Training Manual

FOR

Microsoft Visual Basic Programming

Compiled by **Technology Ed**

April 2012

CHAPTER-3: CODING

This chapter describes in detail the coding part of Visual Basic.

3.1 Examining Code

A logic error is called a bug. Debugging is the process of examining code to look for bugs or to identify problems. Debugging is the ability to monitor the behavior of a variable, a function, or another item throughout a program. Microsoft Visual Basic provides many features to perform debugging operations.

The debugger is the program you use to debug your code. The code or application that you are debugging is called the debuggee.

Probably the most fundamental way of examining code is to read every word and every line, with your eyes, using your experience as a programmer. This can work for short code written in one file and in one class. If the code to examine covers many pages or many files, it could be awful and tiresome to examine code with your eyes line by line. Fortunately, to assist you with this operation, Microsoft Visual Studio provides various tools and windows that you use one window or a combination of objects. One of the tools you can use is the Standard toolbar that is equipped with various debugging buttons.

3.2 Using the Object Browser

The Object Browser allows us to browse through the various properties, events and methods that are made available to us. It is accessed by selecting Object Browser from the View menu or pressing the F2 key. The left column of the Object Browser lists the objects and classes that are available in the projects that are opened and the controls that have been referenced in them. It is possible for us to scroll through the list and select the object or class that we wish to inspect. After an object is picked up from the Classes list, we can see its members (properties, methods and events) in the right column.

A property is represented by a small icon that has a hand holding a piece of paper. Methods are denoted by little green blocks, while events are denoted by yellow lightning bolt icon.

Object naming conversions of controls (prefix):

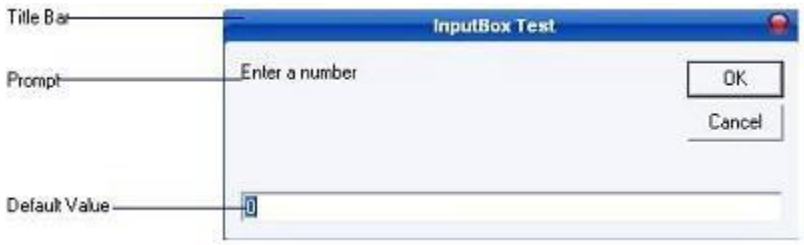
- Form -frm
- Label-lbl

- TextBox-txt
- CommandButton-cmd
- CheckBox -chk
- OptionButton -opt
- ComboBox -cbo
- ListBox-lst
- Frame-fme
- PictureBox -pic
- Image-img
- Shape-shp
- Line -lin
- HScrollBar -hsb
- VScrollBar -vsb

3.3 Statements and Functions

Displays a prompt in a dialog box, waits for the user to input text or click a button, and returns a String containing the contents of the text box.

The following is an expanded InputBox and the syntax.




```
memory_variable = InputBox (prompt[,title][,default])
```

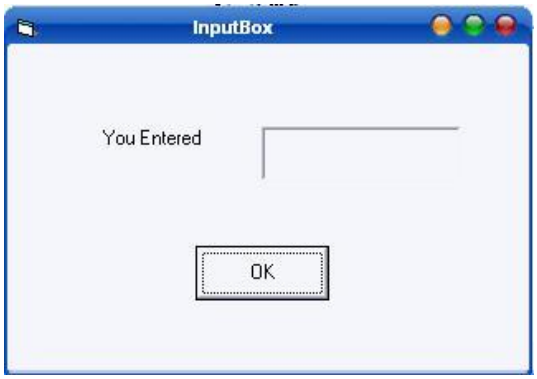
Where memory_variable is a variant data type but typically it is declared as string, which accepts the message input by the users. The arguments are explained as follows:

- **Prompt** - String expression displayed as the message in the dialog box. If prompt consists of more than one line, you can separate the lines using the vbCrLf constant

- **Title** - String expression displayed in the title bar of the dialog box. If you omit the title, the application name is displayed in the title bar
- **default-text** - The default text that appears in the input field where users can use it as his intended input or he may change to the message he wish to key in
- **x-position and y-position** - the position or the coordinate of the input box

Following example demonstrates the use of InputBox function: Open a new project and save the Form as InputBox.frm and save the Project as InputBox.vbp. Design the application as shown below.

