2016 OR 2017 OR 2018(if no underline then it is either of the year)

2 years 3 years

# VB important questions.

1. **What is Intellisense.? List any two Intellisense options.**
   One useful feature of VB .NET code designers is Microsoft's IntelliSense. IntelliSense is what's responsible for those boxes that open as you write your code, listing all the possible options and even completing your typing for you. IntelliSense is one of the first things you encounter when you use VB .NET. Two options : Parameter Info and Quick Info

2. **Mention the purpose of any two option statements in VB.Net**
   Option Explicit— Set to On or Off. On is the default. Requires declaration of all variables before they are used (this is the default).
   Option Compare— Set to Binary or Text. This specifies if strings are compared using binary or text comparison operations.

3. **How to write comments in VB ? Give example.**
   Comments in Visual Basic start with an apostrophe (') and make Visual Basic ignore whatever follows the apostrophe on the line.

```
Module Module1
    Sub Main()
        'Declare the variables we will use
        Dim intGrade1, intGrade2, intGrade3 As Integer
        intGrade1 = 60
        intGrade2 = 70
        intGrade3 = 80
        Str(intGrade1 + intGrade2 + intGrade3)
    End Sub
End Module
```

4. **Differentiate Checkbox and Radio Button.**
   Unlike checkboxes, radio buttons form exclusive groups, where only one radio button can be selected (that is, display a dot in its center) at a time. But multiple checkboxes can be selected.

5. **Write the purpose of ReadOnly and MaxLength Properties of Textbox.**
   ReadOnly - Gets/sets whether the text box is read-only.
   MaxLength Gets/sets the maximum number of characters that may be displayed in the text box.

6. **List the uses of labels.**
   You can use the Label class just as you use labels in Windows forms—to display text. You can change that text with the Text property in code if you want, but the user can't edit the text in a label control directly.

7. **Write the method to**
   a. **Add an element to a listBox.**
      To add or delete items in a ListBox control, you can use the Items.Add
      Module Module1

```
    Sub Main()
        Dim intGrade1 As Integer
        intGrade1 = 60
        listbox1.item.add(intGrade1)
    End Sub
End Module
```

**b. Display the total number of elements in a listbox.**

The Items.Count property holds the number of items in the list.

**8. What is Context Menu ? Why it is used ?**

Another popular type of menus is context menus. You use Context Menu controls to give users access to frequently used menu commands, and bring them up by right clicking another control. You usually use context menus to display control-specific options, such as Cut, Copy, and Paste in text boxes.

**9. What is the use of Image List ? List any two controls that have ImageList property.**

You use image lists to store images; they form a kind of image repository. This doesn't sound terrible useful, but in fact, there are plenty of controls that are designed to work with image lists: list views, tree views, toolbars, tab controls, checkboxes, buttons, radio buttons, and labels—all of which have an ImageList (or SmallImageList and LargeImageList) and ImageIndex property.

**10. Write the purpose of Data adapter.**

You need a data adapter because datasets do not maintain any active connection to the database—they are disconnected from the database. The data adapter is what actually applies your SQL statements to a database and causes your datasets to fill with data.

**11. What is a Data Grid ?**

The DataView provides different views of the data stored in a DataTable. DataViews can be created and configured at both design time and run time . We can create DataView in two ways. Either we can use the DataView constructor, or we can create a reference to the DefaultView property of the DataTable. We can create multiple DataViews for any given DataTable.

**12. What is the use of Server Explorer ?**

the Server Explorer is a great tool for working with data connections. To display the Server Explorer if it's not already visible, use the View|Server Explorer menu item, or press Ctrl+Alt+S. You can explore a database with the Server Explorer.

**13. What is Class Library with reference to .NET Framework. ?**

The code for all elements we use in a VB .NET application-forms, buttons, menus, and all the rest-all comes from the class library. And other Visual Studio applications use the same class library, making it easy to mix languages in your programming, even in the same application.

**14. What is the use of System.date and System.Windows.Forms namespaces.**

DateAdd, DateDiff, DatePart, DateSerial, DateValue, TimeSerial, TimeValue have been replaced by elements in System.DateTime.

You can't build a VB .NET application without using classes from the .NET System namespace. When you want to use a Windows form, for example, you must use the System.Windows.Forms

**15. What is the use of Str and Val functions.give example for each.**

You can use the Str to return a string representation of a number, and you use Val to convert a string to a number.

**16. Write the purpose of IsNumeric() and IsError() functions**

IsNumeric() Returns True if passed an numeric value.
IsError() Returns True if passed an error value

**17. Mention the uses of Rich Text Box.**

You can enter formatted text (selecting fonts, italics, bolding, and more) in a rich text box, you also can save that text in rich text format (RTF) files, and read RTF files in it.

**18. Differentiate Panel control and Group Box Control.**

The Panel control is similar to the GroupBox control; however, only the Panel control can have scroll bars, and only the GroupBox control displays a caption.

**19. What is the use of Finally Block .**

The code in the Finally block, if there is one, is always executed in a Try…Catch…Finally statement, even if there was no exception, and even if you execute an Exit Try statement. This allows you to deallocate resources

**20. What do you mean my block scope? Give an example.**

A block is a series of statements terminated by an End, Else, Loop, or Next statement, and an element declared within a block can be used only within that block.

```
Module Module1
    Sub Main()
        Dim intValue As Integer = 1
        If intValue = 1 Then
            Dim strText As String = "No worries."
            System.Console.WriteLine(strText)
        End If
        System.Console.WriteLine(strText) 'Will not work!
    End Sub
End Module
```

**21. Write the use of Execute Scalar and ExecuteNonQuery methods.**

ExecuteScalar Executes the command and returns the value in the firstcolumn in the first row of the result**.**
ExecuteNonQuery Executes a non-row returning SQL statement, returning the number of affected rows.

**22. Name any two data providers.**

SQL Server, MS Jet.

**23. List and two system namespaces.**

System.Console
System.Object

**24. How do you create a text box in code.**

```
Private Sub Button1_Click(ByVal sender As System.Object, _ByVal e As System.EventArgs) Handles Button1.Click
    Dim TextBox1 As New TextBox()
```

```
        TextBox1.Size = New Size(150, 20)
        TextBox1.Location = New Point(80, 20)
        TextBox1.Text = "Hello from Visual Basic"
        Me.Controls.Add(TextBox1)
    End Sub
```

**25. Differentiate Textboxes and labels.**

A label is meant to be used beside a text box to make a user understand what is to be entered in that text box where as a text box is used normally for user input. The contents of a label is not to be directly modified by a user where as the contents of a text box is for the user to modify.

