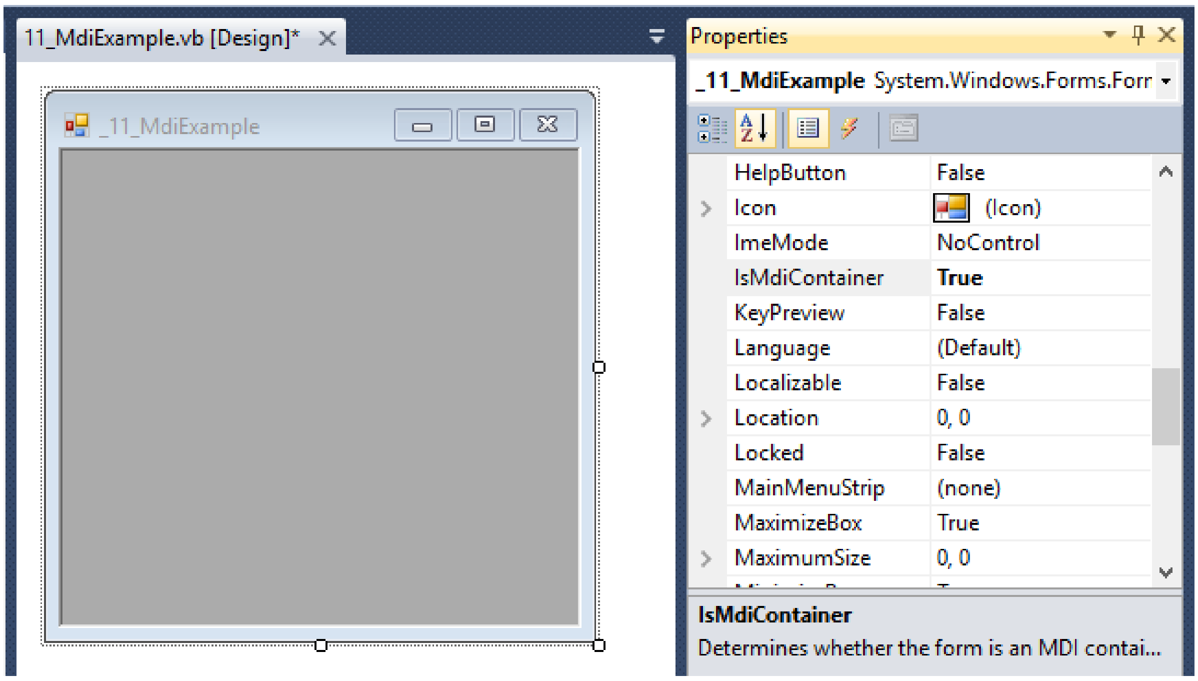
- The multiple-document interface (MDI) allows you to create an application that **displays**

multiple documents or forms 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.
- It **maintains multiple forms** within a **single container form**.
- MDI application have **windows menu item** with sub menu, which is **used for switching** between the windows or documents.
- Applications such as **Microsoft Excel and Microsoft Word** for Windows have multiple-document interfaces.
- 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.
- Any form becomes a **MDI parent form**, for that we have to set **IsMdiContainer property to True**.
- 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.

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.
- Select the form and open property window after that set **IsMDIContainer property value to true**.
- Drag and drop **Menustrip** in the form.
- **Create Menu** and Sub menu.
- This will show in below figures.



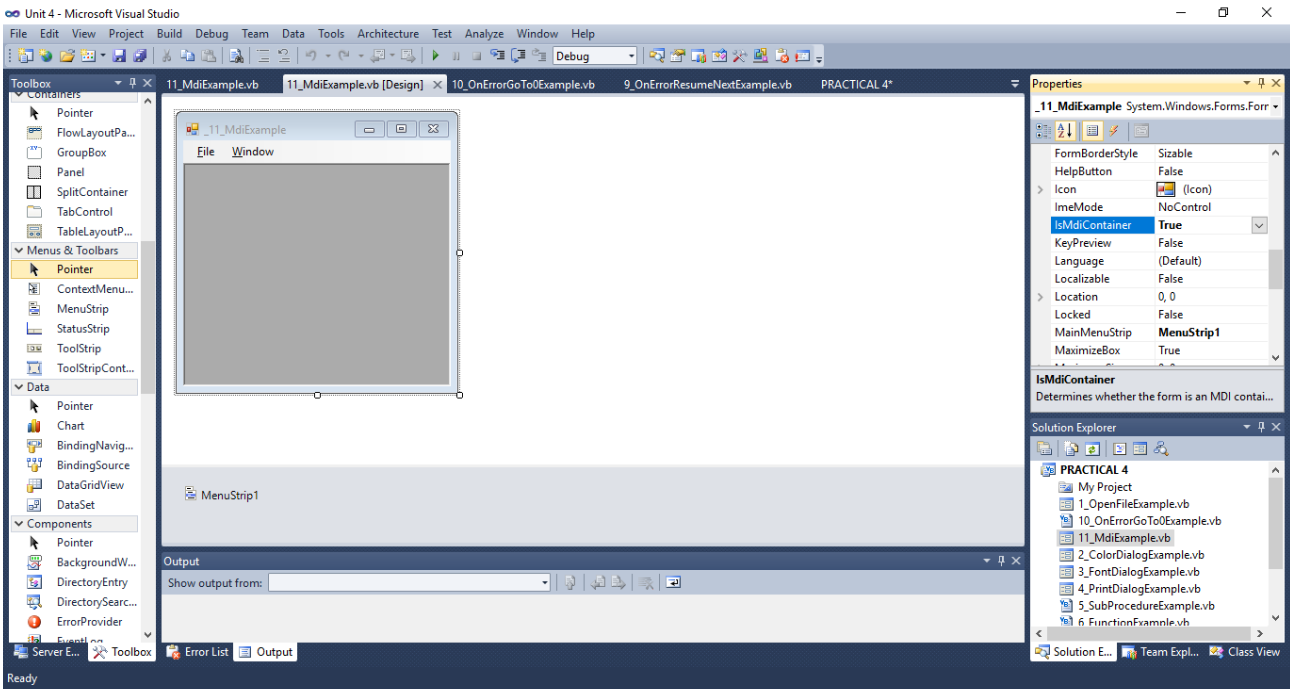
_11_MdiExample System.Windows.Forms.Forr

