Experiment No. (1)

Introduction to Visual Basic.NET

Object:

To identify main visual basic's objects and their properties, methods and events.

Theory:

Visual basic introduced in 1991, visual basic.net introduced in 2002/2003.The latest version of Microsoft's Visual Studio, called Visual Studio 2015, includes Visual Basic, Visual C++, Visual C# (C sharp), Visual F# (F sharp),JScript, and the .NET 4.6 Framework.

The programming languages in Visual Studio run in the .NET Framework. The Framework provides for easier development of Web-based and Windows-based applications, allows objects from different languages to operate together, and standardizes how the languages refer to data and objects.

Microsoft Visual Basic comes with Visual Studio. You also can purchase VB by itself (without the other languages but *with* the .NET Framework). VB is available in an Express version, and in Visual Studio 2010 Professional**,** Visual Studio 2010 Premium, and Visual Studio 2010 Ultimate.

NOTE: Anyone planning to do professional application development that includes the advanced features of database management should use the **Professional version** or higher. The Professional version is available to educational institutions.

Programming Languages—Procedural, Event Driven, and Object Oriented

Procedural languages such as BASIC, C, COBOL, FORTRAN, PL/I, and Pascal.

*Function oriented programming (FOP).
*Functions and subroutines.
*Code executed by run the program.
*Can't change code through runtime and difficult error correction.

The newer programming languages, such as Visual Basic, C#, and Java, use a different approach: object-oriented programming **(OOP)**

*Use objects (controls).
*Event driven: code executed upon some events (button press, mouse move, click, ….).
*Easy to change code and errors.

Object Model
In Visual Basic you will work with objects, which have properties, methods, and events. Each object is based on a class.

Objects
Examples of objects are forms and controls. Forms are the windows and dialog boxes you place on the screen; controls are the components you place inside a form, such as text boxes, buttons, and list boxes.

Properties
Properties tell something about or control the behavior of an object, such as its name, color, size, or location. Example object. property=value textbox.name= new

Methods
Built in procedure actions associated with objects are called methods. Some typical methods are Close, Show, and Clear. Each of the predefined objects has a set of methods that you can use.

You refer to methods as Object. Method Example: Button. Show

Events
An event occurs when the user takes an action, such as clicking a button, pressing a key, scrolling, or closing a window.

Classes

Classes contain the definition of all available properties, methods, and events. Each time that you create a new object, it must be based on a class. For example, you may decide to place three buttons on your form. Each button is based on the Button class and is considered one object, called an *instance* of the class.

Three steps to start V.B. program and to build VB application

1-draw user interface (use toolbox controls (objects)).

2-set properties values for each object.

3-write the code (attach code for each control (object)) then execute the program.

Example1:-

Object-event()	command1-click()
User code	code
End	end

Visual basic operates in three modes:-

1-design mode:build at design time(use property list for each object).

2-run mode:run application.

3-break mode:debugger available to correct errors.

Visual Studio Environment

The Visual Studio environment is where you create and test your projects. A development environment, such as Visual Studio, is called an integrated development environment (IDE). The IDE consists of various tools, including a form designer, which allows you to visually create a form; an editor, for entering and modifying program code; a compiler, for translating the Visual Basic statements into the intermediate machine code; a debugger, to help locate and correct program errors; an object browser, to view available classes, objects, properties, methods, and events; and a Help facility.

In versions of Visual Studio prior to .NET, each language had its own IDE. For example, to create a VB project you would use the VB IDE, and to create a C++ project you would use the C++ IDE. But in Visual Studio, you use one IDE to create projects in any of the supported languages.

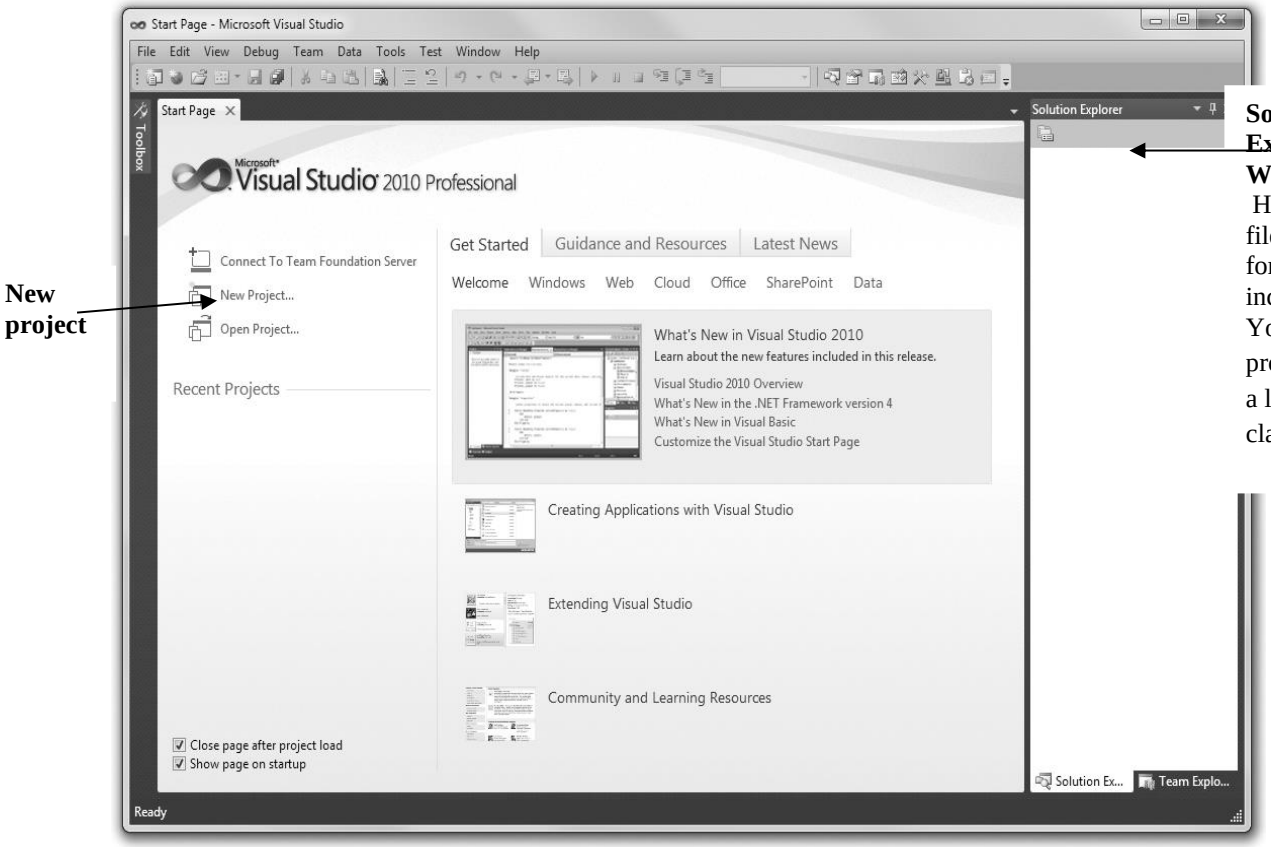


Figure (1.1) Visual Studio IDE with the Start Page open, as it first appears in Windows 7, without an open project.

To write your first Visual Basic project:

*STEP 1***:** Click the Windows ***Start*** button and move the mouse pointer to *All Programs*.

STEP 2: Locate *Microsoft Visual Studio 2010* or *Microsoft Visual Basic 2010.*

STEP 3: If a submenu appears, select *Microsoft Visual Studio 2010*.Visual Studio (VS) will start and display the Start Page(figure above). If you are using Visual Studio Professional and this is the first time that VS has been opened on this computer, you may need to select

Visual Basic Development Settings from the *Choose Default Environment.*

Figure (1.2) Setting dialog box

Steps for starting a new project:

Begin a new VB Windows project using the Windows Forms Application template.
This is a simple example to write visual basic program that explain how to enter text in textbox and display it in messagebox.

After you click on New Project this window will pop up:

- Click on the first icon Windows Application, and name your first application and then click ok (As shown in figure 1.3).
- Now you can see a small form on the corner of the application, and the controls on its left (As shown in figure 1.4).
- Let's add a text box: a text box is a text area that allows the user to enter text.
- In the Tool Box on the left, click on TextBox.
- Draw a shape on the form to draw the TextBox (Figure 1.5).
- After placing the TextBox on the form, you can move anywhere on the Form.

- Now insert a label (Figure1.6).
- Find the Label control from the Tool Box and place it on the Form.

Windows Form Application

Visual Basic

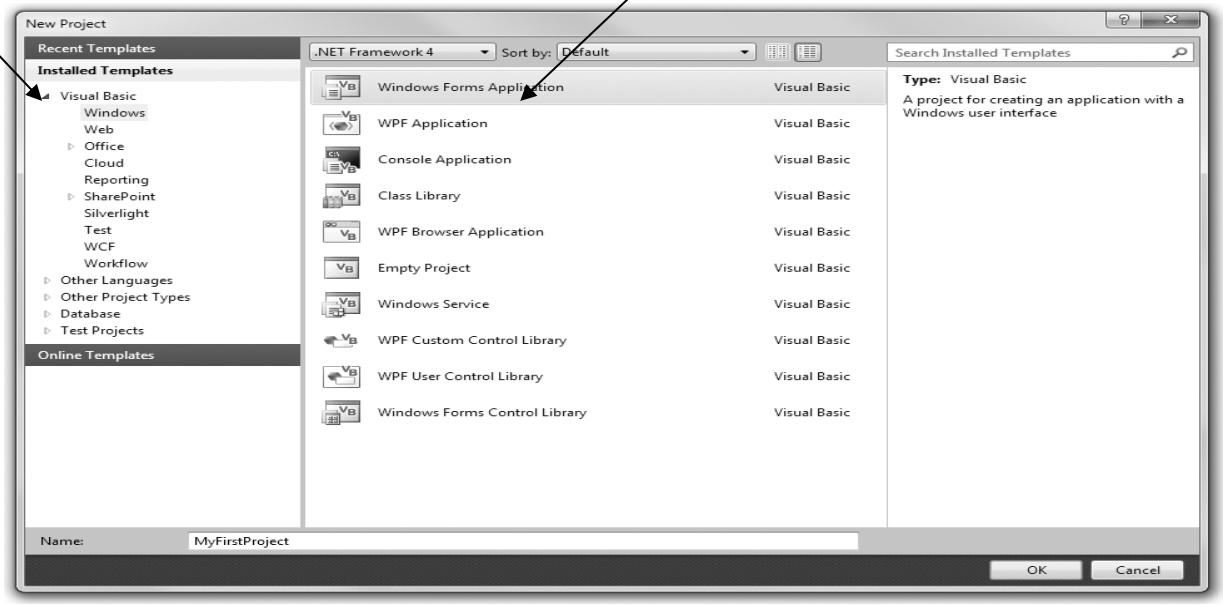


Figure (1.3)

Form Designer Window

Tool Box contain controls (objects) used in the project

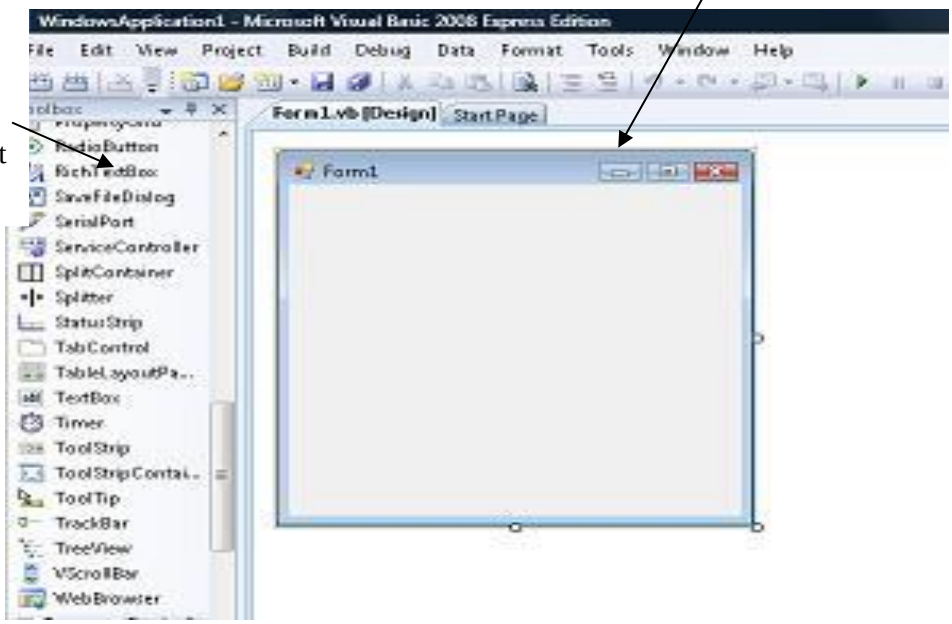


Figure (1.4)

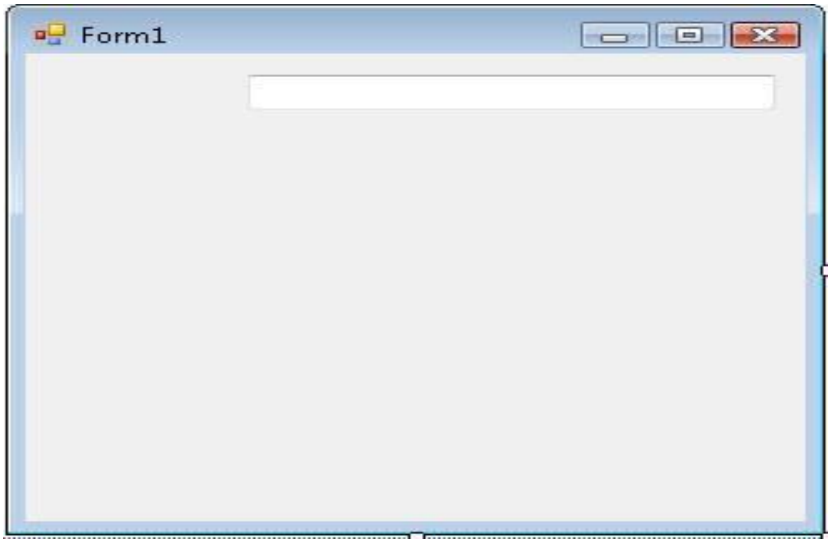


Figure (1.5)