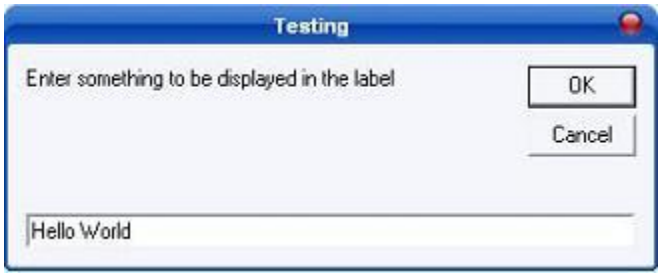
Object	Property	Setting
Form	Caption Name	InputBox test frmInputBox
Label	Caption Name	You entered lbl1
Label	Caption Name BorderStyle	(empty) lbl2 1-Fixed Single
CommandButton	Caption Name	OK cmdOK



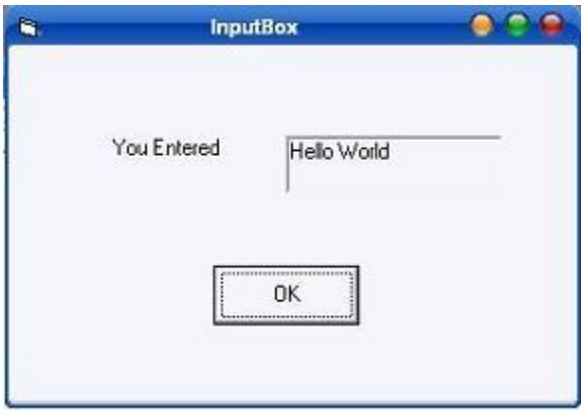
Following code is entered in cmdOK_Click ( ) event

```
Private Sub cmdok Click()
Dim ans As String
ans = InputBox("Enter something to be displayed in the label",
"Testing", 0)
If ans = "" Then
lbl2.Caption = "No message"
Else
lbl2.Caption = ans
End If
End Sub
```

Save and run the application. As soon as you click the OK button you will get the following InputBox:



Here I have entered "Hello World" in text field. As soon as you click OK the output is shown as shown below:



Displays a message in a dialog box and wait for the user to click a button, and returns an integer indicating which button the user clicked.

The following is an expanded MessageBox and the respective syntax:


```
MsgBox ( Prompt [,icons+buttons ] [,title ] )

memory_variable = MsgBox ( prompt [, icons+ buttons] [,title] )
```

Prompt: String expressions displayed as the message in the dialog box. If prompt consist of more than one line, you can separate the lines using the vbrCrLf constant.

Icons + Buttons: Numeric expression that is the sum of values specifying the number and type of buttons and icon to display.

Title: String expression displayed in the title bar of the dialog box. If you omit title, the application name is placed in the title bar.

Icons

Constant	Value	Description
vbCritical	16	Display Critical message icon
vbQuestion	32	Display Warning Query icon
vbExclamation	48	Display Warning message icon
vbInformation	64	Display information icon

Buttons

Constant	Value	Description
vbOkOnly	0	Display OK button only
vbOkCancel	1	Display OK and Cancel buttons
vbAbortRetryIgnore	2	Display Abort, Retry and Ignore buttons
vbYesNoCancel	3	Display Yes, No and Cancel buttons
vbYesNo	4	Display Yes and No buttons
vbRetryCancel	5	Display Retry and Cancel buttons