**26. Write any four keyboard events.**

KeyDown - Occurs when a key is pressed down while the form has focus.
KeyPress - Occurs when a key is pressed while the form has focus.
KeyUp - Occurs when a key is released while the form has focus.
Alt - Holds a value indicating whether the Alt key was pressed

**27. List the different views of listview.**

Large icon mode—displays large icons (large icons are 32×32 pixels) next tothe item text.
Small icon mode—is the same as the large icon mode except that it displays items using small icons (small icons are 16×16 pixels).
List mode — displays small icons, and always in one column.
Details mode — displays items in multiple columns, displaying column headers and fields.

**28. Write any four properties of scrollbar.**

LargeChange - Gets/sets the value added to or subtracted from to the Value property when the scroll bar itself is clicked (outside the scroll box).
Maximum - Gets/sets the upper limit of the scrollable range.
Minimum - Gets/sets the lower limit of the scrollable range.
SmallChange - Gets/sets the value added to or subtracted from to the Value property when the user clicks an arrow button.

**29. Differentiate list box and combo box.**

list box:
1)does not contain a text box to write an item.
2)always shows more than one item.
3)we can select one item from multiple items in the list box.
4)contain a check box with in the list box.
combo box:
1)contain a text box.
2)always shows one item.
3)selection is not available.
4)does not contain a checkbox.

**30. List any four data providers.**

SQL Server Connection
OLEDB Connection
ODBC Connection

# Unit-1

1. **Explain the following components of Visual Basic Integrated Development Environment.**
   a. **Tool Box**
      the toolbox displays the possible controls you can use in Windows forms, such as text boxes, buttons, labels, link labels, and so on.
   b. **Solution Explorer**
      Solution Explorer is a special window that enables you to manage solutions, projects, and files. It provides a complete view of the files in a project, and it enables you to add or remove files and to organize files into subfolders.
   c. **Properties Window.**
      The Properties window displays the properties of single or multiple selected items. If multiple items are selected, the intersection of all properties for all selected objects is displayed.
   d. **Code Window**
      The Code window is found to the right of the Project Explorer. It displays the VBA code for the object currently highlighted in the Project Explorer.

2. **Explain Do Loop with syntax and example.**
   The Do loop keeps executing its enclosed statements while or until (depending on which keyword you use, While or Until) condition is true. You can also terminate a Do loop at any time with an Exit Do statement. The Do loop has two versions; you can either evaluate a condition at the beginning:

```
Do [{While | Until} condition ]
    [statements]
    [Exit Do]
    [statements]
Loop
```

or at the end:

```
Do
    [statements]
    [Exit Do]
    [statements]
Loop [{While | Until} condition]
```

```
Module Module1
    Sub Main()
        Dim strInput As String
        Do Until UCase(strInput) = "STOP"
            System.Console.WriteLine("What should I do?")
            strInput = System.Console.ReadLine()
        Loop
    End Sub
End Module
```

3. **Explain structured exception handling in VB.NET with Syntax example.**

Structured exception handling is based on a particular statement, the Try…Catch…Finally statement, which is divided into a Try block, optional Catch blocks, and an optional Finally block. The Try block contains code where exceptions can occur, the Catch block contains code to handle the exceptions that occur. If an exception occurs in the Try block, the code throws the exception—actually an object based on the Visual Basic Exception class—so it can be caught and handled by the appropriate Catch statement. After the rest of the statement finishes, execution is always passed to the Finally block, if there is one. Here's what the Try…Catch…Finally statement looks like in general:

```
Try
[ tryStatements ]
[Catch [ exception1 [ As type1 ] ] [ When expression1 ]
catchStatements1
[Exit Try] ]
[Catch [ exception2 [ As type2 ] ] [When expression2 ]
catchStatements2
[ Exit Try ] ]
⋮
[Catch [ exceptionn [ As typen ] ] [ When expressionn ]
catchStatementsn ]
[ Exit Try ] ]
[ Finally
[ finallyStatements ]
End Try
```

4. **Write the purpose of the following functions in VB.NET**
   a. **ISArray()**
      Returns True if passed an array
   b. **IsDate()**
      Returns True if passed a date
   c. **IsDBNull()**
      Returns True if passed a database NULL value; that is, a System.DBNull value
   d. **IsNumeric()**
      Returns True if passed an numeric value
   e. **IsError()**
      Returns True if passed an error value

5. **Explain Switch and Choose functions with syntax and example.**

The **Switch function** evaluates a list of expressions and returns an Object value or an expression associated with the first expression in the list that is true. Here's the syntax:

```
Switch(expr-1, value-1[, expr-2, value-2 … [, expr-n, value-n]])
```

In this case, expr-1 is the first expression to evaluate; if true, Switch returns value-1. If expr-1 is not true but expr-2 is, Switch returns value-2 and so on. Here's an example showing how to use Switch. In this case, I'm using Switch to calculate the absolute value of the value in the variable intValue (having temporarily forgotten how to use the built-in Visual Basic absolute value function, Abs):

```
intAbsValue = Switch(intValue < 0, -1 * intValue, intValue >= 0, intValue)
```

You use the **Choose function** to return one of a number of choices based on an index.
Here's the syntax:

Choose(index, choice-1[, choice-2, … [, choice-n]])

If the index value is 1, the first choice is returned, if index equals 2, the second choice is returned, and so on. Here's an example using Choose. In this case, we have three employees, Bob, Denise, and Ted, with employee IDs 1, 2, and 3. This code uses an ID value to assign the corresponding employee name to strEmployName:

```
strEmployeeName = Choose(intID, "Bob", "Denise", "Ted")
```

**6. Write the syntax of select Case and give an example.**

If your program can handle multiple values of a particular variable and you don't want to stack up a lot of If Else statements to handle them, you should consider Select Case. You use Select Case to test an expression, determine which of several cases it matches, and execute the corresponding code. Here's the syntax:

```
Select Case testexpression
[Case expressionlist-n
    [statements-n]]…
[Case Else
    [elsestatements]]
End Select
```

You use multiple Case statements in a Select statement, each specifying a different value to test against textexpression, and the code in the Case statement that matches is executed.

```
Module Module1
    Sub Main()
        Dim intInput As Integer
        System.Console.WriteLine("Enter an integer…")
        intInput = Val(System.Console.ReadLine())
        Select Case intInput
            Case 1
            System.Console.WriteLine("Thank you.")
            Case 2
            System.Console.WriteLine("That's fine.")
            Case 3
            System.Console.WriteLine("OK.")
            Case 4 To 7
            System.Console.WriteLine("In the range 4 to 7.")
            Case Is > 7
            System.Console.WriteLine("Definitely too big.")
            Case Else
            System.Console.WriteLine("Not a number I know.")
        End Select
    End Sub
End Module
```