HelpButton	False
Icon	(Icon)
ImeMode	NoControl
IsMdiContainer	**True**
KeyPreview	False
Language	(Default)
Localizable	False
Location	0, 0
Locked	False
MainMenuStrip	(none)
MaximizeBox	True
MaximumSize	0, 0

IsMdiContainer
Determines whether the form is an MDI contai...

- Select the MDI form and set the following properties of the MDI form.

  IsMdiContainer = True



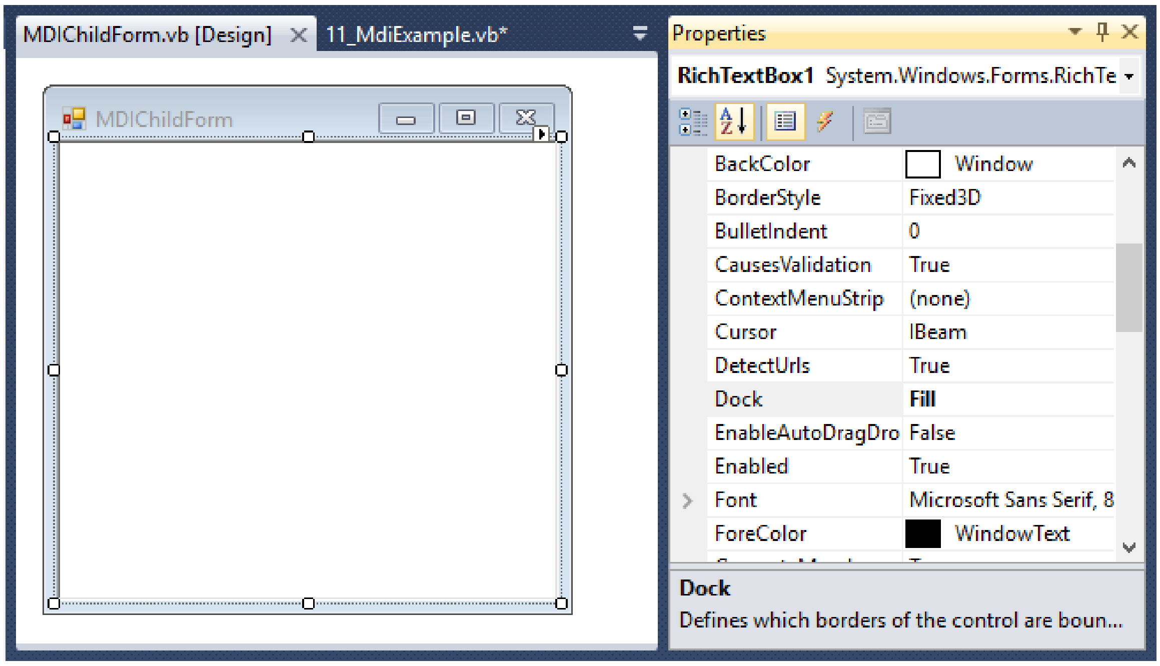
Unit 4 - Microsoft Visual Studio

Properties: _11_MdiExample System.Windows.Forms.Form

FormBorderStyle	Sizable
HelpButton	False
Icon	(Icon)
ImeMode	NoControl
IsMdiContainer	True
KeyPreview	False
Language	(Default)
Localizable	False
Location	0, 0
Locked	False
MainMenuStrip	MenuStrip1
MaximizeBox	True

IsMdiContainer
Determines whether the form is an MDI contai...

- **Add another form** to the application, set the below property.
- **Add Rich Text Box tool** in the MDI Child Form.



Properties: RichTextBox1 System.Windows.Forms.RichTe

BackColor	Window
BorderStyle	Fixed3D
BulletIndent	0
CausesValidation	True
ContextMenuStrip	(none)
Cursor	IBeam
DetectUrls	True
Dock	Fill
EnableAutoDragDro	False
Enabled	True
Font	Microsoft Sans Serif, 8
ForeColor	WindowText

Dock
Defines which borders of the control are boun...