Return Values

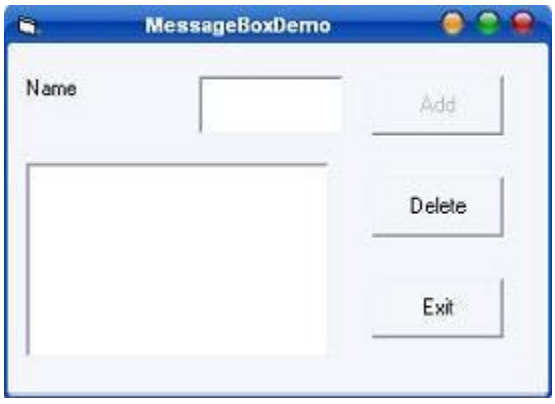
Constant	Value	Description
vbOk	1	Ok Button
vbCancel	2	Cancel Button
vbAbort	3	Abort Button
vbRetry	4	Retry Button
vbIgnore	5	Ignore Button

vbYes	6	Yes Button
vbNo	7	No Button

The following example illustrates the use of Message Boxes in Visual Basic:0

Open a new Project and save the Form as messageboxdemo.frm and save the Project as messageboxdemo.vbp. Design the application as shown below.

Object	Property	Setting
Form	Caption Name	MessageBoxDemo frmMessageBoxDemo
Label	Caption Name	lblName Name
TextBox	Name Text	txtName (empty)
ListBox	Name	lstName
CommandButton	Caption Name	Add cmdAdd
CommandButton	Caption Name	Delete cmdDelete
CommandButton	Caption Name	Exit cmdExit



Following code is entered in the txtName_Change ( ) event

```
Private Sub txtName_Change()
If Len(txtName.Text) > 0 Then
cmdAdd.Enabled = True
End If
End Sub
```

Following code has to be entered in the cmdAdd_Click ( ) event

```
Private Sub cmdAdd_Click()
answer = MsgBox("Do you want to add this name to the list box?",
vbExclamation + vbYesNo,
"Add Confirm")
If answer = vbYes Then
lstName.AddItem txtName.Text
txtName.Text = ""
txtName.SetFocus
cmdAdd.Enabled = False
End If
End Sub
```

Following code is entered in the cmdDelete_Click ( ) event

```
Private Sub cmdDelete_Click()
Dim remove As Integer
remove = lstName.ListIndex
If remove < 0 Then
MsgBox "No names is selected", vbInformation, "Error"
Else
answer = MsgBox("Are you sure you want to delete " & vbCrLf & "the
selected name?",_
vbCritical + vbYesNo, "Warning")
If answer = vbYes Then
If remove >= 0 Then
lstName.RemoveItem remove
txtName.SetFocus
MsgBox "Selected name was deleted", vbInformation, "Delete Confirm"

End If
End If
End If
End Sub
```

Following code is entered in the cmdExit_Click ( ) event

```
Private Sub cmdExit_Click()
answer = MsgBox("Do you want to quit?", vbExclamation + vbYesNo,
"Confirm")
If answer = vbYes Then
End
```

```
Else
MsgBox "Action canceled", vbInformation, "Confirm"
End If
End Sub
```

Save and run the application. You can notice the different type of message box types are used to perform an action

Not only does Visual Basic let you store date and time information in the specific Date data type, it also provides a lot of date- and time-related functions. These functions are very important in all business applications and deserve an in-depth look. Date and Time are internally stored as numbers in Visual Basic. The decimal points represents the time between 0:00:00 and 23:59:59 hours inclusive.

The system's current date and time can be retrieved using the Now, Date and Time functions in Visual Basic. The Now function retrieves the date and time, while Date function retrieves only date and Time function retrieves only the time.

To display both the date and time together a message box is displayed use the statement given below.

```
MsgBox "The current date and time of the system is" & Now
```

In this case “&” is used as a concatenation operator to concentrate the string and the Now function. Selective portions of the date and time value can be extracted using the below listed functions.

Function	Extracted Portion
Year ( )	Year (Now)
Month ( )	Month (Now)
Day ( )	Day (Now)
WeekDay ( )	WeekDay (Now)
Hour ( )	Hour (Now)

Minute ( )	Minute (Now)
Second ( )	Second (Now)

The calculation and conversion functions related to date and time functions are listed below.