**7. Explain For Loop and For Each..Next Loop with syntax and example.**

The **For loop** is probably the most popular of all Visual Basic loops. The Do loop doesn't need aloop index, but the For loop does; a loop index counts the number of loop iterations as the loop executes. Here's the syntax for the For loop—note that you can terminate a For loop at any time with Exit For:

```
For index = start To end [Step step]
```

```
    [statements]
[Exit For]
    [statements]
Next [index]

Module Module1
    Sub Main()
        Dim intLoopIndex As Integer
        For intLoopIndex = 0 To 3
            System.Console.WriteLine("Hello from Visual Basic")
        Next intLoopIndex
    End Sub
End Module
```

You use the **For Each…Next** loop to loop over elements in an array or a Visual Basic collection. This loop is great, because it automatically loops over all the elements in the array or collection—you don't have to worry about getting the loop indices just right to make sure you get all elements, as you do with a For loop. Here's the syntax of this
loop:

```
For Each element In group
    [statements]
[Exit For]
    [statements]
Next [element]

Module Module1
    Sub Main()
        Dim intIDArray(3), intArrayItem As Integer
        intIDArray(0) = 0
        intIDArray(1) = 1
        intIDArray(2) = 2
        intIDArray(3) = 3
        For Each intArrayItem In intIDArray
            System.Console.WriteLine(intArrayItem)
        Next intArrayItem
    End Sub
End Module
```

**8. Explain the use of 'With…end With' with syntax and example.**

The With statement is not a loop, properly speaking, but it can be as useful as a loop— and in fact, many programmers actually think of it as a loop. You use the With statement to execute statements using a particular object. Here's the syntax:

```
With object
    [statements]
End With
```

Here's an example showing how to put With to work. Here, I'm use a text box, Text1, in a Windows form program, and setting its Height, Width, and Text properties in the With statement:

```
With TextBox1
    .Height = 1000
    .Width = 3000
    .Text = "Welcome to Visual Basic"
End With
```

**9. Write the purpose of On Error Goto and Resume Next Statement.**

The old error-handling mechanism in VB6 and before is now called unstructured exception handling, and it revolves around the On Error Goto statement. You use this statement to tell VB .NET where to transfer control to in case there's been an exception, as in this case, where I'm telling Visual Basic to jump to the label "Handler" if there's been an exception. You create labels in your code with the label name followed by a colon, and the exception-handling code will follow that label (note that I've added an Exit Sub statement to make sure the code in the exception handler is not executed by mistake as part of normal program execution):

```
Module Module1
    Sub Main()
            On Error Goto Handler
            ⋮
            Exit Sub
Handler:
            ⋮
    End Sub
End Module
```

Now I can execute some code that may cause an exception, as here, where the code performs a division by zero, which causes an exception. When the exception occurs, control will jump to the exception handler, where I'll display a message and then use the Resume Next statement to transfer control back to the statement immediately after the statement that caused the exception:

```
Module Module1
    Sub Main()
            Dim int1 = 0, int2 = 1, int3 As Integer
            On Error Goto Handler
            int3 = int2 / int1
            System.Console.WriteLine("The answer is {0}", int3)
Handler:
            System.Console.WriteLine("Divide by zero error")
            Resume Next
    End Sub
End Module
```

**10. <u>Explain Option and import statements in Detail. Give an example for each.</u>**

The Option and Imports Statements

Two additional statements that are very important to know about when constructing programs are the Option and Imports statements. The Option statement sets a number of options for the rest of your code, and the Imports statement imports namespaces into your code, making them more readily available.

**Option Statements**

You use Option statements to set the "ground rules" for your code, helping prevent syntax and logic errors. Here are the possibilities:

**Option Explicit**— Set to On or Off. On is the default. Requires declaration of all variables before they are used (this is the default).

**Option Compare**— Set to Binary or Text. This specifies if strings are compared using binary or text comparison operations.

**Option Strict**— Set to On or Off. Off is the default. When you assign a value of one type to a variable of another type Visual Basic will consider that an error if this option is on and there is any possibility of data loss, as when you're trying to assign the value in a variable to a variable of less precise data storage capacity. In that case, you must use explicit conversion functions of the kind we'll see in this chapter, like CLng.

You use Option statements first thing in code, like this one in which I'm turning OptionStrict off:

```
Option Strict Off
Module Module1
    Sub Main()
        System.Console.WriteLine("Hello from Visual Basic")
    End Sub
End Module
```

**Imports Statements**

You use Imports statements to import a namespace so you don't have to qualify items in that namespace by listing the entire namespace when you refer to them. If we import the System.Console namespace, that makes that namespace immediately available, so we don't have to qualify the WriteLine method name anymore (note that Option statements, if there are any, must still come first):

```
Option Strict Off
Imports System.Console
Module Module1
    Sub Main()
        WriteLine("Hello from Visual Basic")
    End Sub
End Module
```

# Unit-2

1. **List and explain any 4 unique properties of Textbox.**

| | |
|---|---|
| BackColor | Sets/gets the background color of the control. |
| BorderStyle | Sets/gets the border type of the text box control. |
| ForeColor | Sets/gets the foreground color. |
| MaxLength | Sets/gets the maximum number of characters the user can type into the text box. |
| Multiline | Sets/gets a value specifying if this is a multiline text box control. |
| ReadOnly | Sets/gets a value specifying if text in the text box is read-only. |
| MaxLength | Gets/sets the maximum number of characters in the text box. |
| PasswordChar | Sets/gets the character used to mask characters of a password in a single-line text box. |
| WordWrap | Indicates if a multiline text box control automatically wraps words. |

2. **Write a note on:**
   a. **Rich Textbox**

The Windows forms RichTextBox control is used for displaying, entering, and manipulating rich text with formatting. The RichTextBox control does everything the TextBox control does, but in addition, it can display fonts, colors, and links; load text and embedded images from a file; undo and redo editing operations; and find specified characters.

Rich text format (RTF) text supports a variety of formats. For example, you can color text in a rich text box, underline it, bold it, or make it italic. You can select fonts and fonts sizes, as well as write the text out to disk or read it back in. Rich text boxes also can hold a great amount of data, unlike standard text boxes

RTF text was designed to be a step beyond plain text, and because many word processors let you save text in the rich text format, it can provide a link between different types of word processors. Using rich text boxes, you also can create your own word processors. Rich text boxes are used to support text manipulation and display features similar to the big-time word processors such as Microsoft Word. Like the TextBox control, the RichTextBox control can display scroll bars.