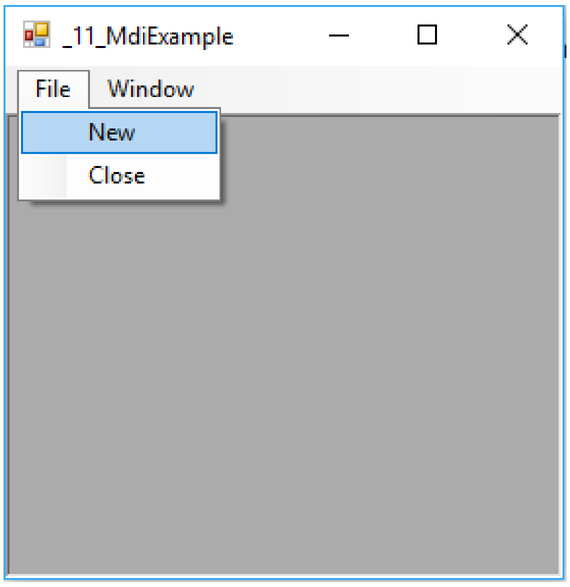
- Generate the click event of the menu strip for NEW menu item.

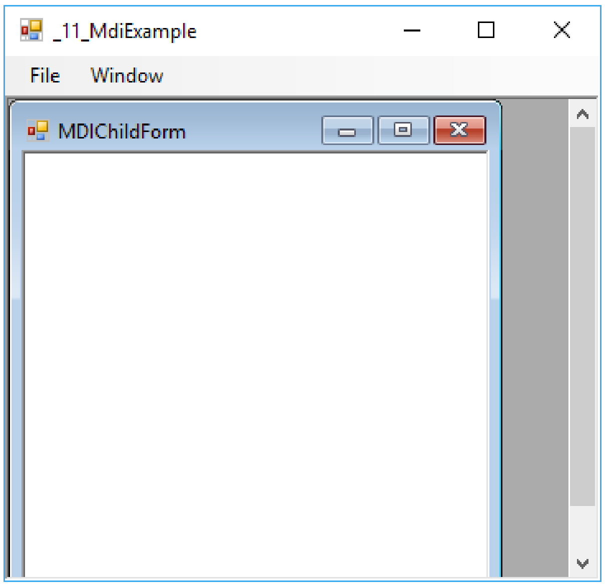
Program:

```
Public Class _11_MdiExample
    Private Sub NewToolStripMenuItem_Click(ByVal sender As
    System.Object, ByVal e As System.EventArgs) Handles
    NewToolStripMenuItem.Click
```

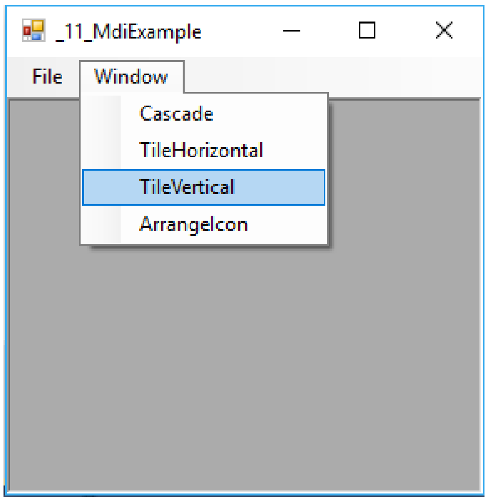
```
            Dim NewMdiChild As New MDIChildForm()
            NewMdiChild.MdiParent = Me
            NewMdiChild.Show()
        End Sub
End Class
```

Output:





- In this situation, we can **generate child forms based on our requirements**. But these child forms are **not arranged in proper manner**.
- We can **set layout for the child** forms by using **LayoutMdi method**.
- There are different **MDILayout enumeration values** to arrange the child forms such as **Tile, Cascade (Vertical or Horizontal) and Arranged** which is used by LayoutMdi method.
- Add this value in the submenu of Windows menu. As shown in figure.