- Name the Label1 "Name". To do that right click on Label1, click on Properties
- The Properties window will appear on the right (contain the properties of the control).
- Change the property of Text to Name (figure 1.7):
- Add a new Textbox and a new Label and a button as the picture below, and name the second change the new label's text properties to Last and the button's text to Get Name (figure 1.8)::

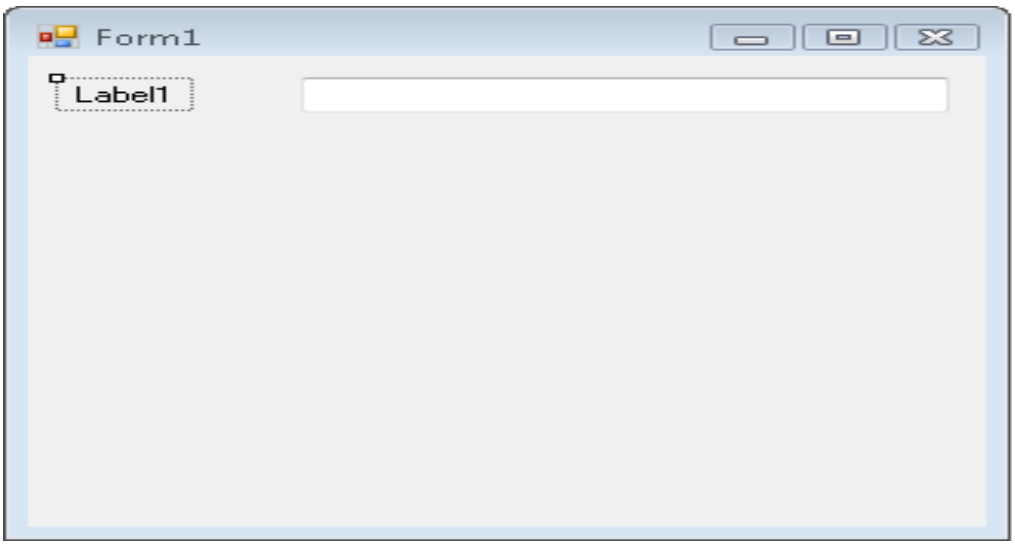


Figure (1.6)

Property window of the label control (object)

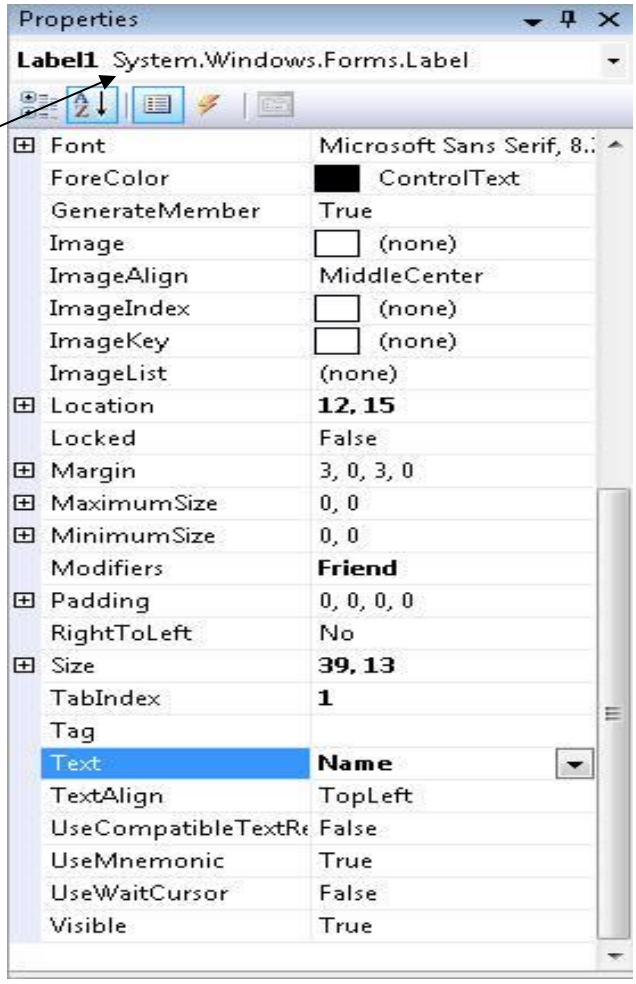


Figure (1.7)

Figure (1.8)

Writing code:

- Double click on the new button that says "Get Name" to go to the codes page.
- You will find a code like this:

```
Public Class Form1

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

    End Sub
End Class
```

- Now we will tell the form to produce a message box and show the name and the last name when we click on "Get Name"
- You will enter this code between the second line and the third line:

```
MsgBox("My Name is " & TextBox1.Text & " " & TextBox2.Text)
```

- After adding this code run the program
- To run the program (F5 hot key) or (menu bar DEBUG then click start).
- Now enter a name inside the first textbox, and a last name inside the second textbox, and then click on "Get Name" and you should get something like this:

Figure (1.9)

Procedure:

Write a V.B Program to make a calculator that calculates two textboxes automatically without any control to explain how to use textbox to enter numbers. Add three textboxes to the form. Add them in this order: put the first one on the left side, the second on the middle and the third one the right like figure (1.10):

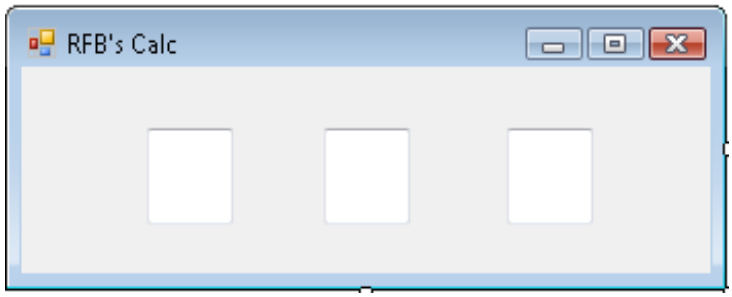


Figure (1.10)

Discussion:

Write a V.B Program to make a calculator that calculates two textboxes with a button to explain how to use textbox to enter numbers and the button to find the result. Discuss the difference between this calculator and the one in the previous example.

Experiment No. (2)

Variables + Data Types + Built in Functions

Object:

Learning: variables declaration, variant data types (which store numeric, date/time or string data) and the categories of built in functions.

Theory:

Variables

Variables are the memory locations which are used to store values temporarily. A variable name must begin with an alphabet letter It must be unique within the same scope. It should not contain any special character like %, &, !, #, @ or $.The name cannot be a reserved word, such as Sub or Double.

By default Visual Basic variables are of variant data types. The variant data type can store numeric, date/time or string data (as shown in table 2.1).

Declaring a Variable Syntax

Dim|private|static *variableName* **As** *dataType*

Examples

```
Dim intAge As Integer
Dim decPayRate As Decimal
Dim dblPrice As Double
```

Private — you can use the Private keyword only for variables declared at the module, class, or structure, not inside a subroutine. A variable declared Private is accessible only to code in the same module, class, or structure. If the variable is in a class or structure, it is available to other

instances of the class or structure. For example, one Customer object can access a Private variable inside another Customer object.

Static — you can use the Static keyword only for variables declared within a subroutine or a block within a subroutine. A variable declared Static keeps its value between lifetimes. For example, if a subroutine sets a Static variable to 27 before it exits, the variable begins with the value 27 the next time the subroutine executes. The value is stored in memory, so it is not retained if you exit and restart the whole program. Use a database, the System Registry, or some other means of permanent storage if you need to save values between program runs.

Table (2.1)

Data type	Stores	Required Memory
Boolean	a logical value (True, False)	2 bytes
Char	one Unicode character	2 bytes
Date	date and time information Date range: January 1, 0001 to December 31, 9999 Time range: 0:00:00 (midnight) to 23:59:59	8 bytes
Decimal	a number with a decimal place Range with no decimal place: +/–79,228,162,514,264,337,593,543,950,335 Range with a decimal place: +/–7.9228162514264337593543950335	16 bytes
Double	a number with a decimal place Range: +/–4.94065645841247 X 10–324 to +/–1.79769313486231 X 10308	8 bytes
Integer	integer Range: –2,147,483,648 to 2,147,483,647	4 bytes
Long	integer Range: –9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	8 bytes
Object	data of any type	4 bytes
Short	integer Range: –32,768 to 32,767	2 bytes
Single	a number with a decimal place Range: +/–1.401298 X 10–45 to +/–3.402823 X 1038	4 bytes
String	text; 0 to approximately 2 billion characters	

TYPE of CHARACTERS

Data type characters identify a value's data type. The following table lists the data type characters of Visual Basic.

CHARACTER DATA TYPE

% Integer
& Long
@ Decimal
! Single
Double
$ String

You can specify a variable's data type by adding a data type character after a variable's name when you declare it.

```
Dim num_desserts &
Dim satisfaction_quotient#

num_desserts = 100
satisfaction_quotient# = 1.23
```

Using Constants

In addition to reserving (or declaring) variables in a program, you also can declare named constants. A named constant is a memory location whose value cannot change while the application is running.

Declaring a Named Constant Syntax

Const *constantName* **As** *dataType* = *value*

Examples

```
Const dblPI As Double = 3.141593
Const intMAX_HOURS As Integer = 40
Const decTAXRATE As Decimal = .05
```

The TryParse Method (or VAL method)

Every numeric data type in Visual Basic has a TryParse method that can be used to convert text to that numeric data type.

Basic Syntax of the TryParse Method

dataType.**TryParse(***text***,** *variable***)**

Example 1

Double.TryParse(txtRadius.Text, dblRadius). If the text entered in the txtRadius control can be converted to a Double number, the TryParse method converts the text and then stores the result in the dblRadius variable; otherwise, it stores the number 0 in the dblRadius variable. Or we can use instead dblradius = **val**(txtradius.text).

The Convert Class

At times, you may need to convert a number (rather than a string) from one data type to another.

Using the Convert class methods

Convert.*method***(***value***)**

Example 2

decTaxRate = Convert.ToDecimal(.05)
converts the Double number .05 to Decimal and then assigns the result to the decTaxRate variable.

Example 3

lblTotal.Text = Convert.ToString(intTotalScore)
converts the integer stored in the intTotalScore variable to String and then assigns the result to the lblTotal control's Text property.

OPTION EXPLICIT AND OPTION STRICT

The Option Explicit and Option Strict compiler options play an important role in variable declarations. When Option Explicit is set to On, you must declare all variables before you use them. If Option Explicit is Off, Visual Basic automatically creates a new variable whenever it sees a variable that it has not yet encountered. For example, the following code doesn't explicitly declare any variables. As it executes the code, Visual Basic sees the first statement, num_managers = 0. It doesn't recognize the variable num_managers , so it creates it. Similarly, it creates the variable i when it sees it in the For loop.

```
Option Explicit Off
Option Strict Off
Public Class Form1
...
Public Sub CountManagers()
num_managers = 0
For i = 0 To m_Employees.GetUpperBound(0)
If m_Employees(i).IsManager Then num_managrs += 1
Next i
MessageBox.Show(num_managers)
End Sub
...
End Class
```

Keeping Option Explicit turned off can lead to two very bad problems. First, it silently hides typographical errors. If you look closely at the preceding code, you'll see that the statement inside the For loop increments the misspelled variable num_managrs instead of the correctly spelled variable num_managers. Because Option Explicit is off, Visual Basic assumes that you want to use a new variable, so it creates num_managrs . After the loop finishes, the programdisplays the value of num_managers , which is zero because it was never incremented

The second problem that occurs when **Option Explicit is off** is that Visual Basic doesn't really know what you will want to do with the variables it creates for you. It doesn't know whether you will use a variable as an Integer, Double, String, or PictureBox.

When Option Strict is turned off, Visual Basic silently converts values from one data type to another, even if the types are not very compatible. For example, Visual Basic will allow the following code to try to copy the string s into the integer i . If the value in the string happens to be a number (as in the first case), this works. If the string is not a number (as in the second case), this throws an error at runtime.

```
Dim i As Integer
Dim s As String
s = "10"
i = s ' This works.
s = "Hello"
i = s ' This Fails.
```

If you turn **Option Strict on**, Visual Basic warns you of possibly illegal conversions at compile time.

ARITHMETIC OPERATORS

The following table lists the arithmetic operators provided by Visual Basic. Most programmers should be very familiar with most of them. The four operators that may need a little extra explanation are \ , Mod , < < , and >>, as shown in table (2.2).