As with the TextBox control, the text displayed is set by the Text property. The RichTextBox control has many properties to format text, and we'll explore them here. You can set font attributes, set indents, create hanging indents, create bulleted paragraphs, and more.

   b. **Inputbox()**

You can use the InputBox function to get a string of text from the user. Here's the syntax for this function:

Public Function InputBox(Prompt As String [, Title As _String = "" [, DefaultResponse As String = "" [, _XPos As Integer = -1 [, YPos As Integer = -1]]]]) As String

And here are the arguments for this function:

Prompt— A string expression displayed as the message in the dialog box. The maximum length is about 1,024 characters (depending on the width of the characters used).

Title— String expression displayed in the title bar of the dialog box. Note that if you omit Title, the application name is placed in the title bar.

DefaultResponse— A string expression displayed in the text box as the default response if no other input is provided. Note that if you omit DefaultResponse, the displayed text box is empty.

XPos— The distance in pixels of the left edge of the dialog box from the left edge of the screen. Note that if you omit XPos, the dialog box is centered horizontally.

YPos— The distance in pixels of the upper edge of the dialog box from the top of the screen. Note that if you omit YPos, the dialog box is positioned vertically about one-third of the way down the screen.

3. **Write a note on handling Mouse Events in VB.NET.**

You can handle mouse events—such as mouse movements—in forms and controls; here are the possible events for the Control class, which is a base class for controls and forms:

MouseDown— Happens when the mouse pointer is over the control and a mouse button is pressed.

MouseEnter— Happens when the mouse pointer enters the control.

MouseHover— Happens when the mouse pointer hovers over the control.

MouseLeave— Happens when the mouse pointer leaves the control.

MouseMove— Happens when the mouse pointer is moved over the control.
MouseUp— Happens when the mouse pointer is over the control and a mouse button is released.
MouseWheel— Happens when the mouse wheel moves while the control has focus.
Here are the properties of the MouseEventArgs object passed to the mouse event handler (not all properties will be filled for all mouse events):
Button— Indicates which mouse button was pressed (see below).
Clicks— The number of times the mouse button was pressed and released.
Delta— A signed count of the number of detents the mouse wheel has rotated.
A detent is the rotation of the mouse wheel one notch.
X— The x-coordinate of a mouse click.
Y— The y-coordinate of a mouse click.

4. **Write a note on**
   a. **Handling keyboard events**

      You can handle keyboard events in forms and many controls with these events:
      KeyDown— Happens when a key is pressed down while the control has focus.
      KeyPress— Happens when a key is pressed while the control has focus.
      KeyUp— Happens when a key is released while the control has focus.

      For KeyDown and KeyUp events, the event handler receives an argument of type KeyEventArgs containing data related to this event, with these properties:
      Alt— Holds a value indicating whether the Alt key was pressed.
      Control— Holds a value indicating whether the Ctrl key was pressed.
      Handled— Holds or sets a value indicating whether the event was handled.
      KeyCode— Holds the keyboard code for a KeyDown or KeyUp event.
      KeyData— Holds the keyboard data for a KeyDown or KeyUp event.
      KeyValue— Holds the keyboard value for a KeyDown or KeyUp event.
      Modifiers— Holds the modifier flags for a KeyDown or KeyUp event. This indicates which modifier keys (Ctrl, Shift, and/or Alt) were pressed.
      Shift— Holds a value indicating whether the Shift key was pressed.

      For KeyPress events, you get an argument of type KeyPressEventArgs containing the following KeyPressEventArgs properties:
      Handled— Gets or sets a value indicating whether the KeyPress event was handled. If you set this value to True, Visual Basic will not handle this key (so if you want to delete it, set Handled to True and do no further processing on it).
      KeyChar— Holds the character corresponding to the key pressed

   b. **MDI forms.**

      Besides standard forms, Visual Basic also supports Multiple Document Interface (MDI) forms. An MDI form closely resembles a standard form, with one major difference—the client area of an MDI form acts as a kind of corral for other forms. That is, an MDI form, also called an MDI parent form, can display MDI children in it, which is how the multiple document interface works.
      MDI forms and MDI child forms are both based on the standard System.Windows.Forms namespace like other Windows forms—you make forms into MDI parents and children by setting the IsMdiContainer and MdiParent properties. the three types of Windows forms available to us in Visual Basic: standard forms, MDI forms, and MDI child forms.

5. **Write the code for creating Textbox.**

```
Private Sub Button1_Click(ByVal sender As System.Object, _ByVal e As System.EventArgs) Handles Button1.Click
    Dim TextBox1 As New TextBox()
    TextBox1.Size = New Size(150, 20)
    TextBox1.Location = New Point(80, 20)
    TextBox1.Text = "Hello from Visual Basic"
    Me.Controls.Add(TextBox1)
End Sub
```

**6. Write the uses of Link Labels with code example.**

They're based on the Label class, but also let you support Web-style hyperlinks to the Internet and other Windows forms. In other words, you can use a link label control for everything that you can use a label control for, and you can also make part of the text in this control a link to a Visual Basic object or Web Page.

Besides functioning as a full label control, you can display multiple hyperlinks in a single link label control, and use the LinkColor, VisitedLinkColor, and ActiveLinkColor properties to set the colors of the link, as you would in a Web page in a browser.

```
Private Sub LinkLabel1_LinkClicked(ByVal sender As Object, _ ByVal e As System.Windows.Forms. LinkLabelLinkClickedEventArgs )
    LinkLabel1.Links(LinkLabel1.Links.IndexOf(e.Link)).Visited = True
    If (e.Link.LinkData.ToString() = "info") Then
            Dim InfoWindow As New Form2()
            InfoWindow.Show()
    End If
End Sub
```

**7. Write VB code for the following.**

**a. Setting Title Bar Text to the form**

```
Private Sub Button1_Click(ByVal sender As System.Object, _ByVal e As System.EventArgs) Handles Button1.Click
    Text = "Welcome to my Application"
End Sub
```

**b. Showing and hiding controls in code**

```
Private Sub Button1_Click(ByVal sender As System.Object, _ByVal e As System.EventArgs) Handles Button1.Click
    TextBox1.visible = True
    TextBox2.visible = False
End Sub
```

**c. Enabling and disabling Forms.**

```
Private Sub Button1_Click(ByVal sender As System.Object, _ByVal e As System.EventArgs) Handles Button1.Click
    Form1.Enabled=False
    Form2.Enabled=True
End Sub
```

**d. Setting Up Startup Form**

Just right-click your project in the Solutions Explorer, select Properties, then select the Common Properties folder and the General item in the box at left. Next, select *form name* from the Startup Object drop-down list on the right, click OK.