Function	Description
DateAdd ( )	Returns a date to which a specific interval has been added
DateDiff ( )	Returns a Long data type value specifying the interval between the two values
DatePart ( )	Returns an Integer containing the specified part of a given date
DateValue ( )	Converts a string to a Date
TimeValue ( )	Converts a string to a time
DateSerial ( )	Returns a date for specified year, month and day

3.3.1 DateDiff Function

The DateDiff function returns the intervals between two dates in terms of years, months or days. The syntax for this is given below.

```
DateDiff (interval, date1, date2[, firstdayofweek[, firstweekofyear]])
```

3.3.2 Format Function

The format function accepts a numeric value and converts it to a string in the format specified by the format argument. The syntax for this is given below.

```
Format (expression[, format[, firstdayofweek[, firstweekofyear]]])
```

The Format function syntax has these parts:

Part	Description
Expression	Required any valid expression
format	Optional. A valid named or user-defined format expression.
firstdayofweek	Optional. A contant that specifies the first day of the week.
firstweekofyear	Optional. A contant that specifies the first week of the year

3.4 Conditional Statements in Visual Basic

Control Statements are used to control the flow of program's execution. Visual Basic supports control structures such as if... Then, if...Then ...Else, Select...Case, and Loop structures such as Do While...Loop, While...Wend, For...Next etc method.

3.4.1 If…Then selection structure

The If...Then selection structure performs an indicated action only when the condition is True; otherwise the action is skipped.

Syntax of the If...Then selection

```
If <condition> Then
statement
End If

e.g.: If average>75 Then
txtGrade.Text = "A"
End If
```

3.4.2 If…Then…Else selection structure

The If...Then...Else selection structure allows the programmer to specify that a different action is to be performed when the condition is True than when the condition is False.

Syntax of the If...Then...Else selection

```
If <condition > Then
statements
Else
statements
End If

e.g.: If average>50 Then
txtGrade.Text = "Pass"
Else
txtGrade.Text = "Fail"
End If
```

3.4.3 Nested If…Then…Else selection structure

Nested If...Then...Else selection structures test for multiple cases by placing If...Then...Else selection structures insideIf...Then...Else structures.

Syntax of the Nested If...Then...Else selection structure

You can use Nested If either of the methods as shown above

Method 1

```
If < condition 1 > Then
statements
ElseIf < condition 2 > Then
statements
ElseIf < condition 3 > Then
statements
Else
Statements
End If
```

Method 2

```
If < condition 1 > Then
statements
Else
If < condition 2 > Then
statements
Else
If < condition 3 > Then
statements
Else
Statements
End If
End If
EndIf
```

For example, assume you have to find the grade using nested if and display in a text box:

```
If average > 75 Then
txtGrade.Text = "A"
ElseIf average > 65 Then
txtGrade.Text = "B"
ElseIf average > 55 Then
txtGrade.text = "C"
ElseIf average > 45 Then
txtGrade.Text = "S"
Else
txtGrade.Text = "F"
End If
```

3.4.4 Select…Case selection structure

Select...Case structure is an alternative to If...Then...ElseIf for selectively executing a single block of statements from among multiple block of statements. Select...case is more convenient to use than the If...Else...End If. The following program block illustrate the working of Select...Case.

Syntax of the Select...Case selection structure

```
Select Case Index
Case 0
Statements
Case 1
Statements
End Select
```

For example, assume you have to find the grade using select...case and display in the text box:

```
Dim average as Integer

average = txtAverage.Text
Select Case average
Case 100 To 75
txtGrade.Text ="A"
Case 74 To 65
txtGrade.Text ="B"
Case 64 To 55
txtGrade.Text ="C"
Case 54 To 45
txtGrade.Text ="S"
Case 44 To 0
txtGrade.Text ="F"
Case Else
MsgBox "Invalid average marks"
End Select
```

3.5 Looping Statements

A repetition structure allows the programmer to that an action is to be repeated until given condition is true.