Table (2.2)

OPERATOR	PURPOSE	EXAMPLE	RESULT
^	Exponentiation	2 ^ 3	(2 to the power 3) = 2 * 2 * 2 = 8
–	Negation	2	- 2
*	Multiplication	2 * 3	6
/	Division	3 / 2	1.5
\	Integer division	17 \ 5	3
Mod	Modulus	17 Mod 5	2
+	Addition	2 + 3	5
< <	Bit left shift	10110111 < < 1	01101110
> >	Bit right shift	10110111 > > 1	01011011

COMPARISON OPERATORS

Comparison operators compare one value to another and return a Boolean value (True or False), depending on the result. The following table lists the comparison operators provided by Visual Basic. The fi rst six (= , < > , < , < = , > , and > =) are relatively straightforward. Note that the Not operator is not a comparison operator, so it is not listed here. It is described in the next section, " Logical Operators. ", as shown in table (2.3)

Table (2.3)

OPERATOR	PURPOSE	EXAMPLE	RESULT
=	Equals	A = B	True if A equals B
< >	Not equals	A < > B	True if A does not equal B
<	Less than	A < B	True if A is less than B
< =	Less than or equal to	A < = B	True if A is less than or equal to B
>	Greater than	A > B	True if A is greater than B
> =	Greater than or equal to	A > = B	True if A is greater than or equal to B
Is	Equality of two objects	emp Is mgr	True if emp and mgr refer to the same
IsNot	Inequality of two objects	emp IsNot mgr	True if emp and mgr refer to different objects
TypeOf Is	Object is of a certaintype	TypeOf(obj) IsManager	True if obj points to aManager object
Like	Matches a text pattern	A Like " ### - #### "	True if A contains three digits, a dash and four digits

LOGICAL OPERATORS

Logical operators combine two Boolean values and return True or False, depending on the result. The following table (2.4) summarizes Visual Basic's logical operators.

Table (2.4)

OPERATOR	PURPOSE	EXAMPLE	RESULT
Not	Logical or bitwise negation	Not A	True if A is false
And	Logical or bitwise And	A And B	True if A and B are both true
Or	Logical or bitwise Or	A Or B	True if A or B or both are true
Xor	Logical or bitwise exclusive Or	A Xor B	True if A or B but not both is true
AndAlso	Logical or bitwise And withshort - circuit evaluation	A AndAlso B	True if A and B are both true
OrElse	Logical or bitwise Or withshort - circuit evaluation	A OrElse B	True if A or B or both are true

Built-in Functions in visual basic 2010

The functions fall into the following basic categories that will be discussed in the following sections at length(see table 2.5).

- Date and Time Functions
- Format Function
- String Functions

*The **Now** function retrieves the **date** and **time**, while **Date** function retrieves only date and **Time** function retrieves only the time.

Table (2.5)

Format	Explanation
Format (Now, "General date")	Formats the current date and time.
Format (Now, "Long Date")	Displays the current date in long format.
Format (Now, "Short date")	Displays current date in short format
Format (Now, "Long Time")	Display the current time in long format.
Format (Now, "Short Time")	Display the current time in short format.

The Format Functions

is a very powerful formatting function which can display the numeric values in various forms. The format of the predefined Format function is: Format (n, "style argument"). Where n is a number and the list of style arguments is given in Table (2.6).

Table (2.6)

Style argument	Explanation	Example
General Number	To display the number without having separators between thousands.	Format(8972.234, "General Number")=8972.234
Fixed	To display the number without having separators between thousands and rounds it up to two decimal places.	Format(8972.2, "Fixed")=8972.23
Standard	To display the number with separators or separators between thousands and rounds it up to two decimal places.	Format(6648972.265, "Standard")= 6,648,972.27
Currency	To display the number with the dollar sign in front, has separators between thousands as well as rounding it up to two decimal places.	Format(6648972.265, "Currency")= $6,648,972.27
Percent	Converts the number to the percentage form and displays a % sign and rounds it up to two decimal places.	Format(0.56324, "Percent")=56.32 %

Example5: this example to explain how to use Format function.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click, Button5.Click, Button4.Click,
Button3.Click
Label1.Text = Format(8972.234, "General Number")
Label2.Text = Format(8972.2, "Fixed")
Label3.Text = Format(6648972.265, "Standard")
Label4.Text = Format(6648972.265, "Currency")
Label5.Text = Format(0.56324, "Percent")
End Sub
```

The Output window is shown in figure (2.1) below:

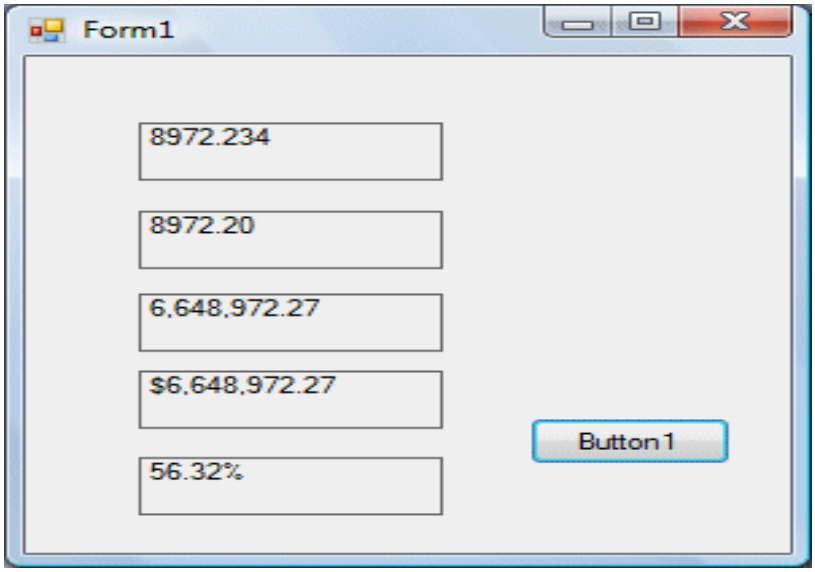


Figure (2.1)

Mathematics and String Functions

**The Abs function*

The Abs function returns the absolute value of a given number

The syntax is

Math. Abs (number)

**The Exp function*

The Exp of a number x is the exponential value of x, i.e. exp . For example, Exp(1)=e=2.71828182

The syntax is

Math.Exp (number)

**The Fix Function*

The Fix function truncates the decimal part of a positive number and returns the largest integer smaller than the number. However, when the

number is negative, it will return smallest integer larger than the number. For example, Fix(9.2)=9 but Fix(-9.4)=-9.

**The Int Function*

The Int is a function that converts a number into an integer by truncating its decimal part and the resulting integer is the largest integer that is smaller than he number. For example

Int(2.4)=2, Int(6.9)=6 , Int(-5.7)=-6, Int(-99.8)=-100

**The Log Function*

The Log function is the function that returns the natural logarithm of a number. For example, Log(10)=2.302585.

**The Rnd( ) Function*

The Rnd is very useful when we deal with the concept of chance and probability. The Rnd function returns a random value between 0 and 1. Random numbers in their original form are not very useful in programming until we convert them to integers.

**The Round Function*

The Round function is the function that rounds up a number to a certain number of decimal places. The Format is Round (n, m) which means to round a number n to m decimal places. For example, Math.Round (7.2567, 2) =7.26

Example 6: how to use **Round** function.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)HandlesButton1.Click


    Dim num1, num2 as single
    num1=TextBox1.Text
    num2=Math.Round(num1) Label1.Text = num2

End Sub
```

String functions

- *The Mid Function*

The Mid function is used to retrieve a part of text form a given phrase. The format of the Mid Function is

Mid(phrase, position,n)

x=Mid(“newdesign”,3,2) , x=wd

- *The Right Function*

The Right function extracts the right portion of a phrase

Microsoft.Visualbasic.Right ("Visual Basic", 4) = asic

- *The Left Function*

The Left function extracts the left portion of a phrase. The format is

Microsoft.Visualbasic.Right ("Phrase", n)

Microsoft.Visualbasic.Left("Visual Basic", 4) = visu

- Len function

Length of the string example x=Len(“basic”) , x=5

- *The Trim Function*

The Trim function trims the empty spaces on both side of the phrase. The format is

Trim (" Visual Basic ") = Visual basic

- *The Ltrim Function, The Rtrim Function*

Ltrim (" Visual Basic")= Visual basic, Rtrim ("Visual Basic ") = Visual Basic

- *The InStr function*

Instr (n, original phase, embedded phrase)

Where n is the position where the Instr function will begin to look for the embedded phrase. For Example:

Instr(1, "Visual Basic","Basic")=8

The function returns a numeric value.

- *The Ucase and the Lcase Functions*

  Microsoft.VisualBasic.Ucase("Visual Basic") =VISUAL BASIC

  Microsoft.VisualBasic.Lcase("Visual Basic") =visual basic

- *The Chr and the Asc functions*

  Chr(65)=A, Chr(122)=z, Chr(37)=% ,

  Asc("B")=66, Asc("&")=38

Procedure

Write a V.B program behave like a virtual dice. (make use of Rnd and Int functions).

Discussion

Write a V.B program to explain how to use Now, Date, Time functions.

Experiment No. (3)

IF…..Then + Loop Structures

Object

Allow the user to learn arrays and Visual Basic control structures such as If ..... Then.... Else, Select ....Case, in addition to the loop structures such as Do while ....Loop, For ....Next, etc.

Theory

Visual Basic supports control structures such as if... Then, if...Then ...Else, Select...Case, and Loop structures such as Do While...Loop, For...Next etc method.

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
```

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.x.:

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

Nested If...Then...Else selection structure

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

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

You can use Nested If either of the methods as shown below.

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
```

EX: 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
```

Nested If Statements

```
It’s possible to nest an If statement inside another:
If intX = 3 Then
MessageBox.Show("intX = 3")
If intY = 6 Then
MessageBox.Show("intY = 6")
End If

End If
```

The Select Case...End Select Structure

Select Case is preferred when there exist multiple conditions because using If...Then..ElseIf statements will become too messy.

The format of the Select Case control structure is as follows:

Select Case test expression

Case expression list 1
Block of one or more VB statements
Case expression list 2
Block of one or more VB Statements
Case expression list 3
Block of one or more VB statements
Case expression list 4
Case Else
Block of one or more VB Statements

End Select

The usage of Select Case is shown in the following examples:

Example1: in this example, you can use the keyword **Is** together with the comparison operators.

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

```
'Examination Marks
 Dim mark As Single
 mark = mrk.Text
  Select Case mark
 Case Is >= 85
      Label1.Text= "Excellence"
Case Is >= 70
     Label2.Text= "Good"
Case Is >= 60
   Label3.Text = "Above Average"
Case Is >= 50
Label4.Text= "Average"
Case Else
Label5.Text = "Need to work harder"
End Select
End Sub
```

Example1 can be rewritten as follows:

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

```
'Examination Marks
Dim mark As Single
```

```
 mark = Textbox1.Text
Select Case mark
 Case 0 to 49
     Label1.Text = "Need to work harder"
Case 50 to 59
    Label1.Text = "Average" s
Case 60 to 69
   Label1.Text= "Above Average"
Case 70 to 84
Label1.Text = "Good"
Case 85 to 100
Label1.Text= "Excellence"
Case Else
Label1.Text= "Wrong entry, please reenter the mark"
End Select
 End Sub
```

Example2: Grades in high school are usually presented with a single capital letter such as A, B, C, D or E. The grades can be computed as follow:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As

System.EventArgs) Handles Button1.Click

'Examination Marks

Dim mark As Single
mark = TextBox1.Text
Select Case mark
Case 0 To 49
Label1.Text = "E"
Case 50 To 59
Label1.Text = "D"
Case 60 To 69
Label1.Text = "C"
Case 70 To 79
Label1.Text = "B"
Case 80 To 100
Label1.Text = "A"
Case Else
Label1.Text = "Error, please reenter the mark"
End Select
End Sub
```

The output of example2 shown the following figure:

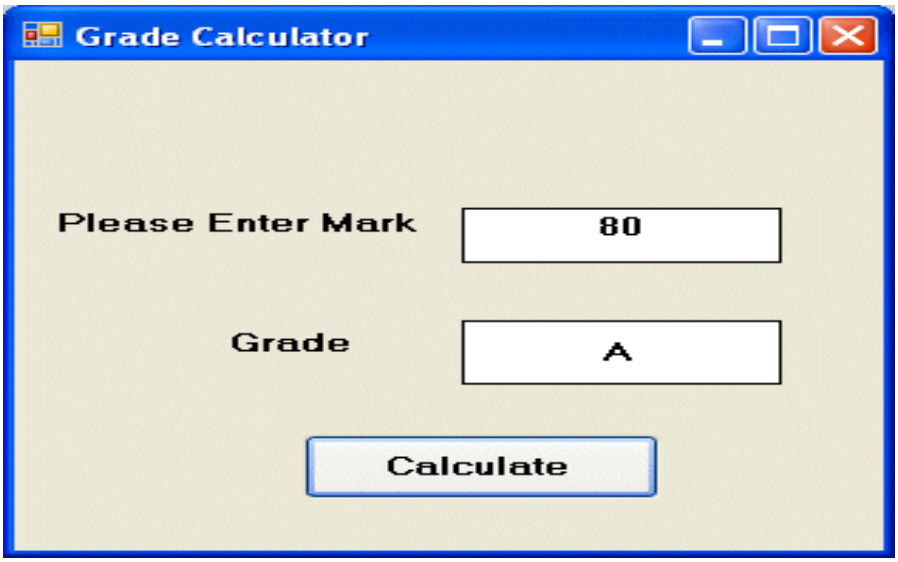


Figure (3.1)

Choose function

Selects and returns a value from a list of arguments.

*__choose__(parameter,val1,val2,val3……),parameter=10 this mean if parameter=5 then will choose val5

X=choose(parameter,val1,val2,val3,val4,…….,val5) if parameter=2 that mean x=val2

Example3: Explain how to use choose function.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim x As Integer
    x = Choose(Val(TextBox1.Text), 4, 6, 2, 7)
    Label1.Text = x
  End Sub
End Class
```

LOOPIN

Looping is required when we need to process something repetitively until a certain condition is met. For example, we can design a program that adds a series of numbers until the sum exceeds a certain value, or a program that asks the user to enter data repeatedly until he/she keys in the word 'Finish'. In Visual Basic 2010, we have three types of Loops, they are the For.....Next loop, the Do loop. and the While.....End while loop.

For....Next Loop

The format is: For counter=startNumber to endNumber (Step increment)

```
One or more VB statements
Next
```

Example4: simple example that use for loop to get summation of numbers.