**e. Creating Multiple Document Interface (MDI) Applications.**
isMdiContainer=True

**8. <u>Explain MsgBox Function with syntax and example.</u>**

The MessageBox class built into the .NET Framework to display messages and accept input from the user.
Public Function MsgBox(Prompt As Object [, Buttons As MsgBoxStyle = MsgBoxStyle.OKOnly [, Title As Object = Nothing]]) As MsgBoxResultArguments

Here are the arguments you pass to this function:
**Prompt**—A string expression displayed as the message in the dialog box. The maximum length is about 1,024 characters (depending on the width of the characters used).
**Buttons**—The sum of values specifying the number and type of buttons to display, the icon style to use, the identity of the default button, and the modality of the message box. If you omit Buttons, the default value is zero. See below.
**Title**—String expression displayed in the title bar of the dialog box. Note that if you omit Title, the application name is placed in the title bar.

```
Private Sub Button1_Click(ByVal sender As System.Object, _ByVal e As System.EventArgs) Handles Button1.Click
    Dim Result As Integer
    Result = MsgBox("This is a message box!", MsgBoxStyle.OKCancel MsgBoxStyle.Information + MsgBoxStyle.SystemModal, "Message”)
    If (Result = MsgBoxResult.OK) Then
            TextBox1.Text = "You clicked OK"
    End If
End Sub
```

**9. Describe any five noteworthy public properties of CheckBox.**

| | |
|---|---|
| Appearance | Gets/sets the appearance of a checkbox. |
| AutoCheck | Specifies if the Checked or CheckState values and the checkbox's appearance are automatically changed when the checkbox is clicked. |
| CheckAlign | Gets/sets the horizontal and vertical alignment of a checkbox in a checkbox control. |
| Checked | Gets/sets a value indicating if the checkbox is in the checked state. |
| CheckState | Gets/sets the state of a three-state checkbox. |
| FlatStyle | Gets/sets the flat style appearance of the checkbox. |
| Image | Gets/sets the image that is displayed in a checkbox. |

**10. <u>Explain the method to copy or select text to from the clipboard.</u>**

After entering their new novels into your program, users were surprised that they couldn't copy them to the clipboard and so paste them into other applications. How can you support the clipboard with text in a text box? You can use the Clipboard object's SetDataObject and GetDataObject class methods. Here's an example, which is called Clipboard on the CD-ROM. In this case, I'm placing the selected text from one text box into the clipboard when the user clicks a button, and putting it into another text box when

the user clicks another. The call to SetDataObject places the data in the clipboard; GetDataObject gets the data from the clipboard; you can check if it is text data with the GetDataPresent method and the DataFormats enumeration's Text item; and you get the actual data with GetData:

```
Public Class Form1
    Inherits System.Windows.Forms.Form
    'Windows Form Designer generated code

    Private Sub Button1_Click(ByVal sender As System.Object, _ ByVal e As System.EventArgs) Handles
    Button1.Click
        System.Windows.Forms.Clipboard.SetDataObject_ (TextBox1.SelectedText)
    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, _ ByVal e As System.EventArgs) Handles
    Button2.Click
        Dim ClipboardData As System.Windows.Forms.IDataObject = _
        System.Windows.Forms.Clipboard.GetDataObject()
        If ClipboardData.GetDataPresent(DataFormats.Text) Then
            TextBox2.Text = ClipboardData.GetData(DataFormats.Text)
        End If
    End Sub
End Class
```

**11. Explain any three important properties of :**

**a. Rich Textbox**

| | |
|---|---|
| **SelectionLength -** | Sets/gets the number of characters selected in control. |
| **Rtf -** | Sets/gets the text of the RichTextBox control, including all rich text format (RTF) codes. |
| **Modified** - | Control has been modified by the user since the Control was created or its contents were last set. |

**b. Link Label.**

| | |
|---|---|
| **ActiveLinkColor** | Sets/gets the color for an active link. |
| **DisabledLinkColor** | Sets/gets the color for a disabled link. |
| **LinkColor** | Sets/gets the color for a normal link. |

**12. Explain any five important properties of Button.**

| | |
|---|---|
| **DialogResult** | Gets/sets the value returned to the parent form when the button is clicked. Often used when you're creating dialog boxes. |
| **FlatStyle** | Gets/sets a flat style appearance. |
| **Image** | Gets/sets an image displayed in a button. |
| **ImageAlign** | Gets/sets the alignment of the image in a button. |
| **ImageIndex** | Gets/sets the image list index value of the image displayed in the button. |

**13. Explain the process of creating MDI Application.**

The main form, Form1, will be our MDI container or parent, containing MDI children, so set its IsMdiContainer property to True. This alters its appearance from a white client area to a gray one; next, drag a MainMenu control from the toolbox onto Form1. This causes a new control, MainMenu1, to appear in the

new pane at the bottom of the form designer, as you see in Figure 4.13, and a new menu bar to appear in Form1, with the text "Type Here" in it. To create a File menu, type "File" in the "Type Here" box

When you create a new File menu, additional "Type Here" boxes appear for the next menu in the menu bar and the first item in the File menu. Add a New item to the File menu; when you do, another "Type Here" box appears beneath that item. Create a new Arrange menu item in that new box.

Now double-click the New item in the File menu to open the event handler for this menu item in code (as you see from this code, the New item is MenuItem2, not MenuItem1; the File menu itself is MenuItem1):

```
Public Class Form1
    Inherits System.Windows.Forms.Form
    'Windows Form Designer generated code
    Private Sub MenuItem2_Click(ByVal sender As System.Object, _ByVal e As System.EventArgs) Handles
    MenuItem2.Click

    End Sub
⋮
```

When the user clicks the New menu item, we want to display a new MDI child window. To do that, create a new form class, Form2, by adding a new form to the project.