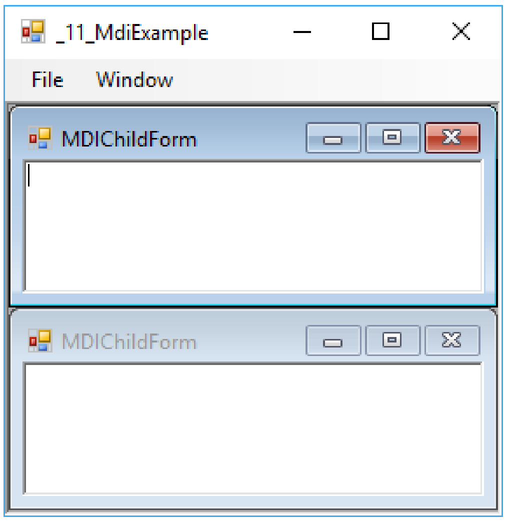


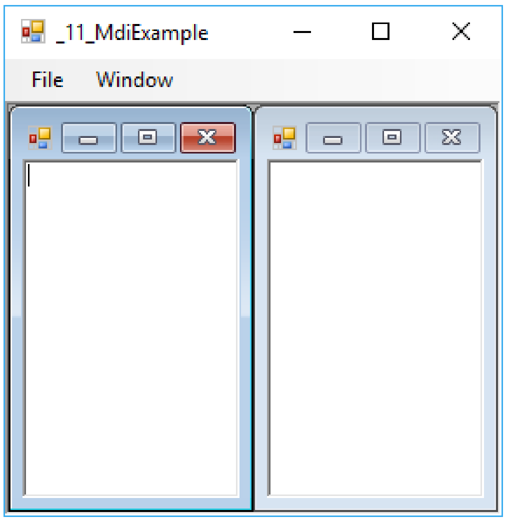
Program:

```
Public Class _11_MdiExample
        Private Sub NewToolStripMenuItem_Click(ByVal sender As
        System.Object, ByVal e As System.EventArgs) Handles
        NewToolStripMenuItem.Click
```

```
            Dim NewMdiChild As New MDIChildForm()
            NewMdiChild.MdiParent = Me
            NewMdiChild.Show()
        End Sub

        Private Sub CascadeToolStripMenuItem_Click(ByVal sender As
        System.Object, ByVal e As System.EventArgs) Handles
        CascadeToolStripMenuItem.Click
            Me.LayoutMdi(MdiLayout.Cascade)
        End Sub

        Private Sub TileHorizontalToolStripMenuItem_Click(ByVal sender As
        System.Object, ByVal e As System.EventArgs) Handles
        TileHorizontalToolStripMenuItem.Click
            Me.LayoutMdi(MdiLayout.TileHorizontal)
        End Sub

        Private Sub TileVerticalToolStripMenuItem_Click(ByVal sender As
        System.Object, ByVal e As System.EventArgs) Handles
        TileVerticalToolStripMenuItem.Click
            Me.LayoutMdi(MdiLayout.TileVertical)
        End Sub

        Private Sub ArrangeIconToolStripMenuItem_Click(ByVal sender As
        System.Object, ByVal e As System.EventArgs) Handles
        ArrangeIconToolStripMenuItem.Click
            Me.LayoutMdi(MdiLayout.ArrangeIcons)
        End Sub
End Class
```

Output:



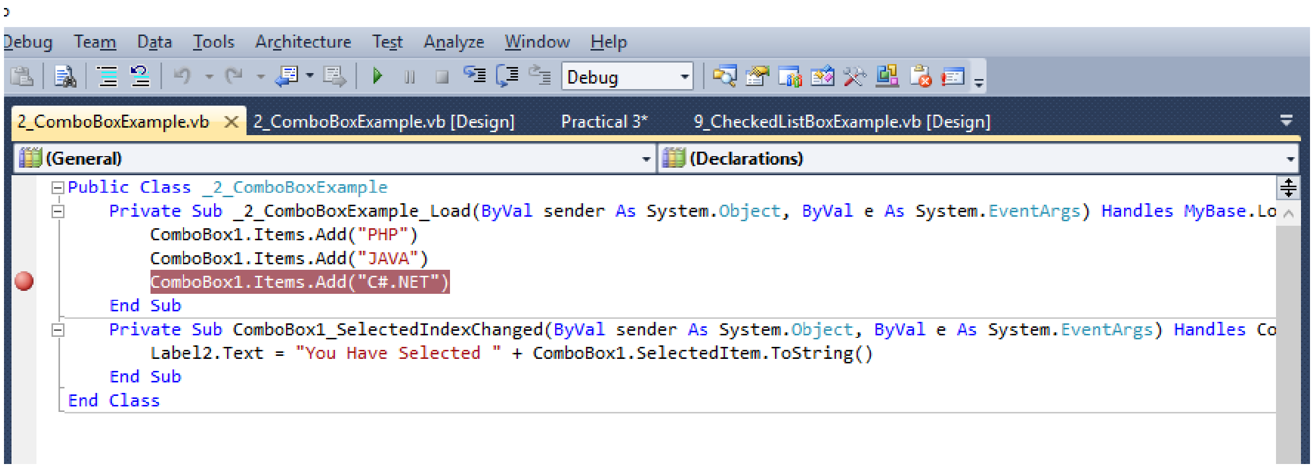


5. DEBUGGING APPLICATION:

Steps to debug vb.net application with Watch Window:

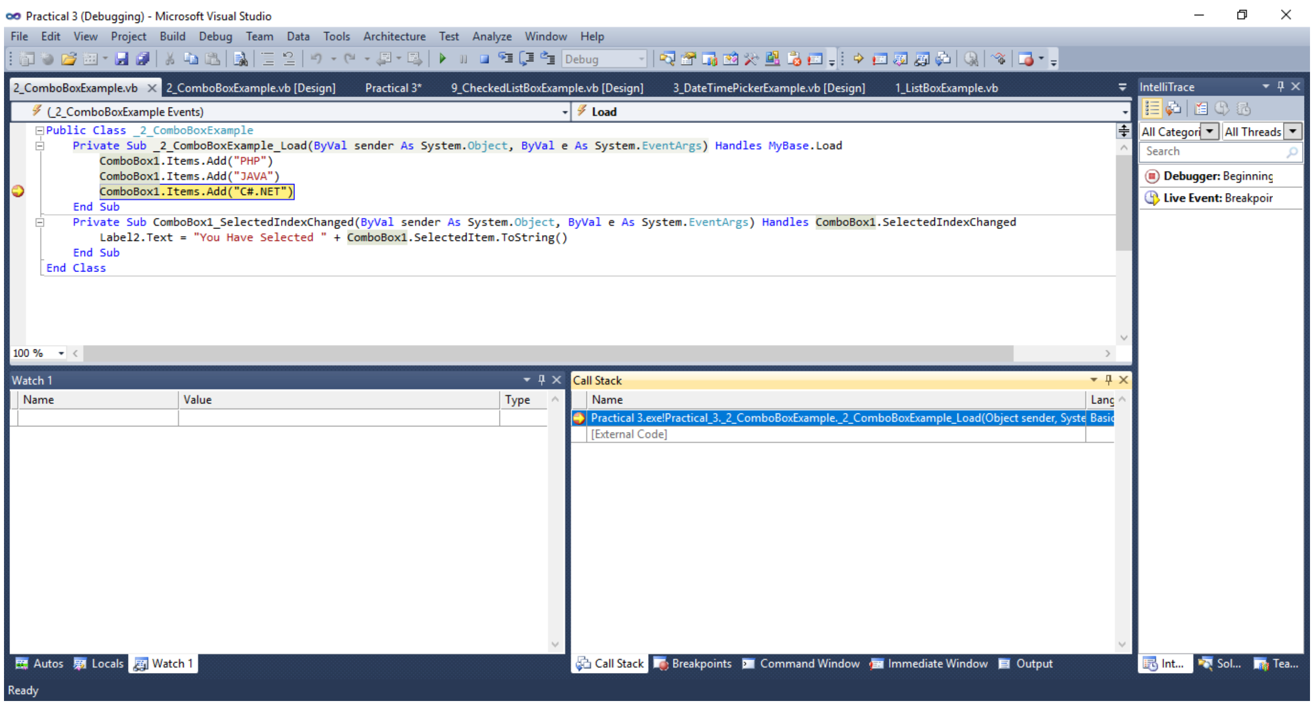
- Open visual studio then **start any existing application**.