```
Dim counter , sum As Integer
For counter=1  to 100 step 10
sum+=counter
 Next
Label1.text=sum
```

* The program will calculate the sum of the numbers as follows:

sum=0+10+20+30+40+.....

Sometimes the user might want to get out from the loop before the whole repetitive process is executed; the command to use is Exit For. To exit a For.....Next Loop, you can place the Exit For statement within the loop; and it is normally used together with the If....Then.... statement. For its application, you can refer to example

```
Dim n as Integer

For n=1 to 10
 If n>6 then
 Exit For
 End If
```

```
Else
n+=n
Next
End If
Next
```

The process will stop when n is greater than 6.

DO….LOOP

The formats of Do Loop are:

a) Do While condition

```
    Block of one or more VB statements
Loop
```

b) Do

```
    Block of one or more VB statements
Loop While condition
```

c) Do Until condition

```
    Block of one or more VB statements
Loop
```

d) Do

```
    Block of one or more VB statements
Loop Until condition
```

*** Exiting the Loop**

Sometime we need exit to exit a loop prematurely because of a certain condition is fulfilled. The syntax to use is known as Exit Do. Let's examine the following examples:

Example5: Do Loop example explanation

```
Do while counter <=1000

    TextBox1.Text=counter

    counter +=1
Loop
```

* The above example will keep on adding until counter >1000.

The above example can be rewritten as:

```
Do
    TextBox1.Text=counter

    counter+=1

Loop until counter>1000
```

While ...End While Loop
The loop will end when the condition is met.

Example6: While…..end While explanation

```
Dim sum, n As Integer
While n <> 100
n += 1
sum = sum + n
End While
```

Example7: (For Loop +if) example to calculate the factorial of any number.

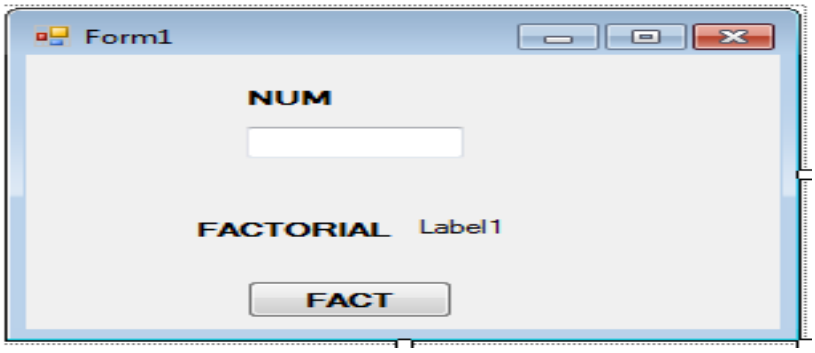


Figure (3.2)

```
Dim y, z, r As Integer
    y = 1 : z = Val(TextBox1.Text)
    If z = 0 Then
       y = y
    Else
       For r = z To 1 Step -1
          y = y * r
       Next r
    End If
    Label1.Text = y
```

Example8: (If-Then) example to implement the following equation:

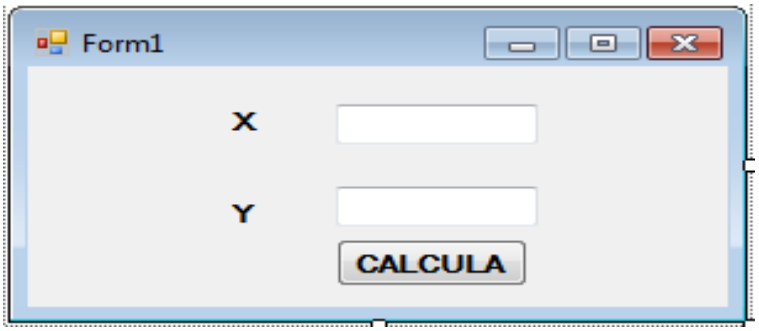


Figure (3.3)

Y=x^2-3*x x>0 OR Y=2*x+x^3 x=0 OR Y=sqr(100+3*x) x<0

```
Dim x As Integer

Dim y As Single

x = Val(TextBox1.Text)

If x > 0 Then

y = x ^ 2 - 3 * x

ElseIf x = 0 Then

y = 2 * x + x ^ 3

Else

y = Math.Sqrt(100+3*x)

End If

TextBox2.Text = y
```

ARRAYS

Declaring a procedure-level one-dimensional array

Syntax—Version 1

Dim *arrayName***(***highestSubscript***) As** *dataType*

Syntax—Version 2

Dim *arrayName***() As** *dataType* **= {***initialValues***}**

Example 9—Version 1's syntax

Dim intNumbers(5) As Integer
declares and initializes (to 0) a six-element Integer array named intNumbers

Example 10—Version 2's syntax

Dim strFriends() As String = {"Sue", "Bob", "John", "Mary"}
declares and initializes a four-element String array named strFriends

Array.Sort and Array.Reverse methods

Syntax

Array.Sort(*arrayName***)**
Array.Reverse(*arrayName***)**

Example 11

Dim intNums() As Integer = {1, 3, 10, 2, 4, 23}
Array.Sort(intNums)
sorts the values in the intNums array in ascending order as follows: 1, 2, 3, 4, 10, 23

Example 12

Dim intNums() As Integer = {1, 3, 10, 2, 4, 23}
Array.Reverse(intNums)
reverses the order of the values in the intNums array as follows: 23, 4, 2, 10, 3, 1

Example 13

Dim intNums() As Integer = {1, 3, 10, 2, 4, 23}
Array.Sort(intNums)
Array.Reverse(intNums)

Sorts the values in the intNums array in descending order as follows: 23, 10, 4, 3, 2, 1.

Example14: will store four names in a one-dimensional String array named strFriends. It then will display the contents of the array in three label controls named lblOriginal, lblAscending, and lblDescending. In the lblOriginal control, the names will appear in the same order as in the array. In the lblAscending and lblDescending controls, the names will appear in ascending and descending order, respectively.

```
Private Sub btnDisplay_Click_1(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnDisplay.Click

    ' displays names in original, ascending,
    ' and descending order
    Dim strFriends() As String =
    {"Sue", "Bob", "John", "Mary"}
    ' clear label controls
    lblOriginal.Text = String.Empty
    lblAscending.Text = String.Empty
    lblDescending.Text = String.Empty
    ' display array in original order
    For intSub As Integer = 0 To strFriends.Length - 1
       lblOriginal.Text = lblOriginal.Text &
       strFriends(intSub) & ControlChars.NewLine
    Next intSub
    ' display array in ascending order
    Array.Sort(strFriends)
    For intSub As Integer = 0 To strFriends.Length - 1
       lblAscending.Text = lblAscending.Text &
       strFriends(intSub) & ControlChars.NewLine
    Next intSub
    ' display array in descending order
    Array.Sort(strFriends)
    Array.Reverse(strFriends)
    For intSub As Integer = 0 To strFriends.Length - 1
       lblDescending.Text = lblDescending.Text &
       strFriends(intSub) & ControlChars.NewLine
    Next intSub
End Sub
```

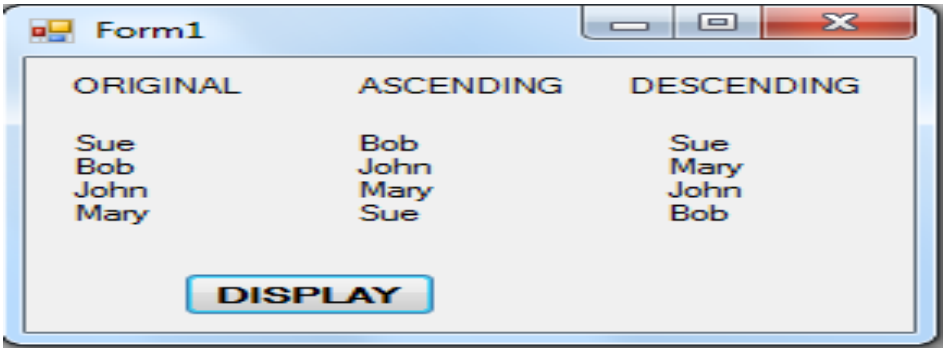


Figure (3.4)

DYNAMIC ARRAY

An array whose number of elements changes while an application is running is referred to as a dynamic array. The ReDim statement allows you to make an array either larger or smaller; however, in most cases you will use it to increase the size of an array

Redim statement

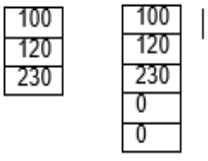
Syntax

ReDim [**Preserve**] *arrayName***(***highestSubscript***)**

Example 15

```
Dim intNums() As Integer = {100, 120, 230}
ReDim Preserve intNums(4)
```

Result of Dim stateme nt:



Example 16

```
Dim intNums() As Integer = {100, 120, 230}
ReDim intNums(4)
```

Result of ReDim statement:

100
120
230

0
0
0
0
0

Example 17

```
Dim intNums() As Integer = {100, 120, 230}
ReDim intNums(1)
```

Result of Dim statement

100
120

100
120
230

An EMPTY array is an array that contains no elements. You declare an empty array using an empty set of braces, like this:

```
Dim decSales() As Decimal = {}
ReDim Preserve decSales(intSub)
```

TWO DIMENTIONAL ARRAY

A two-dimensional array, on the other hand, resembles a table in that the variables (elements) are in rows and columns

Declaring a two-dimensional array

Syntax—Version 1

{**Dim** | **Private**} *arrayName***(***highestRowSubscript***,** *highestColumnSubscript***)** **As** *datatype*

Syntax—Version 2

{**Dim**|**Private**}*arrayName***(,)** **As** *dataType* **=** **{{***initialValues***},**…**{***initialValues***}}**

Example 18—Version 1's syntax

```
Dim intScores(5, 3) As Integer
```

Declares and initializes (to 0) a procedure-level array named intScores; the array has six rows and four columns

Example 19—Version 2's syntax

```
Private strProducts(,) As String =
{{"AC34", "Shirt", "Red"},
{"BD12", "Coat", "Blue"},
{"CP14", "Blouse", "White"}}
```

Declares and initializes a class-level array named strProducts; the array has three rows and three columns.

Example20:assigns the number 0 to each element in the six-row, four-column intNumbers array

```
Private intNumbers(5, 3) As Integer
For intRow As Integer = 0 To 5
For intColumn As Integer = 0 To 3
intNumbers(intRow, intColumn) = 0
Next intColumn
Next intRow
```

Procedure

Write VB program to count number of retired, young and new employees, then print the results on the form. Note: enter the ages of the employees in the text box and make the multiline property of the textbox true.

Discussion

1- Repeat example8 (if then) by using RND function to enter X values.
2- Write VB program to enter 10 materials for student and count the number of excellent,verygood,good,middle,accept, and weak degrees, also calculate the average and the level according to this average then print the results on the form.(hint:can use for loop with select case).

3- Repeat H.W2 but without specify (limit) the number of materials(hint:make use of Redim statement and Len function).

Experiment No. (4)

Form – TextBox – Menus

Object

Allow the user to understand the properties, events and methods of form and menus.

Theory

Forms

The terms *form* and *window* describe the same entity. A window is what the user sees on the Desktop when the application is running. A form is the same entity at design time. The proper term is *Windows form*. Forms have a built-in functionality that is always available without any programming effort on your part.

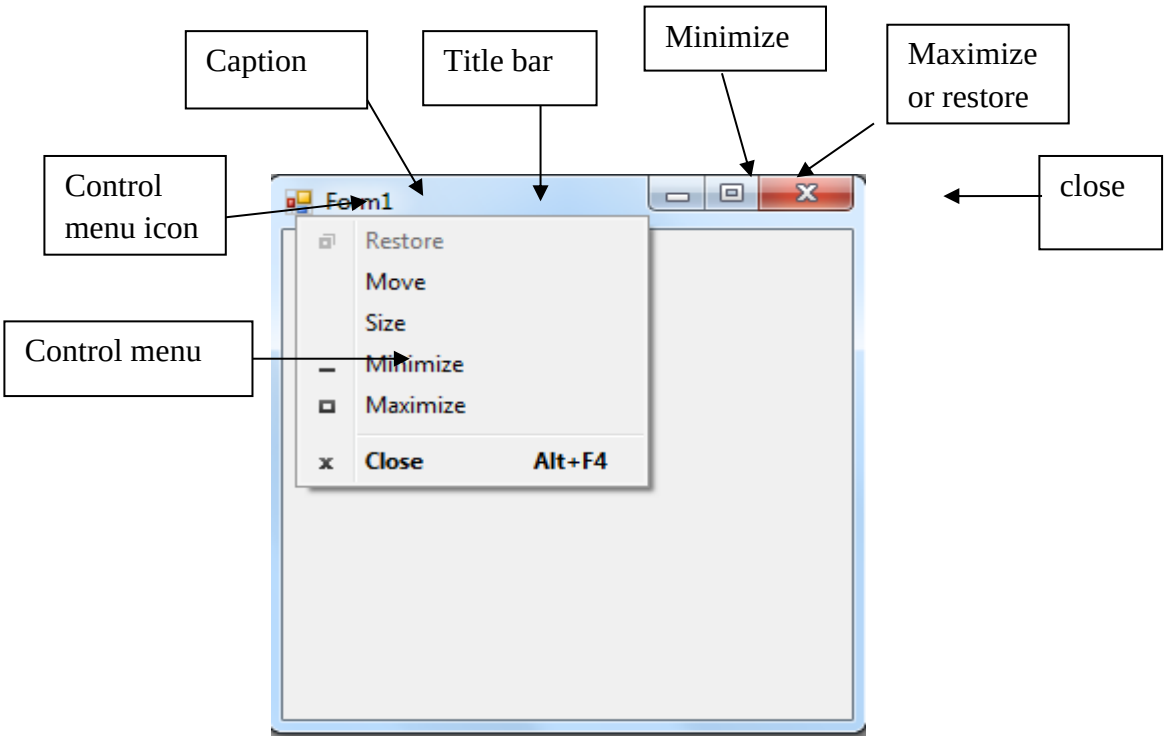


Figure (4.1)

Properties of the Form Object

AcceptButton, CancelButton
These two properties let you specify the default Accept and Cancel buttons. The Accept button is the one that's automatically activated when you press Enter (the OK caption), the Cancel button is the one that's automatically activated when you hit the Esc key.

AutoScroll
The AutoScroll property is a True/False value, will be automatically attached to the form if the form is resized to a point that not all its controls are visible.

AutoScrollPosition
This property is available from within your code only, and it indicates the number of pixels that the form was scrolled up or down. Its initial value is zero, and it takes on a value when the user scrolls the form (provided that the AutoScroll property is True). Use this property to find out the visible controls from within your code or to scroll the form from within your application's code to bring a specific control into view.

FormBorderStyle
The FormBorderStyle property determines the style of the form's border.

MinimizeBox, MaximizeBox
These two properties, which specify whether the Minimize and Maximize buttons will appear

MinimumSize, MaximumSize
These two properties read or set the minimum and maximum size of a form

KeyPreview
This property enables the form to capture all keystrokes before they're passed to the control that has the focus. Some forms perform certain actions when you hit a specific key (the F5 key for refreshing the form being a very common example, To handle a specific keystroke at the

form's level, set the form's KeyPreview property to True and insert the appropriate code in the form's KeyDown or KeyUp event handler, The same keystrokes are then passed to the control with the focus, unless you kill the keystroke by setting its SuppressKeystroke property to True when you process it on the form's level.

Example1:

```
If Char.IsLetterOrDigit(e.KeyChar) Then
    Select Case UCase(e.KeyChar)
      Case "1", "2", "3", "4", "5", "6", "7", "8", "9", "0"
        TextBox1.SelectedText = e.KeyChar
      Case "A", "B", "C", "D", "E", "F"
        TextBox1.SelectedText = UCase(e.KeyChar)
    End Select
    e.Handled = True
  End If
```

Handling Keystrokes

Although the Tab key is the Windows method of moving to the next control on the form, most users will find it more convenient to use the Enter key for that purpose. The Enter key is the most important one on the keyboard. When the user presses Enter in a single-line TextBox, for example, the obvious action is to move the focus to the following control. I included a few statements in the KeyDown event handlers of the TextBox controls to move the focus to the following one:

```
Private Sub txtAddress1_KeyDown(…) Handles txtAddress1.KeyDown
If e.KeyData = Keys.Enter Then
e.SuppressKeyPress = True
txtAddress2.Focus()
End If
End Sub
```

Processing Keys from within Your Code

The code shown in the preceding KeyDown event handler will work, but you must repeat it for every TextBox control on the form. A more convenient approach is to capture the Enter keystroke in the form's KeyDown event handler and process it for all TextBox controls. First, you must figure out whether the control with the focus is a TextBox control. The property Me.ActiveControl returns a reference to the control with the focus.

```
If Me.ActiveControl.GetType Is GetType(TextBox) Then
```

```
' process the Enter key
End If
```

Once you can figure out the active control's type, you need a method of simulating the Tab keystroke from within your code so you don't have to code every TextBox control's KeyDown event. An interesting method of the Form object is the ProcessTabKey method, which imitates the Tab keystroke. Calling the ProcessTabKey method is equivalent to pressing the Tab key from within your code. The method accepts a True/False value as an argument, which indicates whether it will move the focus to the next control in the tab order (if True) or to the previous control in the tab order. Start by setting the form's KeyPreview property to True and then insert the following statements in the form's KeyDown event handler:

```
If e.KeyCode = Keys.Enter Then
If Me.ActiveControl.GetType Is GetType(TextBox) Then
e.SuppressKeyPress = True
If e.Shift Then
Me.ProcessTabKey(False)
Else
Me.ProcessTabKey(True)
End If
End If
End If
```

StartPosition, Location

The StartPosition property, which determines the initial position of the form when it's first displayed

TopMost

This property is a True/False setting that lets you specify whether the form will remain on top of all other forms in your application

Form Events

The *Activated* and *Deactivate* Events

When more than one form is displayed, the user can switch from one to the other by using the mouse or by pressing Alt+Tab. Each time a form is activated, the Activated event takes place. Likewise, when a form is activated, the previously active form receives the Deactivate event. Insert the code you want to execute when a form is activated

The FormClosing and FormClosed Events

The FormClosing event is fired when the user closes the form by clicking its Close button. For example, you might display a warning if the user has unsaved data, you might have to update a database, and so on.

The Resize, ResizeBegin, and ResizeEnd Events: the Resize event is fired every time the user resizes the form by using the mouse.

The *Scroll* Event
The Scroll event is fired by forms that have the AutoScroll property set to True when the user scrolls the form.

The Paint Event
This event takes place every time the form must be refreshed, and we use its handler to execute code for any custom drawing on the form. When you switch to another form that partially or totally overlaps the current one and then switch back to the first form, the Paint event will be fired to notify your application that it must redraw the form.

Loading and Showing Forms
Most practical applications are made up of multiple forms and dialog boxes. you might want to maintain two forms open at once and let the user switch between them. A text editor and its Find &Replace dialog box is a typical example. To show *Form2* when an action takes place on *Form1*, call the Show method of the auxiliary form:

```
Form2.Show
```

A dialog box is simply a modal form. When you display forms as dialog boxes, change the border of the forms to the setting *FixedDialog* and invoke them with the ShowDialog method. Modeless forms are more difficult to program because the user may switch among them at any time. Moreover, the two forms that are open at once must interact with one another. When the user acts on one of the forms, it might necessitate changes in the other, and you'll see shortly how this is done. If the two active forms don't need to interact, display one of them as a dialog box.
When you're finished with the second form, you can either close it by calling its Close method or hide it by calling its Hide method. The Close method closes the form, and its resources are returned to the system. The Hide method sets the form's Visible property to False; you can still access a hidden form's controls from within your code, but the user can't interact with it.

The Startup Form

A typical application has more than a single form. When an application starts, the main form is loaded. You can control which form is initially loaded by setting the startup object in the project Properties window. To open this dialog box, right-click the project's name in the Solution Explorer and select Properties. In the project's Properties pages, switch to the Application tab and select the appropriate item in the Startup Form combo box. By default, the IDE suggests the name of the first form it created, which is *Form1*.

Sharing Variables between Forms

Variables are declared in the form's declarations section, outside any procedure, with the keyword Public

```
Public NumPoints As Integer
Public DataValues(100) As Double
```

To access a public variable declared in Form1 from within another form's code

```
Form1.NumPoints = 99
Form1.DataValues(0) = 0.339502
Form1.TextBox1.Text
```

The controls on a form can be accessed by the code in another form because the default value of the Modifiers property of the controls on a form is *Friend*, which means that all components in a solution can access them. Other settings of the Modifiers property are *Public* (any application can access the control) and *Private* (the control is private to the form to which it belongs and cannot be accessed from code outside its own form).

```
Form2.TextBox1.Text = "some text"
Form2.Show()
```

The TextBox Control

The TextBox control is the primary mechanism for displaying and entering text.

Basic Properties

MultiLine

This property determines whether the TextBox control will hold a single line or multiple lines of text.

MaxLength

This property determines the number of characters that the TextBox control will accept.

ScrollBars
This property lets you specify the scroll bars

CharacterCasing
This property tells the control to change the casing of the characters

PasswordChar
Hides text with single char.

ReadOnly, *Locked*
If you want to display text on a TextBox control but prevent users from editing it (such as for an agreement or a contract they must read, software installation instructions, and so on), you can set the ReadOnly property to True.

Text
The most important property, which holds the control's text.

Text-Selection Properties
The TextBox control provides three properties for manipulating the text selected by the user:

SelectedText, **SelectionStart**, and **SelectionLength**. Users can select a range of text with a
click-and-drag operation and the selected text will appear in reverse color

SelectedText
This property returns the selected text
SelectionStart, SelectionLength
The SelectionStart property returns or sets the position of the first character of the selected text
The SelectionLength property returns or sets the length of the selected text.

e.g. text=EXAMPLE , text name=textbox1


```
Dim x As String
    TextBox1.SelectionStart = 2
    TextBox1.SelectionLength = 3
```

```
x = TextBox1.SelectedText
MsgBox(x)
```

TAG (hidden text with in text box).

Events

*change (triggered every time the text property changes)

*lostfocus (triggered when user leaves the text box).

*keypress (triggered when key is pressed)

Methods

*setfocus (place the cursor in a specified text box , give it the focus).

e.g. txt.setfocus (this will move cursor to the box named **txt**).

Example2 to explain how to dealing with password programs by using the textbox properties

Also explain how to make use of forms events when using more than one form in your project First form1 to enter valid password, then second form2 is (MODAL Form) to enter two numbers to add them and show the result in the third form3 by using CLICK event of the form3 and if user forget to click on the form and decide to close form3, then we shall make use of event CLOSE of form3 to remember user to before (he/she) leaving.

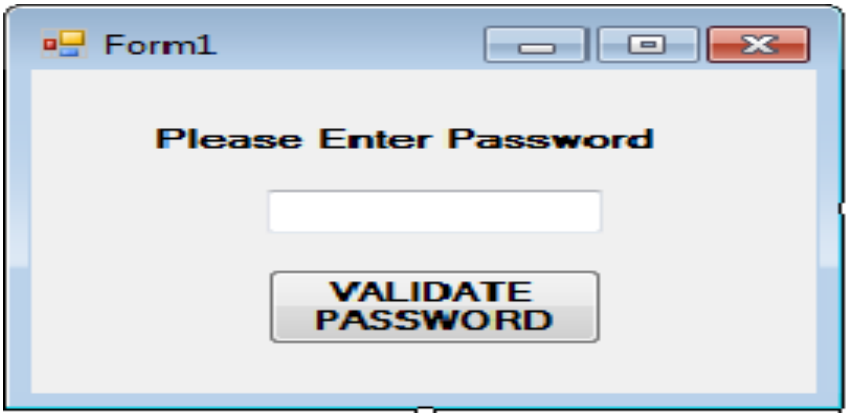


Figure 4.2

Here is the code for the command button:

```
Dim pasword As String
    If TextBox1.Text = "" Then
       MsgBox("please enter the password")
    Else
       TextBox1.SelectionStart = 0
       TextBox1.SelectionLength = 7
       pasword = TextBox1.SelectedText
       TextBox1.Tag = "example"
       If pasword = TextBox1.Tag Then
          MsgBox("true password")
          Form2.ShowDialog()
       Else
          MsgBox("invalid password")
       End If
    End If
```

And another code for the **change event** of the textbox control**.**

```
TextBox1.PasswordChar = "*"
```

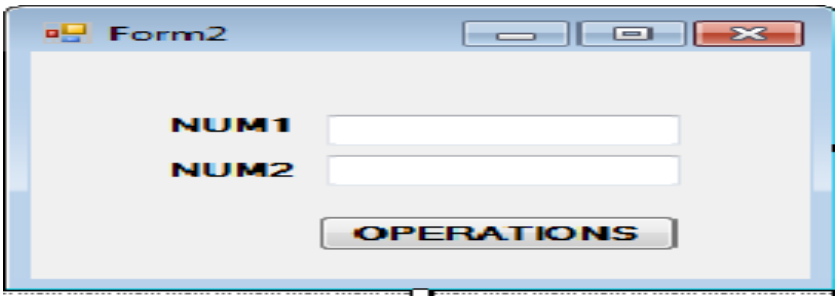


Figure 4.3

Here is the code for command button for form2:

```
If TextBox1.Text <> "" And TextBox2.Text <> "" Then
       Form3.Show()
    End If
```

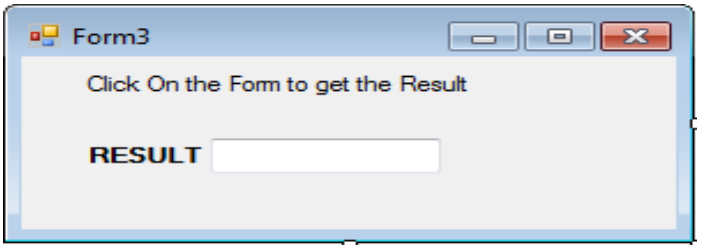


Figure (4.4)

The code for **CLICK** event of form3, and **CLOSE** event of form3:

```
Private Sub Form3_Click
      Me.TextBox1.Text = Val(Form2.TextBox1.Text) +
Val(Form2.TextBox2.Text)
   End Sub