**14. Explain any 5 important methods associated with Form.**

| | |
|---|---|
| **Activate** | Activates the form (gives it focus and makes it active). |
| **Contains** | Indicates if the specified control is a child of this form. |
| **Dispose** | Releases the resources used by the form. |
| **Focus** | Gives the form the focus. |
| **LayoutMdi** | Arranges the MDI child forms within the MDI parent form. |
| **Select** | Selects this form. |
| **ShowDialog** | Displays the form as a modal dialog box. |

# Unit-3

**1. List and explain any four methods of ComboBox.**

| | |
|---|---|
| BeginUpdate | Turns off visual updating of the combo box until the EndUpdate method is called. |
| EndUpdate | Resumes visual updating of the combo box. |
| FindString | Finds the first item in the combo box that begins with the indicated string. |
| FindStringExact | Finds the item that matches the indicated string exactly. |

**2. Write the use of**

**a. ToolTip**

Small windows that appear with explanatory text when you let the mouse rest on a control or window. That's what tool tips are used for—to give quick help when the mouse rests on an item

**b. Progress bar**

Progress bars are those simple controls that show the progress of some operation by displaying rectangles in a horizontal bar

**c. Image List**

Component that does not appear at run time. It store images for use by various controls, including list views, tree views, toolbars, tab controls, checkboxes, buttons, radio buttons, and labels

d. **Timer**

Timers are also very useful controls, because they let you create periodic events

e. **Notify Icon**

Notify icons display icons in the Windows system tray; you can handle events like Click and DoubleClick for these icons

f. **Splitter**

You can use splitters to let the user resize controls.

g. **Trackbar**

Track bars work much like scroll bars, but they have a different appearance

h. **Save File Dialog**

To let the user specify the name of a file to save data to.

**3. Explain any five properties of Date Time Picker.**

| | |
|---|---|
| Checked | Gets/sets whether the Value property holds a valid date-time value. |
| CustomFormat | Gets/sets a custom date-time format string. |
| Format | Gets/sets the format of dates and times. |
| MaxDate | Gets/sets the maximum selectable date and time. |
| MinDate | Gets/sets the minimum selectable date and time. |
| CalendarFont | Gets/sets the font style for the calendar. |

**4. Write a note on**

a. **Toolbar**

those bars full of buttons that appear under menu bars. There are various kinds of options here for the buttons in a toolbar-you can have standard push buttons, toggle buttons (that can appear up or pressed), drop-down buttons that can display a drop-down menu, and buttons that display images. Buttons also can be converted into separators, which display empty horizontal space to separate other buttons.

Typically, the buttons in a toolbar correspond to the most popular menu items in the application. In such cases, the code for a toolbar button is easy to implement-you just use the corresponding MenuItem object's PerformClick method, which clicks the menu item just as if the user did.

Although toolbars are usually docked along the top of its parent window, they can actually be docked to any side of a window. Toolbars also can display tool tips when the user points the mouse pointer at a toolbar button. (Note that to display ToolTips, the ShowToolTips property must be set to True.)

When the Appearance property is set to Normal, the toolbar buttons appear raised (that is, three-dimensional). You can set the Appearance property of the toolbar to Flat to give the toolbar and its buttons a flat appearance. (Note that when the mouse pointer moves over a flat button, the button's appearance changes to three-dimensional.) The TextAlign property specifies the alignment of the text in a button, such as at the top of bottom of the button.

b. **Tab Control**

Tab control is designed to help you conserve space. Tab controls work much like the tabs in a set of folders in a filing cabinet; you can click a tab to display a whole new client area, and each such client area can display other controls. The central property of the TabControl is TabPages, which contains the individual tab

pages in the control, each of which is a TabPage object. When a tab is clicked, it displays its page and causes a Click event for that TabPage object. You can add new tab pages with the TabPages collection's Add method, and remove them with the Remove method.

**5. Explain with example how to determine the items that are checked in a checked list box.**

You can determine if an item displays a checkmark in a checked list box using the GetItemChecked method, which returns True if an item is checked. For example, in the CheckedListBox example, I can loop over all items in the checked list box and display those that are checked in a text box, like this:

```
Private Sub Button2_Click(ByVal sender As System.Object, _ ByVal e As System.EventArgs) Handles Button2.Click
    Dim intLoopIndex As Integer
    Dim strText, strData As String
    strText = "Checked Items: "
    For intLoopIndex = 0 To (CheckedListBox1.Items.Count - 1)
        If CheckedListBox1.GetItemChecked(intLoopIndex) = True Then
            strText &= CheckedListBox1.Items(intLoopIndex).ToString
        End If
    Next
    TextBox1.Text = strText
End Sub
```

That was the hard way; there's an easier way—you can just loop through the collection returned by the CheckedItems property, which holds all the checked items:

```
Dim strText, strData As String
strText = "Checked Items: "
For Each strData In CheckedListBox1.CheckedItems
    strText &= strData & ", "
Next
TextBox1.Text = strText
```

**6. List and explain any <u>three methods</u> and two events and two properties of ListBox.**

| | |
|---|---|
| BeginUpdate | Turns off visual updating of the list box until the EndUpdate method is called. |
| ClearSelected | Unselects all the items in a list box. |
| EndUpdate | Resumes visual updating of the list box. |

| | |
|---|---|
| SelectedIndexChanged | Occurs when the SelectedIndex property has changed. |
| Click | Occurs when the List Box is clicked |

**7. List the Built in dialog controls in VB.NET Explain any three of them.**

- Open File dialogs
- Save File dialogs
- Font dialogs
- Color dialogs
- Print Preview dialogs
- Page Setup dialogs
- Print dialogs

**Open File dialog** lets the user select a file to open. In fact, it's the same Open File dialog used by Windows itself. You can see this dialog box in Figure 9.3, as displayed in the OpenFileDialog example on the CD-ROM.

Open File dialogs are supported with the OpenFileDialog class. You can let users select multiple files with the Multiselect property. You can use the ShowReadOnly property to determine if a read-only checkbox

appears in the dialog box. The ReadOnlyChecked property indicates whether the read-only checkbox is selected. And the Filter property sets the current file name filter string, which determines the choices that appear in the "Files of type" box in the dialog box. The name and path the user selected is stored in the FileName property of the OpenFileDialog object—and there's a neat shortcut here: you can use the OpenFile method to open the selected file directly.

**Save File dialogs** are supported by the SaveFileDialog class. These dialogs let the user specify the name of a file to save data to. These dialogs are the same as the standard Save File dialog box used by Windows from the SaveFileDialog project on the CD-ROM. You can use the ShowDialog method to display the dialog box at run time. You can use the FileName property to get the file the user selected, open a file in read-write mode using the OpenFile method, and so on.

**Font dialogs** let the user select a font size, face, color, and so on. What's handy about these dialogs, besides the fact that they're the same as those used by Windows, is that they return Font and Color objects directly (using the properties of the same name), ready for installation in controls that can use them, like rich text boxes. This saves you the trouble of creating and configuring these objects from scratch.