- In the window check the **left margin on the line where you want to use breakpoint** as shown in below figure.



```
Public Class _2_ComboBoxExample
    Private Sub _2_ComboBoxExample_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Lo
        ComboBox1.Items.Add("PHP")
        ComboBox1.Items.Add("JAVA")
        ComboBox1.Items.Add("C#.NET")
    End Sub
    Private Sub ComboBox1_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Co
        Label2.Text = "You Have Selected " + ComboBox1.SelectedItem.ToString()
    End Sub
End Class
```

- Double click in the left margin, where you want to put debugging point.
- A red dot appears and the texts on that **line are highlighted by red color**.
- Now **run the application**, the **debugger will break the application execution** at the location where the code is hit or **break point is found**.
- In this program, we have used break point in the Form Load event. When the application is run, you can see the **yellow highlighted at the line** where we put the break point.
- The code is stop to execute that line, and wait for the response.
- We can also view the state of the application and debug it.

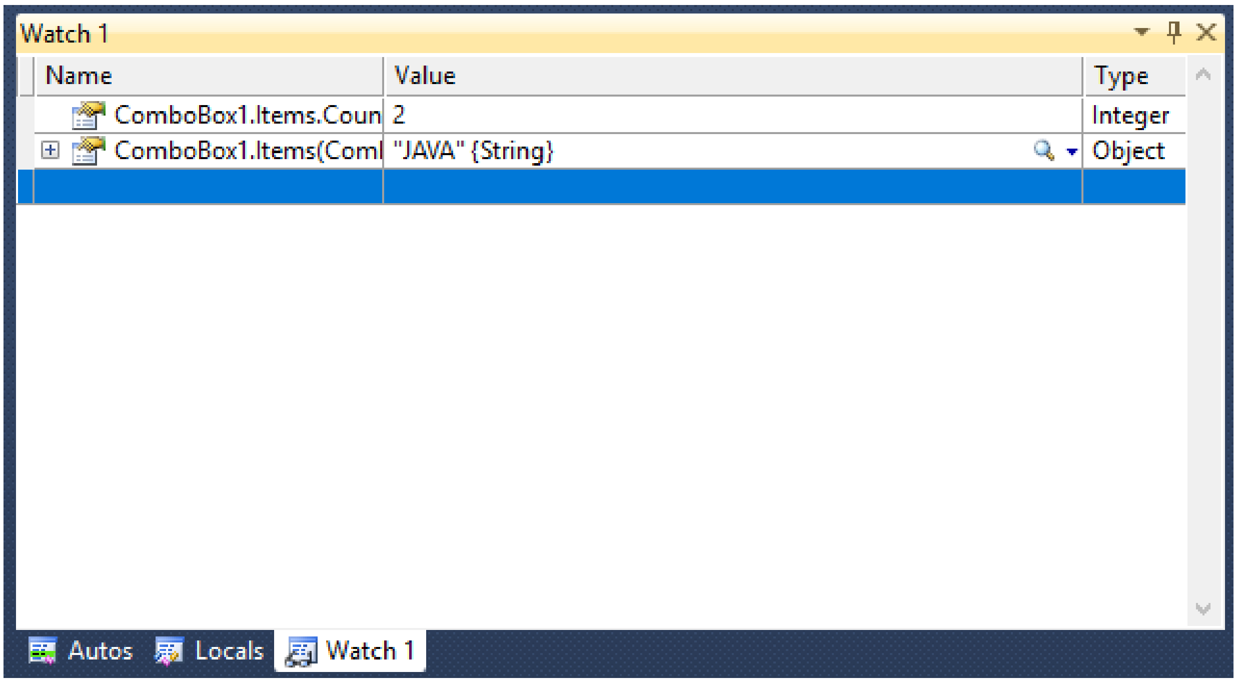


```
Public Class _2_ComboBoxExample
    Private Sub _2_ComboBoxExample_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        ComboBox1.Items.Add("PHP")
        ComboBox1.Items.Add("JAVA")
        ComboBox1.Items.Add("C#.NET")
    End Sub
    Private Sub ComboBox1_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ComboBox1.SelectedIndexChanged
        Label2.Text = "You Have Selected " + ComboBox1.SelectedItem.ToString()
    End Sub
End Class
```

Call Stack: Practical 3.exe!Practical_3._2_ComboBoxExample._2_ComboBoxExample_Load(Object sender, Syste Basic
[External Code]

- To view the **state of application**, go to **Debug Menu -> Windows -> Watch -> Watch 1.** The watch window is opened at the bottom of application. Watch window only available when you are in debugging mode.