Private Sub Form3_FormClosing
      If TextBox1.Text = "" Then
          MsgBox("remember that you don't get the result")
      End If
```

The Menu Editor

Menus can be attached only to forms, and they're implemented through the MenuStrip control which contain ToolStripMenuItem objects. You can design menus visually and then program their Click event handlers. Double-click the MenuStrip icon in the Toolbox, then you can create any menu you want such as (FILE, EDIT, FORMAT….).

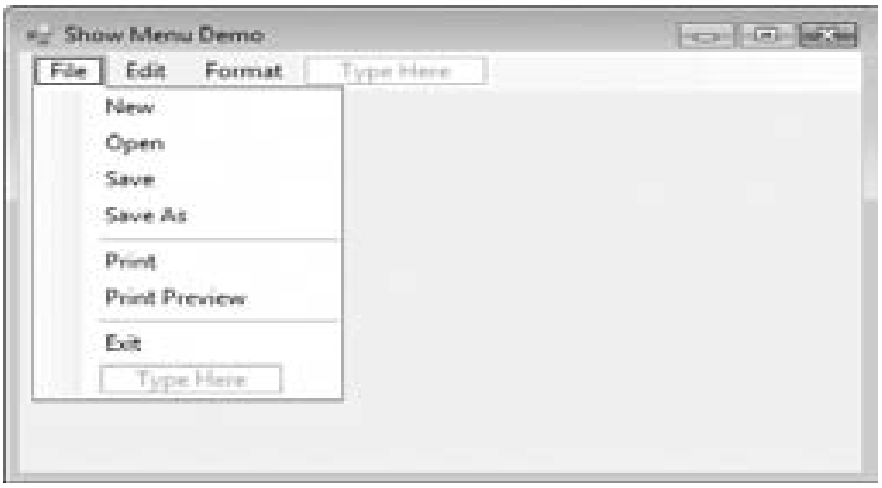


Figure (4.5)

Can change or set any property of menustrip or any of menuitems by the properties window. NOTE: The most convenient method of editing a menu is to use the Items Collection Editor window (you can set all the properties of each menu item in the dialog box without having to switch to the Properties window). Right click on menustrip then choose Edit Items…. , finally the Items Collection Editor window will opened.

Right click on menustrip then choose **Insert Standard Items** will add the menus(File, Edit, Tools, Help) in your form.

The ToolStripMenuItem Properties

Enabled Some menu commands aren't always available. The Paste command, for example, has no meaning if the Clipboard is empty (or if it contains data that can't be pasted in the current application).

Programming Menu Commands

When a menu item is selected by the user, it triggers a Click event. The Exit command's code would be something like the following:

```
Sub menuExit(…) Handles menuExit.Click
End
End Sub
```

If you need to execute any cleanup code before the application ends, place it in the CleanUp() subroutine and call this subroutine from within the Exit item's Click event handler:

```
Sub menuExit(…) Handles menuExit.Click
CleanUp()
End
End Sub
```

NOTE: The same subroutine must also be called from within the FormClosing event handler of the application's main form because some users might terminate the application by clicking the form's Close button.

DropDownOpened event

(In effect, when the user clicks a menu item that leads to a submenu). Edit menu of just about any application contains the ubiquitous Cut/Copy/Paste commands. These commands are not meaningful at all times. If the Clipboard doesn't contain text, the Paste command should be disabled. If no text is selected, the Copy and Cut commands should also be disabled. Here's how you could change the status of the Paste command from within the DropDownOpened event handler of the Edit menu:

```
If My.Computer.Clipboard.ContainsText Then
PasteToolStripMenuItem.Enabled = True
Else
PasteToolStripMenuItem.Enabled = false
End If
```

Likewise, to change the status of the Cut and Copy commands, use the following statements in the DropDownOpened event of the ToolStripMenuItem that represents the Edit menu:

```
If txtEditor.SelectedText.Trim.Length > 0 Then
CopyToolStripMenuItem.Enabled = True
CutToolStripMenuItem.Enabled = True
Else
CopyToolStripMenuItem.Enabled = False
CutToolStripMenuItem.Enabled = False
End If
```

Access Keys

Access keys allow the user to open a menu by pressing the Alt key and a letter key (for example: Alt+E to open Edit menu). To assign an access key to a menu item, insert the ampersand symbol (&)in front of the character you want to use as an access key in the ToolStripMenuItem's Text property.

Creating Context Menus

Nearly every Windows application provides a context menu that the user can invoke by right-clicking a form or a control. (It's sometimes called a shortcut menu or pop-up menu.) This is a regular menu, but it's not anchored on the form. It can be displayed anywhere on the form or on specific controls. Different controls can have different context menus, depending on the operations you can perform on them at the time.To create a context menu, place a ContextMenuStrip control on your form. The new context menu will appear on the form just like a regular menu, but it won't be displayed there at runtime. You can create as many context menus as you need by placing multiple instances of the ContextMenuStrip control on your form and adding the appropriate commands to each one. To associate a context menu with a control on your form, set the ContextMenu property for that control to the name of the corresponding context menu.

Procedure

Repeat the above (example2) but with different FORM3 EVENTS. And add more mathematics operations.

Discussion

1- WriteV.B.NET program to implement the following menus (**GENERAL**_OPERATIONS_CircleArea and CircleParameter), **EXIT**_Close.

2- Write V.B.NET program to implement Password login that give user three trials to enter right password, otherwise the program ended.

Hint: use Tag and Passwordchar properties of TextBox control. And static variable to count the user trials.

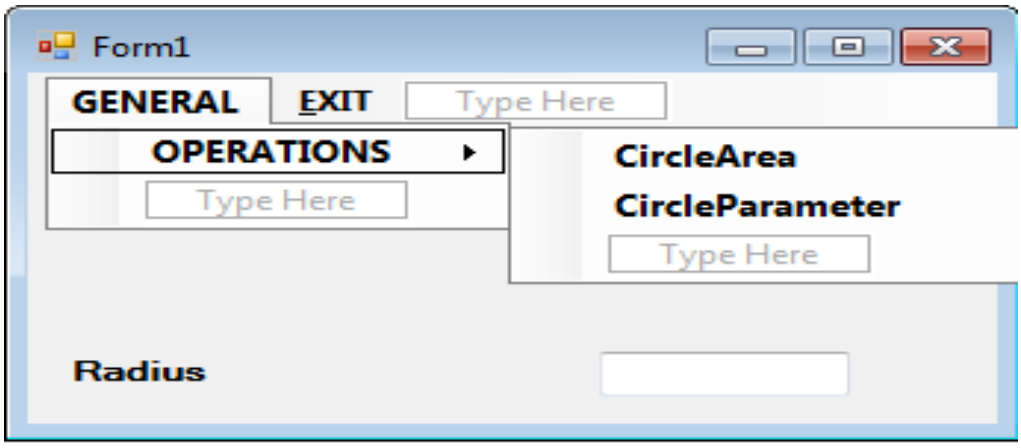


Figure (4.6)

Experiment No. (5)

MessageBox – InputBox - DialogueBox

Object

To explain how the user can communicate with an application during the run time by using MessageBox – InputBox – DialogueBox.

Theory

MessageBox

At times, an application may need to communicate with the user during runtime; one means of doing this is through a message box. MessageBox is one of those dialog boxes that you will use often as a developer.

MsgBox (Prompt [,icons+buttons+default button+modality] [,title] **)**

Table (5.1)

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

Table (5.2)

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

Which button is default mean which one be active with pressing enterkey.

Table (5.3)

Value	Meaning	Symbolic constant
0	First button	VbDefaultbutton1
256	Second button	VbDefaultbutton2
512	Third button	VbDefaultbutton3

Modality:either application modal=0 VbApplicationModal,or system modal=4096 VbSystemModal.

NOTE: you can use MSGBOX method or MessageBox.Show method.

The MessageBox.Show Method

You can display a messagebox using the MessageBox.Show method.

Syntax
MessageBox.Show(*text*, *caption*, *buttons*, *icon*[, *defaultButton*])

NOTE:You can specify the Show method in several ways; the more common syntaxes are shown in the following list:

- MessageBox.Show(*message text*)
- MessageBox.Show(*message text*, *caption*)
- MessageBox.Show(*message text*, *caption*, *buttons*)
- MessageBox.Show(*message text*, *caption*, *buttons*, *icon*)
- MessageBox.Show(*message text*, *caption*, *buttons*, *icon*, *default button*)

Argument Meaning
text text to display in the message box; use sentence capitalization
caption text to display in the message box's title bar; use book title capitalization
buttons: buttons to display in the message box; can be one of the following
constants:

MessageBoxButtons.AbortRetryIgnore

MessageBoxButtons.OK (default setting)
MessageBoxButtons.OKCancel
MessageBoxButtons.RetryCancel
MessageBoxButtons.YesNo
MessageBoxButtons.YesNoCancel

***icon* icon to display in the message box; typically, one of the following constants:**

MessageBoxIcon.Exclamation 

MessageBoxIcon.Information 

MessageBoxIcon.Question 

MessageBoxIcon.Stop 

defaultButton button automatically selected when the user presses Enter; can be one of the following constants:

MessageBoxDefaultButton.Button1 (default setting)
MessageBoxDefaultButton.Button2
MessageBoxDefaultButton.Button3

Example 1

```
MessageBox.Show("Record deleted.", "Payroll",
MessageBoxButtons.OK, MessageBoxIcon.Information)
```

Displays an information message box that contains the message "Record deleted."

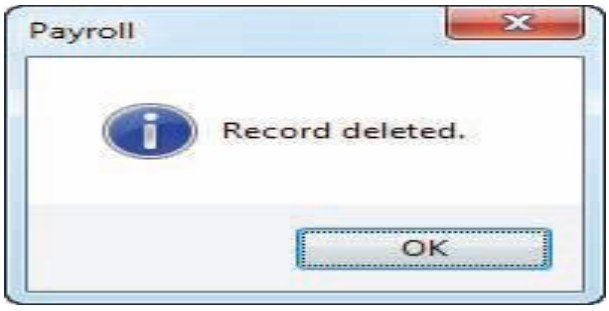


Figure (5.1)

Example 2

```
MessageBox.Show("Delete this record?", "Payroll",
```

```
MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation,
MessageBoxDefaultButton.Button2)
```

Displays a warning message box that contains the message “Delete this record?”

MessageBox.Show method’s return values

Integer	DialogResult value	Meaning
1	DialogResult.OK	user chose the OK button
2	DialogResult.Cancel	user chose the Cancel button
3	DialogResult.Abort	user chose the Abort button
4	DialogResult.Retry	user chose the Retry button
5	DialogResult.Ignore	user chose the Ignore button
6	DialogResult.Yes	user chose the Yes button
7	DialogResult.No	user chose the No button

Example 3

```
Dim dlgButton As DialogResult
dlgButton =MessageBox.Show("Delete this record?", "Payroll",
MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation,
MessageBoxDefaultButton.Button2)
If dlgButton = DialogResult.Yes Then
instructions to delete the record
End If
```

Example4: this is a simple example(using MsgBox method) to explain the return value(integer) of the VbOkCancel message box which will return number 1 if the user click ok button, if user click cancel the returned value is 2.

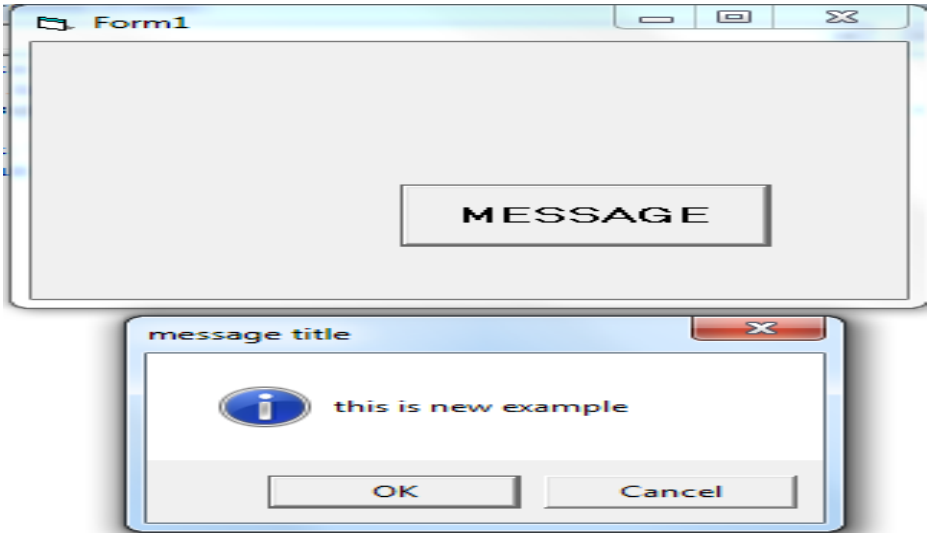


Figure (5.2)

```
Dim r As Integer
    r = MsgBox("this is new example", vbOKCancel + vbInformation +
vbDefaultButton1 + _
        vbApplicationModal, "message title")
    MsgBox(r)
```

The same example:

```
Dim r As Integer
    r = MsgBox("this is new example", 1 + 64 + 256 + _
        0, "message title")
    MsgBox(r)
```

Displaying Dialog Boxes

Visual Basic 2010 provides several built-in dialog boxes that help you provide a rich user interface in your front-end applications. These dialog boxes provide the same common user interface that is found in most Windows applications.

- ➢ Creating an Open dialog box that enables you to open files
- ➢ Creating a Save dialog box that enables you to save files
- ➢ Creating a Font dialog box that enables you to apply the selected font to text
- ➢ Creating a Color dialog box that enables you to define and select custom colors
- ➢ Creating a Print dialog box that prints text from your application
- ➢ Creating a Browse dialog box that enables you to browse for folders

THE OPENFILEDIALOG CONTROL

Many Windows applications process data from files, so you need an interface to select files to open and save. The .NET Framework provides the OpenFileDialog and SaveFileDialog classes to do just that. You can use OpenFileDialog as a .NET class by declaring a variable of that type in your code and modifying its properties in code without dragging a control onto the Forms Designer., or as a control by dragging the control from the Toolbox onto the form at design time.
Example: Public Class Dialogs

Common OpenFileDialog Control Properties

Table (5.4)

PROPERTY	DESCRIPTION
AddExtension	Indicates whether an extension is automatically added to a filename if the user omits the extension. This is mainly used in the SaveFileDialog, described in the next section.
AutoUpgradeEnabled	Indicates whether this dialog should automatically upgrade its appearance and behavior when running on different versions of Windows. When false, it will appear with XP styles.
CheckFileExists	Indicates whether the dialog box displays a warning if the user specifies a filename that does not exist.
CheckPathExists	Indicates whether the dialog box displays a warning if the user specifies path that does not exist.
DefaultExt	Indicates the default filename extension.
DereferenceLinks	. Indicates whether the dialog returns the location of the file referenced by the Used with shortcuts shortcut (True) or whether it returns only the location of the shortcut itself (False).
FileName	Indicates the path and filename of the selected file in the dialog box.
FileNames	Indicates the path and filenames of all selected files in the dialog box. Thisis a read-only property.
Filter	Indicates the current filename filter string, which determines the options that appear in the Files of Type: combo box in the dialog

FilterIndex	Indicates the index of the filter currently selected in the dialog box.
InitialDirectory	Indicates the initial directory displayed in the dialog box.
Multiselect	Indicates whether the dialog box allows multiple files to be selected.
ReadOnlyChecked	Indicates whether the read-only check box is selected.
SafeFileName	Indicates the filename of the selected file in the dialog box.

OpenFileDialog Methods

- ➤ OpenFile opens the file selected by the user with read-only permission. The file is specified by the FileName property.
- ➤ Reset resets all properties of the Open dialog box to their default values.
- ➤ ShowDialog shows the dialog box.

Example5: put this code in command button , also put openfiledialoge control and textbox on the form, then run it.