8. **Write a note on Nodes and SelectedNode properties of TreeView Control, Also write the use of Tree View Control.**

You use a tree view to display a hierarchy of nodes. Each node is not only displayed visually, but also can have child nodes. An example of this is the Windows Explorer, which uses a tree view in its left pane to display the hierarchy of folders on disk. You can expand and collapse parent nodes by clicking them; when expanded, their children are also visible.

| | |
|---|---|
| Nodes | Gets the collection of tree nodes. |
| SelectedNode | Gets/sets the node that is selected. |

# Unit-4

1. **<u>What is Data Binding?</u> <u>Explain different types of data binding in VB.NET.</u>**

**Data binding** is the method by which controls on a user interface (UI) of a client application are configured to fetch from, or update **data** into, a **data** source, such as a database or XML document.

**Simple binding**: The **Simple** Data **Binding** is the process of **binding** the control with the single value in the dataset. The controls like text box, label can be bound to the control through the control properties.
**For eg**: Perform simple binding in code, using a control's **DataBindings** ,corresponding to the bindings for the control. For example, in code, we could bind the text box to the same au_lname field that we just bound it to at design time.
Using the collection's **Add** method, you pass this method the property to bind, the data source to use, and the specific field you want to bind:
TextBox1.DataBindings.Add("Text", DataSet11, "authors.au_lname")

**Complex binding**: complex data binding allows a control to bind to more than one data element, such as more than one record in a database, at the same time.
**Eg**: Complex data binding revolves around these properties:
**DataSource—** The data source, typically a dataset such as **DataSet11**.

**DataMember—** The data member you want to work with in the data source, typically a table in a dataset such as the **authors** table in the pubs database. Data grids use this property to determine which table they should display.

**DisplayMember—** The field you want a control to display, such as the author's last name, au_lname. List boxes use the **DisplayMember** and **ValueMember** properties instead of a **DataMember** property.

**ValueMember—** The field you want the control to return in properties like **SelectedValue**, such as au_id. List boxes use the **DisplayMember** and **ValueMember** properties instead of a **DataMember** property.

**2. Explain the following**

**a. Data Reader**

DataReader objects hold a read-only, forward-only (i.e., you can only move from one record to the succeeding record, not backwards) set of data from a database. Using a data reader can increase speed because only one row of data is in memory at a time

**b. Data Table**

DataTable objects hold a data table from a data source. Data tables contain two important properties: Columns, which is a collection of the DataColumn objects that represent the columns of data in a table, and Rows, which is a collection of DataRow objects, representing the rows of data in a table.

**c. Data Row**

DataRow objects correspond to a particular row in a data table. You use the Item property to get or set a value in a particular field in the row.

**3. Explain the properties and methods associated with OleDbDataAdapter objects.**

| | |
|---|---|
| DeleteCommand | Gets/sets the SQL for deleting records. |
| InsertCommand | Gets/sets the SQL for inserting new records. |
| SelectCommand | Gets/sets the SQL for selecting records. |
| UpdateCommand | Gets/sets the SQL for updating records. |

Fill Adds or refreshes rows to a dataset to make them match the rows in a data store.

**4. Write a note on Data Views.**

Data views represent a customized view of a single table that can be filtered, searched, or sorted. In other words, a data view, supported by the DataView class, is a data "snapshot" that takes up few resources. Data views are much like read-only mini-datasets; you typically load only a subset of a dataset into a data view.

```
Private Sub Form1_Load(ByVal sender As System.Object, _ ByVal e As System.EventArgs) Handles MyBase.Load
    DataSet11.Clear()
    SqlDataAdapter1.Fill(DataSet11)
End Sub
```

**5. Write a note on OleDbConnection Class**

An **OleDbConnection** object supports a connection to an OLE DB data provider. A central property of connection objects is the **ConnectionString** property, which holds a string full of attribute/value pairs that contain data needed to log on to a data provider and choose a specific database. These attribute/value pairs are specific to the data provider you're using, and make up a list of items separated by semicolons. You can either assign a connection string to the connection's **ConnectionString** property, or you can pass the connection string to the connection object's constructor, like this:

Dim ConnectionString As String = "Provider=SQLOLEDB.1;Integrated" & Security=SSPI;Persist Security Info=False;Initial" & "Catalog=pubs;Packet Size=4096;Workstation ID=STEVE;" & "Use Encryption for Data=False"

Dim Connection1 As OleDbConnection = New OleDbConnection(ConnectionString)

| Property | Means |
|---|---|
| **ConnectionString** | Gets/sets the connection string to open a database. |
| **ConnectionTimeout** | Gets the amount of time to wait trying to make a connection. |
| **Database** | Gets the name of the database to open. |
| **DataSource** | Gets the data source (usually the location and file name to open). |
| **Provider** | Gets the OLE DB provider's name. |
| **ServerVersion** | Gets the version of the server. |
| **State** | Gets the connection's current state. |

| Method | Means |
|---|---|
| **BeginTransaction** | Starts a database transaction. |
| **ChangeDatabase** | Changes the current database. |
| **Close** | Closes the connection to the data provider. |
| **CreateCommand** | Creates an **OleDbCommand** object for this connection. |
| **GetOleDbSchemaTable** | Returns the current schema table. |
| **Open** | Opens a database connection. |
| | |

| Event | Means |
|---|---|
| **InfoMessage** | Occurs if the provider sends a message (including warnings). |
| **StateChange** | Occurs when a connection's state changes. |

6. **Explain the process of creating Data Forms using Data Form Wizard.**

There's an easy way to create a data-entry form in Visual Basic—just use the Data Form Wizard. example on the CD-ROM, which creates a data-entry form for the authors table in the pubs database.

A data form is a new form added to your project, so to create such a form, DataForm1.vb, use the Project|Add New Item menu item, then select the Data Form Wizard icon in the Templates pane and click OK. This opens the Data Form Wizard

Click the Next button in the Data Form Wizard to move to the pane, where the Wizard is asking for the name of a dataset to create (or you can use an existing dataset); I'll name the new dataset dsDataSet1 here.

In the next pane, the Wizard asks what data connection to use (or allows you to create a new connection), and I'll use a connection to the pubs database.