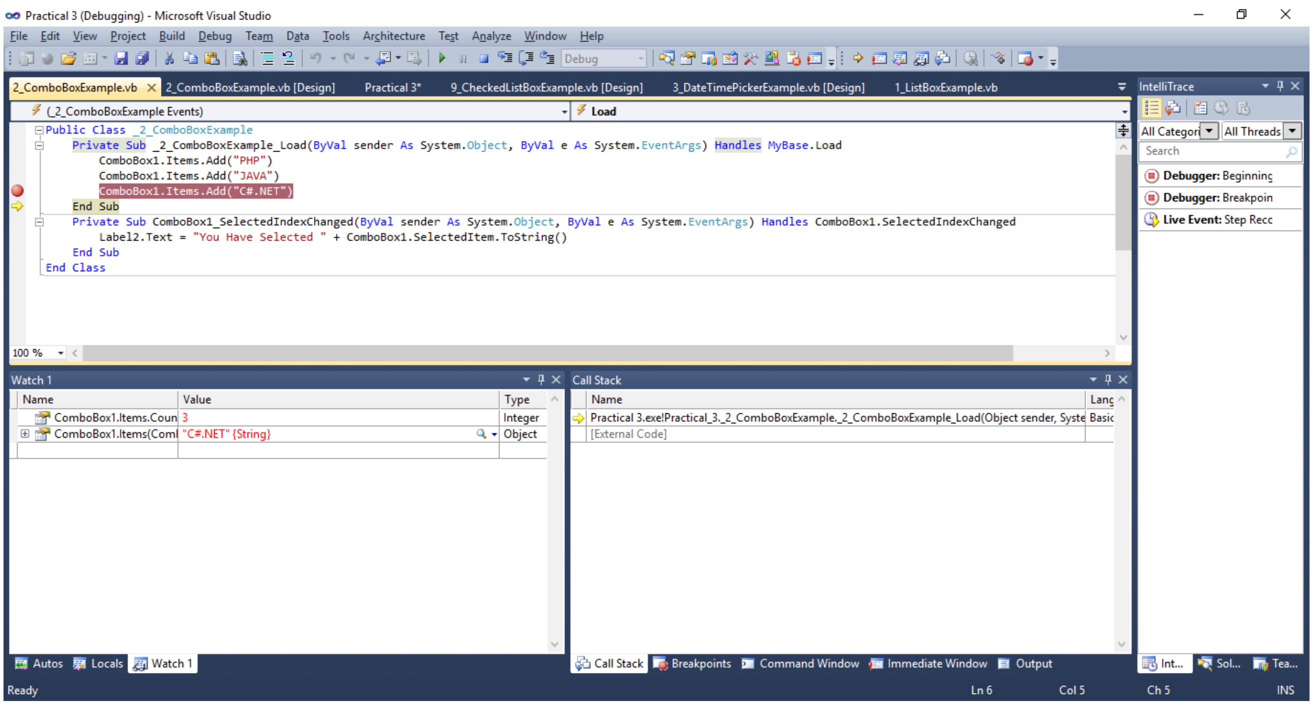
- In the watch window no details will be displayed. To **view details click on Blank Row**. In the name column type **ComboBox1.Items.Count**, then press Enter.
- In the next line of watch window, type **ComboBox1.Items(ComboBox1.Items-1)**, then press Enter.
- The Watch window shows the value of this expression as shown in below figure.



Watch 1

Name	Value	Type
ComboBox1.Items.Coun	2	Integer
ComboBox1.Items(Coml	"JAVA" {String}	Object

- From the debug menu, **choose step into**. The value of expression in watch window updated.



Practical 3 (Debugging) - Microsoft Visual Studio

```
Public Class _2_ComboBoxExample
    Private Sub _2_ComboBoxExample_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        ComboBox1.Items.Add("PHP")
        ComboBox1.Items.Add("JAVA")
        ComboBox1.Items.Add("C#.NET")
    End Sub
    Private Sub ComboBox1_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ComboBox1.SelectedIndexChanged
        Label2.Text = "You Have Selected " + ComboBox1.SelectedItem.ToString()
    End Sub
End Class
```

Watch 1

Name	Value	Type
ComboBox1.Items.Coun	3	Integer
ComboBox1.Items(Coml	"C#.NET" {String}	Object