```
'Set the Open dialog properties
    Dim strFileName As String
    With OpenFileDialog1
       .Filter = "Text Documents (*.txt)|*.txt|All Files (*.*)|*.*"
       .FilterIndex = 1
       .Title = "Demo Open File Dialog"
    End With
    'Show the Open dialog and if the user clicks the Open button,
    'load the file
    If OpenFileDialog1.ShowDialog = Windows.Forms.DialogResult.OK
Then

       strFileName = OpenFileDialog1.FileName
       TextBox1.Text = strFileName
End If
```

THE SAVEDIALOG CONTROL

Now that you can open a file with the OpenFileDialog control, take a look at the SaveFileDialog control so that you can save a file. Like the OpenFileDialog, the SaveFileDialog can be used as a control or a class without dragging a control onto the Forms Designer.. Example: Public Class Dialogs

Common SaveFileDialog Control Properties

Table (5.5)

PROPERTY	DESCRIPTION
AddExtension	Indicates whether an extension is automatically added to a filename if the user omits the extension
AutoUpgradeEnabled	Indicates whether this dialog box should automatically upgrade its Appearance behavior when running on different versions of Windows. When false, it will appear with XP styles.
CheckFileExists	Indicates whether the dialog box displays a warning if the user specifies a filename that does not exist. This is useful when you want the user to save a file to an existing name.
CheckPathExists	Indicates whether the dialog box displays a warning if the user specifies a path that does not exist.
CreatePrompt	Indicates whether the dialog box prompts the user for permission to create a file if the user specifies a file that does not exist.
DefaultExt	Indicates the default file extension
DereferenceLinks	Indicates whether the dialog box returns the location of the file referenced by the shortcut or whether it returns the location of the shortcut itself.
FileName	Indicates the filename of the selected file in the dialog box. This is a readonly property.
FileNames	Indicates the filenames of all selected files in the dialog box. This is a readonly property that is returned as a string array.
Filter	Indicates the current filename filter string, which determines the options that appear in the Files of Type: combo box in the dialog box.
FilterIndex	Indicates the index of the filter currently selected in the dialog box
InitialDirectory	Indicates the initial directory displayed in the dialog box
OverwritePrompt	Indicates whether the dialog box displays a warning if the user specifies a filename that already exists.
ShowHelp	Indicates whether the Help button is displayed in the dialog box
SupportMultiDotted Extensions	Indicates whether the dialog box supports displaying and saving files that have multiple filename extensions
Title	Indicates the title that is displayed in the title bar of the dialog box

ValidateNames	Indicates whether the dialog box should accept only valid Win32 filenames

SaveFileDialog Methods

The SaveFileDialog control exposes the same methods as the OpenFileDialog.

Example6: put this code in command button

```
‘Set the Save dialog properties
With SaveFileDialog1
        .DefaultExt = "txt"
        .FileName = strFileName
        .Filter = "Text Documents (*.txt)|*.txt|All Files (*.*)|*.*"
        .FilterIndex = 1
        .OverwritePrompt = True
        .Title = "Demo Save File Dialog"
        End With
        ‘Show the Save dialog and if the user clicks the Save button,
        ‘save the file
        If SaveFileDialog1.ShowDialog = Windows.Forms.DialogResult.OK Then
         ‘Save the file path and name
        strFileName = SaveFileDialog1.FileName
        msgbox(strFileName)
        End If
```

THE FONT DIALOG CONTROL

Sometimes you may need to write an application that allows users to choose the font in which they want their data to be displayed or entered. can be used as a control or a class without dragging a control onto the Forms.

Common FontDialog Control Properties

Table (5.6)

PROPERTY	DESCRIPTION
AllowScriptChange	Indicates whether the user can change the character set specified in the Script drop-down box to display a character set other than the one currently displayed
Color	Indicates the selected font color
Font	Indicates the selected font
FontMustExist	Indicates whether the dialog box specifies an error condition

	if theuser attempts to enter a font or style that does not exist
MaxSize	Indicates the maximum size (in points) a user can select
MinSize	Indicates the minimum size (in points) a user can select
ShowApply	Indicates whether the dialog box contains an Apply button
ShowColor	Indicates whether the dialog box displays the color choice
ShowEffects	Indicates whether the dialog box contains controls that allow the user to specify strikethrough, underline, and text color options
ShowHelp	Indicates whether the dialog box displays a Help button

E.X.:

```
FontDialog1.ShowDialog()
FontDialog1.ShowColor = True
FontDialog1.ShowDialog()
```

Example7: to change font of string placed in textbox

```
‘Set the Font dialog properties
        FontDialog1.ShowColor = True
‘Show the Font dialog and if the user clicks the OK button,
‘update the font and color in the text box
        If FontDialog1.ShowDialog = Windows.Forms.DialogResult.OK Then
        txtFile.Font = FontDialog1.Font
        txtFile.ForeColor = FontDialog1.Color
        End If
```

THE COLORDIALOG CONTROL

Sometimes you may need to allow users to customize the colors on their form. This may be the color of the form itself, a control, or text in a text box. can be used as a control or a class without dragging a control onto the Forms.

Common ColorDialog Control Properties

Table (5.7)

PROPERTY	**DESCRIPTION**
AllowFullOpen	Indicates whether users can use the dialog box to define custom colors
AnyColor	Indicates whether the dialog box displays all available colors in the set of basic colors
Color	Indicates the color selected by the user
CustomColors	Indicates the set of custom colors shown in the dialog box

FullOpen Indicates whether the controls used to create custom colors are visible when the dialog box is opened

Example8: update the form background color

```
'Show the Color dialog and if the user clicks the OK button,
'update the background color of the form
    If ColorDialog1.ShowDialog = Windows.Forms.DialogResult.OK Then
    Me.BackColor = ColorDialog1.Color
    End If
```

THE PRINTDIALOG CONTROL

Any application worth its salt will incorporate some kind of printing capabilities, whether it is basic printing or more sophisticated printing, such as allowing a user to print only selected text or a range of pages.

Common PrintDialog Control Properties

Table 5.8

PROPERTY	DESCRIPTION
AllowCurrentPage	Indicates whether the Current Page option button is enabled
AllowPrintToFi	Indicates whether the Print to File check box is enabled
AllowSelection	Indicates whether the Selection option button is enabled
AllowSomePages	Indicates whether the Pages option button is enabled
Document	Indicates the print document used to obtain the printer settings
PrinterSettings	Indicates the printer settings that the dialog box will be modifying
PrintToFile	Indicates whether the Print to File check box is checked
ShowHelp	Indicates whether the Help button is displayed
ShowNetwork	Indicates whether the Network button is displayed

THE FOLDERBROWSERDIALOG CONTROL

Occasionally, you'll need to allow your users to select a folder instead of a file. Perhaps your application performs backups, or perhaps you need a folder to save temporary files. The FolderBrowserDialog control displays the Browse For Folder dialog box, which enables users to select a folder.(can also be used as a class declared in code).

Common FolderBrowserDialog Control Properties

Table 5.9

PROPERTY	DESCRIPTION
Description	Provides a descriptive message in the dialog box.
RootFold	Indicates the root folder from which the dialog box should start browsing.
SelectedPath	Indicates the folder selected by the user.
ShowNewFolderButton	Indicates whether the Make New Folder button is shown in the dialog box

Example 9:

```
Private Sub btnBrowse_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles btnBrowse.Click
'Set the FolderBrowser dialog properties
With FolderBrowserDialog1
.Description = "Select a backup folder"
.RootFolder = Environment.SpecialFolder.MyComputer
.ShowNewFolderButton = False
End With
'Show the FolderBrowser dialog and if the user clicks the
'OK button, display the selected folder
If FolderBrowserDialog1.ShowDialog = Windows.Forms.DialogResult.OK
Then
txtFile.Text = FolderBrowserDialog1.SelectedPath
End If
End Sub
```

The InputBox Function

The input dialog box contains a message, an OK button, a Cancel button, and an input area where the user can enter information. The message in the dialog box should prompt the user to enter the appropriate information in the input area. The user closes the dialog box by clicking the OK button, Cancel button, or Close button. The value returned by the InputBox function depends on the button the user chooses. If the user clicks the OK button, the InputBox function returns the value contained in the input area of the dialog box; the return value is always treated as a string. If the user clicks either the Cancel button in the dialog box or the Close button on the dialog box's title bar, the InputBox function returns an empty string. The empty string is represented by the String.Empty constant in Visual Basic.

InputBox Function
Syntax

InputBox(*prompt*[, *title*][, *defaultResponse*])

Example10

```
strName = InputBox("First name:", "Name Entry")
```

Displays an input dialog box that shows First name: as the prompt, Name Entry in the title bar, and an empty input area. When the user closes the dialog box, the assignment statementassigns the user's response to a String variable named strName.

Example11

```
strState = InputBox("State name:", "State", "Alaska")
```

Displays an input dialog box that shows State name: as the prompt, State in the title bar, and Alaska in the input area. When the user closes the dialog box, the assignment statement
assigns the user's response to a String variable named strState.

Example12

```
Const strPROMPT As String =
"Enter a sales amount. Click Cancel to end."

Const strTITLE As String = "Sales Entry"
strInputSales = InputBox(strPROMPT, strTITLE, "0.00")
```

Displays the input dialog box shown in Figures below.When the user closes the dialog box, the assignment statement assigns the user's response to a String variable namedstrInputSales.

Example13

```
Integer.TryParse(InputBox("How old are you?",
"Discount Verification"), intAge)
```

NOTE: can use Val instead of TryParse.

```
intAge=Val(InputBox("How old are you?","Discount Verification"))
```

Displays an input dialog box that shows How old are you? as the prompt, Discount Verification in the title bar, and an empty input area. When the user closes the dialog box, the TryParse method converts the user's response from String to Integer and then stores the result in anInteger variable named intAge.

Procedure

Write AV.B program to keep track of the number of sales amounts entered by the sales manager, uses the accumulator variable to total the sales amounts entered by the user

Discussion

1- Repeat question (2) in discussion of Experement (4). *Hint*:make use of msgbox returnd buttons value, if wrong user must click on vbretry button to renter password, Also use static variable to count the number of user trails.

2- Give the summation of array items. Using inputbox to enter the number of items to be added and to enter the items itself, then put the result in textbox.

Experiment No. (6)

Checkbox, RadioButton(Option button), Frames(Groupbox), ListBox, CheckedListBox,ComboBox, ScrollBars and TrachBar

Object

Allow the user to select one or more items from a list of items.

Theory

Checkbox : It allows the user to select one or more items by checking the checkbox/checkboxes concerned.

Example1: In this example, the user can enter text into a textbox and format the font using the three checkboxes that represent bold, italic and underline.

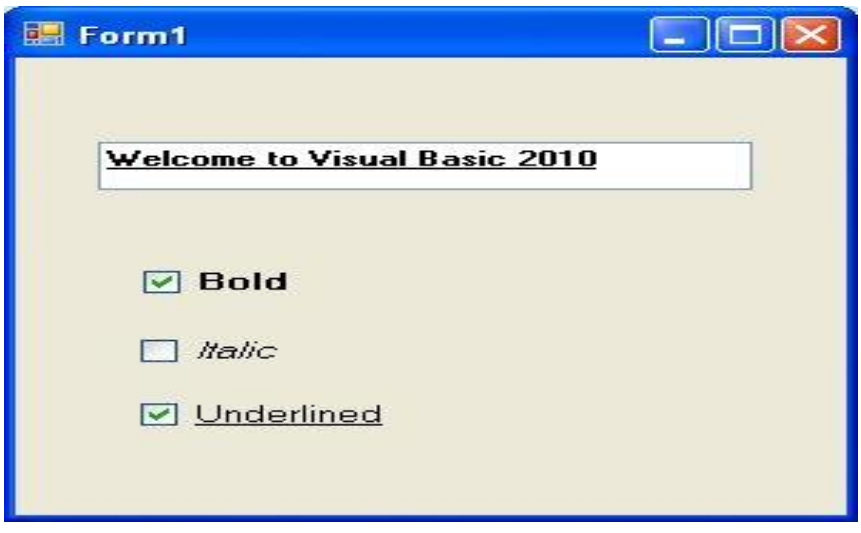


Figure (6.1)

The code is as follow:

```
Private Sub CheckBox1_CheckedChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles CheckBox1.CheckedChanged
If CheckBox1.Checked Then
TextBox1.Font = New Font(TextBox1.Font, TextBox1.Font.Style Or
FontStyle.Bold)
Else
TextBox1.Font = New Font(TextBox1.Font, TextBox1.Font.Style And Not
FontStyle.Bold)
End If
End Sub