In the next pane, you can choose which table(s) to add to the data form, and I'll add the authors table

If you are working with multiple tables, you can create a master/detail relationship between the tables in the next pane. I'll just click next to move on

In the next pane, you can select the display style—whether the data form will use a data grid or separate, simply bound controls. I'll specify separate controls here; that means the data form also can contain Add, Delete, and other controls.
Finally, click the Finish button to create the data form, DataForm1.vb, and add it to the project,where I've clicked the Load button to load the authors table. In this case, I've added code to the main form to make the data form visible:

```
Private Sub Form1_Load(ByVal sender As System.Object, _ByVal e As System.EventArgs) Handles MyBase.Load
    Dim d As New DataForm1()
    d.Show()
End Sub
```

In the data form, you can see all the data in the current record displayed (you can move the controls around in a data form if the default layout doesn't suit you, of course), as well as navigation buttons, and Add, Delete, Cancel, and Update buttons. The Add, Delete, Cancel, and Update buttons let you edit the data in the dataset in the form, and send it back to the database. When the user changes the data in the bound controls, the changed data is sent back to the dataset immediately, starting an edit operation in the dataset—note that any changes to the dataset are only sent back to the database when the user clicks the Update button. The Add button adds a new empty record to the end of the dataset, the Delete button deletes a record, and the Cancel button cancels any edit operation in the dataset that hasn't been sent back to the database yet. I'll take a look at the data form code for these various operations here.

**7. Explain the following ADO.Net objects :**

**a. Data Connection Objects**

To start working with a database, we must have a data connection. A data adapter needs a connection to a data source to read and write data, and it uses OleDbConnection or SqlConnection objects to communicate with a data source.

**b. Data Adapters**

Data adapters are a very important part of ADO.NET. We use them to communicate between a data source and a dataset. We typically configure a data adapter with SQL to execute against the data source. The two types of data adapters are OleDbDataAdapter and SqlDataAdapter objects.

**c. Command Objects**

Data adapters can read, add, update, and delete records in a data source. To allow you to specify how each of these operations work, a data adapter contains command objects for each of them. Data adapters support four properties that give you access to these command objects: SelectCommand, InsertCommand, UpdateCommand, and DeleteCommand

**d. Data Sets**

The DataSet is a memory-resident representation of data that provides a consistent relational programming model regardless of the data source. It can be used with multiple and differing data sources, with XML data, or to manage data local to the application. The DataSet represents a complete set of data, including related tables, constraints, and relationships among the tables.

**8. Explain the following , give an example for each.**

**a. SELECT Command.**

You use the SELECT statement to get fields from a table; here's an example where I'm getting all the records in the Customers table, using the wildcard character *:

```
SELECT * FROM Customers
```

This returns a dataset that holds all the records in the Customers table. You also can use the SELECT statement to select specific fields from a table, like this, where I'm selecting the CustomerID, Address, and City fields of all the records in the Customers table:

SELECT CustomerID, Address, City FROM Customers

This returns a dataset that holds all the records in the Customers table, and each record will have a CustomerID, Address, and City field.

**b. Using Where Clauses**

In SQL, you can use the WHERE clause to specify criteria that you want records to meet. For example, to select all the records in the Customers table where the City field holds "Boston", you can execute this statement:

SELECT * FROM Customers WHERE City = "Boston"

You don't have to use an equals sign here; you can test fields using these operators:

- < (less than)
- <= (less than or equal to)
- > (greater than)
- >= (greater than or equal to)
- BETWEEN
- IN
- LIKE

**c. Using the Between Clause**

BETWEEN clause to indicate a range of values you will accept. For example, here's how to select all the records from the Customers table where the CustomerID record starts with the letter H (the CustomerID field is alphabetic, not numeric, in this table):

SELECT * FROM Customers WHERE CustomerID BETWEEN

"H*" AND "I*"

Note the use of wildcard characters: "H*" and "I*". Using these wildcards lets you specify that you want all the CustomerID values that start with the letter H.

**d. Using the IN Clause**

IN clause to specify a set of values that fields can match. For example, here's how I get records that have values in the City field that match Boston or York:

SELECT * FROM Customers WHERE City IN ("Boston","York")

**e. Using the ORDER BY clause.**

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

ORDER BY Syntax

Select col, col2 from student order col ASC|DESA

**9. <u>Explain the process of navigating DataSets with Suitable Code Example.</u>**

We can use a BindingContext object's Position property to move through a dataset, setting the current record that simple-bound controls are bound to and display

**Part cLab program-using binding source (LOL)**

**10. Write a note on Creating Command Object in code.**

After creating a connection object to connect to the authors table in the pubs database in the DataSetCode example on the CD-ROM (see the previous two topics), we need a command object to load the authors table into our dataset. Here's how I create an OleDbCommand object, give it the SQL "SELECT * FROM authors" and set the command's type to CommandType.Text (which is the value you use for SQL, and is the default). Then, after opening the connection object we created in the previous topic,

assign that connection object to the command object's Connection property:

```
Private Sub Button1_Click(ByVal sender As System.Object, _ ByVal e As System.EventArgs) Handles Button1.Click
    Dim ds As New DataSet()
    ds = New DataSet("authors")
    Dim ConnectionString As String = "Provider=SQLOLEDB.1;Integrated "Security=SSPI;Persist Security Info=False;Initial " & _
    "Catalog=pubs;Packet Size=4096;Workstation ID=STEVE;" & _
    "Use Encryption for Data=False"
    Dim Connection1 As OleDbConnection = New _
    OleDbConnection(ConnectionString)
    Dim Command1 As OleDbCommand = _
    New OleDbCommand("SELECT * FROM authors")
    Command1.CommandType = CommandType.Text
    Connection1.Open()
    Command1.Connection = Connection1
    ⋮
```

Our command object is now ready to be used with a data adapter to get the authors table from the pubs database.

**11. Write a note on creating Data Connection in Code.**

To create a dataset in code in the DataSetCode example on the CD-ROM, and to load the authors table from the pubs database into it, I start by creating a dataset object when the user clicks the "Load data" button in this example:

```
Private Sub Button1_Click(ByVal sender As System.Object, _ ByVal e As System.EventArgs) Handles Button1.Click
    Dim ds As New DataSet()
    ds = New DataSet("authors")
    ⋮
```

Now we'll need a connection object to connect to the authors table, and I create that connection like this, using a connection string (for more on creating connection strings, see the In Depth section of this chapter):

```
Private Sub Button1_Click(ByVal sender As System.Object, _ ByVal e As System.EventArgs) Handles Button1.Click
    Dim ds As New DataSet()
    ds = New DataSet("authors")
    Dim ConnectionString As String = "Provider=SQLOLEDB.1;Integrated "Security=SSPI;Persist Security Info=False;Initial " & _
    "Catalog=pubs;Packet Size=4096;Workstation ID=STEVE;" & _
    "Use Encryption for Data=False"
    Dim Connection1 As OleDbConnection = New _
    OleDbConnection(ConnectionString)
    ⋮
```

## IF YOU REACHED THIS FAR YOU'RE GREAT
## ALL THE BEST!