Call Stack

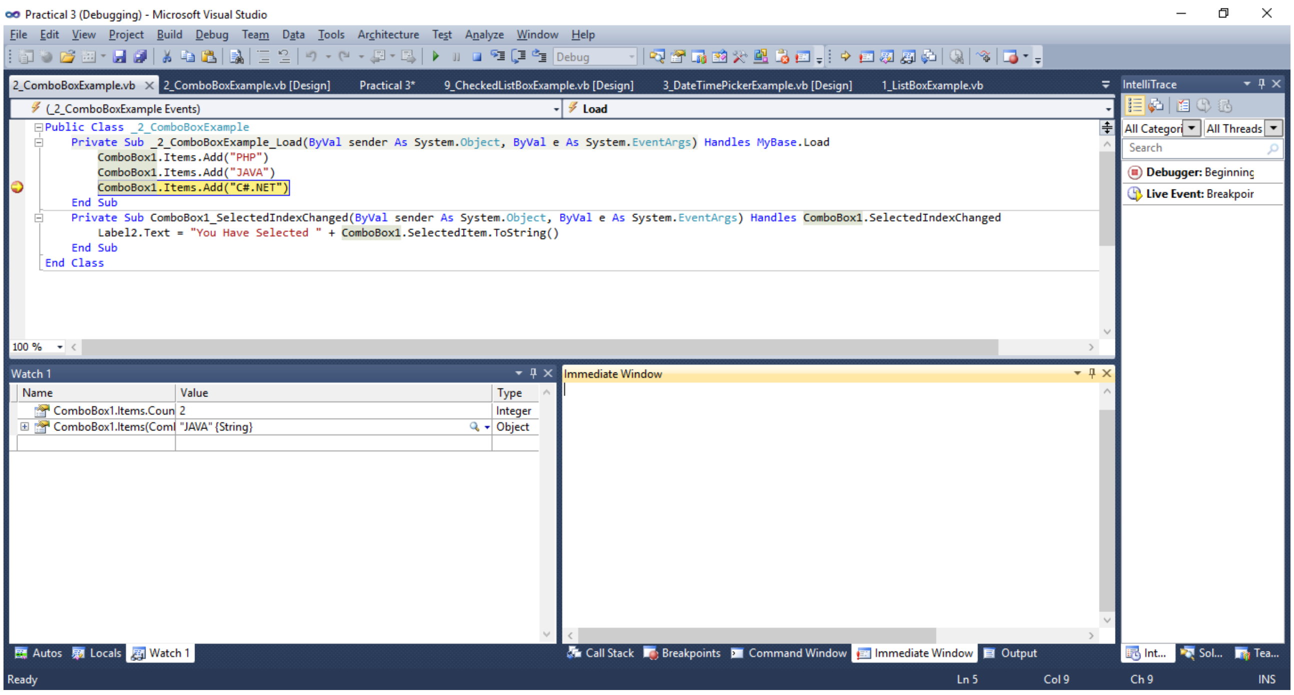
Name	Lang
Practical 3.exe!Practical_3._2_ComboBoxExample._2_ComboBoxExample_Load(Object sender, Syste	Basic
[External Code]	

- From the debug menu, **choose Continue to resume debugging the program**.
- To remove the **Break Point, just click on the break point**, then it is removed from the code.

Steps to debug vb.net application with Immediate:

- In some situation when we want to examine the code, Visual Studio provides Immediate Window for such kind of code. Below are the steps for code examining using Immediate Window.

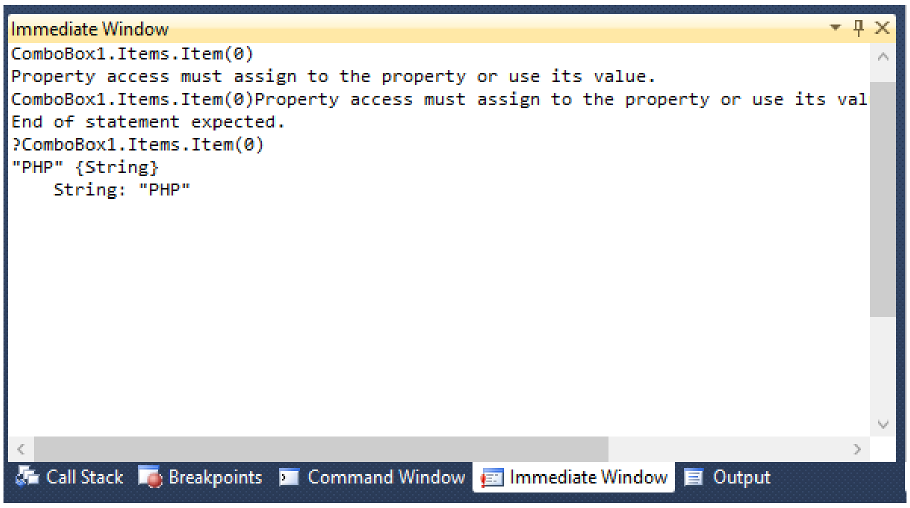
1. Put Break Point. Then run application.
2. From Debug menu select, **Debug Menu -> Windows -> Immediate**. It will displayed in the bottom of the IDE.