3.5.1 Do While... Loop Statement

The Do While...Loop is used to execute statements until a certain condition is met. The following Do Loop counts from 1 to 100.

```
Dim number As Integer

number = 1

Do While number <= 100

number = number + 1

Loop
```

A variable number is initialized to 1 and then the Do While Loop starts. First, the condition is tested; if condition is True, then the statements are executed. When it gets to the Loop it goes back to the Do and tests condition again. If condition is False on the first pass, the statements are never executed.

3.5.2 While... Wend Statement

A While...Wend statement behaves like the Do While...Loop statement. The following While...Wend counts from 1 to 100

```
Dim number As Integer

number = 1

While number <=100

number = number + 1

Wend
```

3.5.3 Do...Loop While Statement

The Do...Loop While statement first executes the statements and then tests the condition after each execution. The following program block illustrates the structure:

```
Dim number As Long
```

```
number = 0

Do

number = number + 1

Loop While number < 201
```

The programs execute the statements between Do and Loop While structure in any case. Then it determines whether the counter is less than 501. If so, the program again executes the statements between Do and Loop While else exits the Loop.

3.5.4 Do Until…Loop Statement

Unlike the Do While...Loop and While...Wend repetition structures, the Do Until… Loop structure tests a condition for falsity. Statements in the body of a Do Until...Loop are executed repeatedly as long as the loop-continuation test evaluates to False.

An example for Do Until...Loop statement. The coding is typed inside the click event of the command button:

```
Dim number As Long

number=0

Do Until number > 1000

number = number + 1

Print number

Loop
```

Numbers between 1 to 1000 will be displayed on the form as soon as you click on the command button.

3.5.5 The For…Next Loop

The For...Next Loop is another way to make loops in Visual Basic. For...Next repetition structure handles all the details of counter-controlled repetition. The following loop counts the numbers from 1 to 100:

```
Dim x As Integer

For x = 1 To 50
```

```
Print x

Next
```

In order to count the numbers from 1 to 50 in steps of 2, the following loop can be used

```
For x = 1 To 50 Step 2

Print x

Next
```

The following loop counts numbers as 1, 3, 5, 7..etc

The above coding will display numbers vertically on the form. In order to display numbers horizontally the following method can be used.

```
For x = 1 To 50

Print x & Space$ (2);

Next
```

To increase the space between the numbers increase the value inside the brackets after the & Space$.

Following example is a For...Next repetition structure which is with the If condition used.

```
Dim number As Integer

For number = 1 To 10

If number = 4 Then

Print "This is number 4"

Else

Print number

End If

Next
```

In the output instead of number 4 you will get the "This is number 4"..

3.6 The Native Code Compiler

Visual Basic includes a native-language compiler. This compiler can translate the Visual Basic source code that programmers can read and understand into executable programs consisting of native machine code that computers can read and understand.

Versions of Visual Basic before version 5 also produce executable files, but the code in those executables does not consist of native machine code. Until versions 5 and 6, executable program files produced by Visual Basic consisted of something generally known as pseudo code, or P-Code, rather than native machine code. Native code provides a performance advantage over P-Code that VB programmers have wanted for a long time.

As if it weren't enough to include a native code compiler, Microsoft also gave VB an optimizing compiler. In addition to the performance advantage inherent in native code compilation, the optimization switches enable discerning VB programmers to tweak their applications for even greater performance.

However, Microsoft still allows the programmer to choose P-Code compilation for any project.

Although the primary purpose of source code is to provide clear instructions for the CPU, it is occasionally useful to be able to talk to the compiler too. When necessary, this permits you to give the compiler special instructions about how it should produce the code that finally gets sent to the CPU.

Visual Basic enables the programmer to talk directly to the compiler itself by embedding instructions directed to the compiler in VB source code.

This chapter discusses the consequences of choosing different compiler options in a VB project and also discusses how you can manipulate the compiler directly by using compiler directives.

Specifically, this chapter covers the following topics:

- P-Code versus native code
- When and how to optimize
- Using compile on demand
- Conditional compilation defined
- Preprocessor directives, and constants
- Applications and styles