Practical 3 (Debugging) - Microsoft Visual Studio
File Edit View Project Build Debug Team Data Tools Architecture Test Analyze Window Help
Debug
2_ComboBoxExample.vb
2_ComboBoxExample.vb [Design]
Practical 3*
9_CheckedListBoxExample.vb [Design]
3_DateTimePickerExample.vb [Design]
1_ListBoxExample.vb
(_2_ComboBoxExample Events)
Load
IntelliTrace
All Categori
All Threads
Search
Debugger: Beginning
Live Event: Breakpoir
100 %
Watch 1
Immediate Window
Autos Locals Watch 1
Call Stack Breakpoints Command Window Immediate Window Output
Int... Sol... Tea...
Ready
Ln 5
Col 9
Ch 9
INS

```
Public Class _2_ComboBoxExample
    Private Sub _2_ComboBoxExample_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        ComboBox1.Items.Add("PHP")
        ComboBox1.Items.Add("JAVA")
        ComboBox1.Items.Add("C#.NET")
    End Sub
    Private Sub ComboBox1_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ComboBox1.SelectedIndexChanged
        Label2.Text = "You Have Selected " + ComboBox1.SelectedItem.ToString()
    End Sub
End Class
```

Name	Value	Type
ComboBox1.Items.Coun	2	Integer
ComboBox1.Items(Coml	"JAVA" {String}	Object

3. **Type ?ComboBox1.Items.Item(0)** in the Immediate Window and press Enter. ? operator will put before any valid VB.net Expression in Immediate window and you can see the result as shown in below figure.



```
ComboBox1.Items.Item(0)
Property access must assign to the property or use its value.
ComboBox1.Items.Item(0)Property access must assign to the property or use its val
End of statement expected.
?ComboBox1.Items.Item(0)
"PHP" {String}
    String: "PHP"
```

6. GTU QUESTIONS:

Sr. No	Questions	Marks
	18/02/2021	
1	Explain “On Error GoTo line” statement.	2
2	List the Dialog Boxes.	2
3	Write a program that shows Call By Value and Call By Reference.	4
4	Differentiate between MDI and SDI.	3
5	Explain procedure and Function in detail.	3
6	Describe about Structured error handling.	3
	29/10/2020	
1	List the Dialog Boxes.	2
2	Define On error GOTO -1.	2
3	Write Syntax of Sub procedure.	2
4	Define DialogBox control and explain Colordialog control.	4
5	Write a program to explain Call by reference.	4
6	Write a program to explain Call by value.	4
7	Describe about Structured error handling.	4
8	Write a program to multiply two numbers using Procedure.	3
9	Describe MDI with example.	4
	16/11/2019	
1	List types of Error and explain Structured Error with suitable example.	3
2	Explain procedure and Function in detail.	3
3	List the Dialog Boxes and Explain Save File dialog Box in brief	3
	17/05/2019	
1	Explain “On Error GoTo line” statement.	2
2	Explain Procedure and Function in detail.	3
3	Write a program using Call by value and Call by reference.	3
4	What is Exception handling? Explain structured exception handling in detail.	4
5	List different Dialog boxes. Explain any one in detail.	3
6	Describe color dialog box.	4
	28/11/2018	
1	Differentiate Sub routine and Function in .NET	2
2	Explain Sub procedure and Function	3
3	Explain structured exception handling.	4
4	Explain unstructured exception handling.	4
	04/05/2018	
1	Differentiate between pass by value and pass by reference.	2
2	Write a program which is showing Call By Value and Call By Reference.	4
3	What is Exception Handling? Explain with example.	3
4	Explain any one Common Dialog Control with example.	3
	03/05/2017	
1	Explain any two types of Error.	2
2	Write Syntax of Sub Routine. Suppose your form contains four TextBoxes. Write subroutine to clear all textboxes and set the focus on the first TextBox.	4
3	Write Syntax of Function. Create one function that takes one integer argument, finds its factorial and returns it.	4

4	Explain Structured Error Handling.	4
5	When do we use MDI?	3
6	Describe the main features of OpenFileDialog Control.	3
7	Describe the main features of FontDialog Control.	3
	24/11/2016	
1	Explain call by value and call by reference.	4
2	Explain Picture box.	2
3	What is exception handling? List out the types. and explain anyone with suitable example.	4
4	Explain MDI with suitable example.	4
5	Explain Function and procedure with appropriate Example.	4
6	Give the list of windows common dialog box and explain open file dialog box with suitable example	4
	20/05/2016	
1	Differentiate between pass by value and pass by reference.	2
2	What are differences between sub procedure and function?	2
3	What is a MDI form? Write merits of using it in an application.	4
4	Discuss structured error handling.	3
5	Describe Color Dialog box.	2
6	Describe Font Dialog box.	4
	07/12/2015	
1	Explain MDI with example.	4
2	Write a program to explain Call By Value and Call By Reference.	3
3	Explain procedure and Function in detail.	3
4	What is Exception Handling? Explain Unstructured exception handling in detail with suitable example.	4
5	Explain ColorDialogBox with it's properties and methods in brief	2
6	List down various Common Dialog Controls .Explain any one in detail with suitable example.	4
	12/05/2015	
	Define Method or Function	2
	Differentiate between procedure and function.	2
	What is Exception Handling? Explain structured error Handling in detail.	4
	What is call by value and call by reference	3
	Explain in detail Multiple Document Interface.	3
	04/12/2014	
1	Explain "Passing Variable Number of Arguments to Procedure" using example.	7
2	Explain any two of following with example i) Err Object ii) Try…Catch…Finally statement iii) Function Procedure	7
3	Explain "Passing Named Argument to Procedure" using example.	7
4	Explain any two of following with example i) Try…Catch Statement ii) On Error Statement iii) Sub Procedure	7