Private Sub CheckBox2_CheckedChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles CheckBox2.CheckedChanged
If CheckBox2.Checked Then
TextBox1.Font = New Font(TextBox1.Font, TextBox1.Font.Style Or
FontStyle.Italic)
Else
TextBox1.Font = New Font(TextBox1.Font, TextBox1.Font.Style And Not
FontStyle.Italic)
End If
End Sub

Private Sub CheckBox3_CheckedChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles CheckBox3.CheckedChanged
If CheckBox2.Checked Then
TextBox1.Font = New Font(TextBox1.Font, TextBox1.Font.Style Or
FontStyle.Underline)
Else
TextBox1.Font = New Font(TextBox1.Font, TextBox1.Font.Style And Not
FontStyle.Underline)
```

```
End If
End Sub
```

* The above program uses the CheckedChanged event to respond to the user selection by checking a particular checkbox, it is similar to the click event.

Radio Button

it operates differently from the check boxes. While the checkboxes work independently and allows the user to select one or more items , radio buttons are mutually exclusive, which means the user can only choose one item only out of a number of choices.

Example 2: Here is an example which allows the user to select one color only.

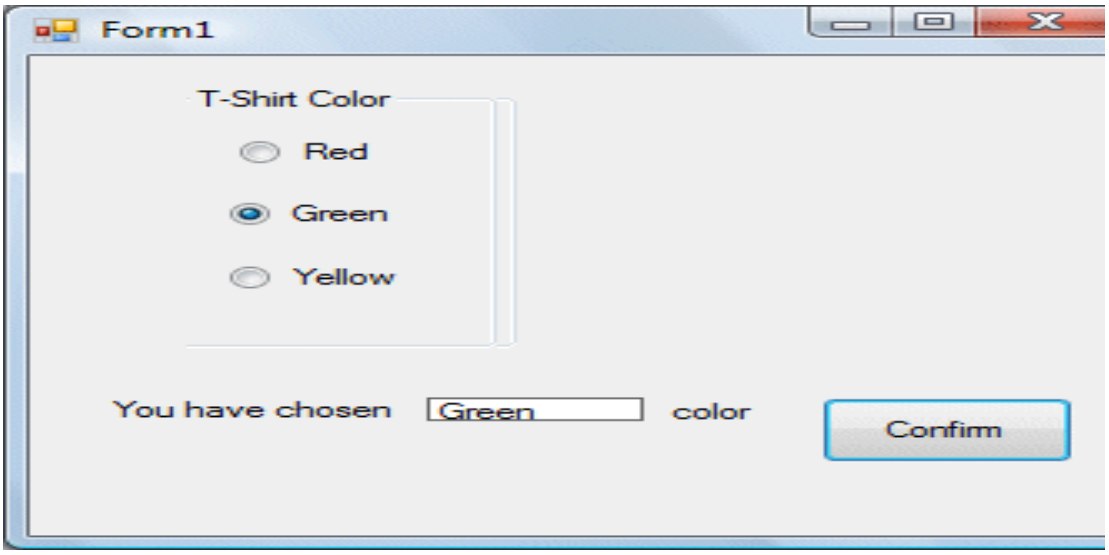


Figure (6.2)

The Code: Dim strColor As String

```
Private Sub RadioButton1_CheckedChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles RadioButton1.CheckedChanged
strColor = "Red"
End Sub

Private Sub RadioButton2_CheckedChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles RadioButton2.CheckedChanged
strColor = "Green"
```

```
End Sub

Private Sub RadioYellow_CheckedChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles RadioYellow.CheckedChanged
strColor = "Yellow"
End Sub

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
Label2.Text = strColor
End Sub
```

Although the user may only select one item at a time, he may make more than one selection if those items belong to different categories. For example, the user wish to choose T-shirt size and color, he needs to select one color and one size, which means one selection in each category. This is easily achieved in VB2010 by using the Groupbox control under the containers categories. After inserting the Groupbox into the form, you can proceed to insert the radio buttons into the Groupbox. Only the radio buttons inside the Groupbox are mutually exclusive, they are not mutually exclusive with the radio buttons outside the Groupbox.

NOTE: The GroupBox tool is located in the Containers section of the toolbox, because a group box serves as a container for other controls. You can use a group box to visually separate related controls from other controls on the form.

The ListBox, CheckedListBox, and ComboBox Controls

The ListBox, CheckedListBox, and ComboBox controls present lists of choices from which the user can select one or more of the items. The ListBox displays a list of choices from which the user can select zero choices, one choice, or multiple choices. Th e number of choices the user can select is controlled by the list box's SelectionMode property. The default value for the property is One, which allows the user to select only one choice at a time.

The CheckedListBox control is a variation of the ListBox control. It's identical to the List- Box control, but a check box appears in front of each item. The user can select any number of items by checking or clearing the boxes. As you know, you can also select multiple items from a ListBox control by pressing the Shift or Ctrl key. The ComboBox control also contains multiple items but typically occupies less space on the screen.

Note: the user can enter new item in the ComboBox, but in listbox user can't enter new item.

Basic Properties

In the following sections, you'll find the properties that determine the functionality of the List-Box, CheckedListBox, and ComboBox controls. These properties are usually set at design time,but you can change the settings from within your application's code.

IntegralHeight

This property can be set to a True/False value that indicates whether the control's height will be adjusted to avoid the partial display of the last item.

Items
The Items property is a collection that holds the list items for the control.

SelectionMode
This property, which applies to the ListBox and CheckedListBox controls only, determines how the user can select the list's items., The possible values of this property

Table 6.1

Value	Description
None	No selection at all is allowed.
One (Default)	Only a single item can be selected.
MultiSimple	Simple multiple selection: A mouse click (or pressing the spacebar) selects or deselects an item in the list. You must click all the items you want to select.
MultiExtended	Extended multiple selection: Press Shift and click the mouse (or press one of the arrow keys) to select multiple contiguous items. This process highlights

all the items between the previously selected item and the current selection.

Press Ctrl and click the mouse to select or deselect multiple single Items in the list.

Text

The Text property returns the selected text on the control.

Sorted property

True, the item is sorted along with the existing items and then placed in its proper position in the list.

Manipulating the Items Collection

To manipulate a ListBox control from within your application, you should be able to do the following:

- ◆Add items to the list
- ◆Remove items from the list
- ◆Access individual items in the list

The *Add* Method

To add items to the list, use the Items.Add or Items.Insert method. The Add method accepts as an argument the object to be added to the list. New items are appended to the end of the list.

Example3: The following loop adds the elements of the array *words* to a ListBox control, one at a time:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim words() As String = {"new", "old", "big", "small"}
    Dim i As Integer
    For i = 0 To words.Length - 1
       ListBox1.Items.Add(words(i))
    Next

  End Sub
```

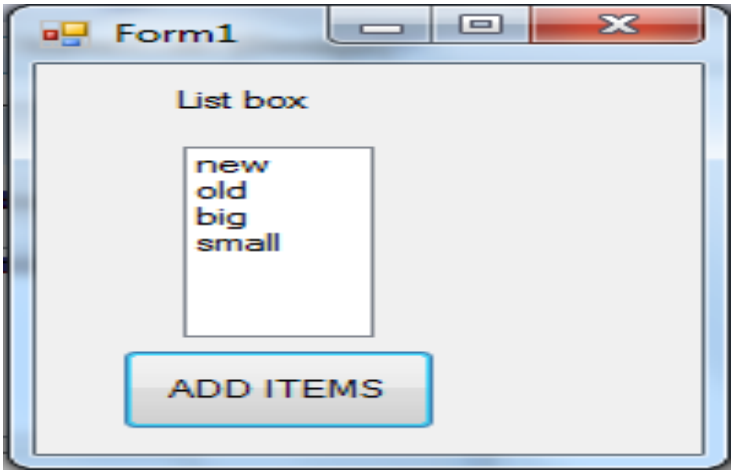


Figure (6.3)

Then, to iterate through all the items on the control, use a loop such as the following:

```
Dim i As Integer
For i = 0 To ListBox1.Items.Count - 1
{ statements to process item ListBox1.Items(i) }

Next
```

The *Insert* Method

To insert an item at a specific location, use the Insert method, whose syntax is as follows:

```
ListBox1.Items.Insert(index, item)
```

The *Clear* Method

The Clear method removes all the items from the control. Its syntax is quite simple:

```
ListBox1.Items.Clear
```

The *Count* Property

This is the number of items in the list. If you want to access all the items with a For…Nextloop, the loop's counter must go from 0 to ListBox.Items.Count – 1, as shown in the exampleof the Add method.

The *CopyTo* Method

The CopyTo method of the Items collection retrieves all the items from a ListBox control and stores them in the array passed to the method as an

argument. The syntax of the CopyTo method is as follows, where destination is the name of the array that will accept the items, and index is the index of an element in the array where the first item will be stored:

```
ListBox1.CopyTo(destination, index)
```

The *Remove* and *RemoveAt* Methods

To remove an item from the list, you can simply call the Items collection's Remove method, passing the object to be removed as an argument You can also remove an item by specifying its position in the list via the RemoveAt method, which accepts as argument the position of the item to be removed:

```
ListBox1.Items.RemoveAt(index)
```

The index parameter is the order of the item to be removed, and the first item's order is 0.

The *Contains* Method

returns a True/False value that indicates
whether the collection contains this object. Use the Contains method to avoid the insertion of identical objects into the ListBox control.

```
Dim itm As String = "Remote Computing"
If Not ListBox1.Items.Contains(itm) Then
ListBox1.Items.Add(itm)

End If
```

Selecting Items

The ListBox control allows the user to select either one or multiple items, depending on the setting of the SelectionMode property.

NOTE: the ComboBox does not allow the selection of multiple items, it provides only the SelectedIndex and SelectedItem properties.

To iterate through all the selected items in a multiselection ListBox control, use a loop such as the following:

```
For Each itm As Object In ListBox1.SelectedItems
Debug.WriteLine(itm)
Next
```

NOTE: The *itm* variable should be declared as Object because the items in the ListBox control are objects.

The ListBox Demo Project

Example4:The ListBox Demo application (shown in Figure below) demonstrates the basic operations of the ListBox control.

Figure (6.4)

It shows you how to do the following:

- ◆Add and remove items at runtime
- ◆Transfer items between lists at runtime
- ◆Handle multiple selected items
- ◆Maintain sorted lists

The Add Item Buttons

The Add Item buttons use the InputBox() function to prompt the user for input, and then they add the user-supplied string to the ListBox control.. The code is identical for both buttons.

```
Dim ListItem As String
ListItem = InputBox("Enter new item's name")
If ListItem.Trim <> "" Then
sourceList.Items.Add(ListItem)
End If
```

NOTE: Clear button is very simple using Clear method to clear the corresponding list

Removing Items from the Two Lists

The code for the Remove Selected Item button removes the selected item, while the Remove Selected Items buttons must scan all the items of the left list and remove the selected one(s).

```
Private Sub bttnDestinationRemove_Click(…)
Handles bttnDestinationRemove.Click
destinationList.Items.Remove( destinationList.SelectedItem)
End Sub

Private Sub bttnSourceRemove_Click(…)
Handles bttnSourceRemove.Click
Dim i As Integer
For i = 0 To sourceList.SelectedIndices.Count - 1
sourceList.Items.RemoveAt( sourceList.SelectedIndices(0))
Next
End Sub
```

Moving Items Between Lists

The two single-arrow buttons (located between the ListBox controls shown in Figure above) transfer selected items from one list to another. The button with the single arrow pointing to the right transfers the items selected in the left list after it ensures that the list contains at least one selected item. Its code is presented in Listing 5.12. First, it adds the item to the second list, and then it removes the item from the original list. Notice that the code removes an item by passing it as an argument to the Remove method because it doesn't make any difference which one of two identical objects will be removed. Moving the selected items

```
sourceList.SelectedIndices(0)))
sourceList.Items.Remove(sourceList.Items(
sourceList.SelectedIndices(0)))
End While
End Sub
```

The second single-arrow button transfers items in the opposite direction.

NOTE: The destination control (the one on the right) doesn't allow the selection of multiple items, so you can use the SelectedIndex and SelectedItem properties. The event handler that moves a single item from the right to the left ListBox is shown next:

```
sourceList.Items.Add(destinationList.SelectedItem)
destinationList.Items.RemoveAt(destinationList.SelectedIndex)
```

Searching the ListBox

Two of the most useful methods of the ListBox control are the FindString and FindString- Exact methods, which allow you to quickly locate any item in the list.

The FindString method

locates a string that partially matches the one you're searching for; FindStringExact finds an exact match.

The syntax for both methods is the same, where *searchStr* is the string you're searching for:

```
itemIndex = ListBox1.FindString(searchStr)
```

An alternative form of both methods allows you to specify the index where the search begins:

```
itemIndex = ListBox1.FindString(searchStr, startIndex)
```

NOTE:Both methods return the index of the item you're searching for on the control, or the value –1 if no such item exists.

The ComboBox Control

The ComboBox control is similar to the ListBox control in the sense that it contains multiple items and the user may select one, but it typically occupies less space onscreen. The ComboBox is practically an expandable ListBox control, which can grow when the user wants to make a selection and retract after the selection is made. Normally, the ComboBox control displays oneline with the selected item because this control doesn't allow multiple-item selection. The essential difference, however, between ComboBox and ListBox controls is that the ComboBox allows the user to specify items that don't exist in the list.

There are three types of ComboBox controls. The value of the control's DropDownStyle property determines which box is used, these values are shown in the table below.

DropDownStyle options for the ComboBox control

Table (6.2)

Value	Effect
DropDown (Default)	The control is made up of a drop-down list, which is visible at all times, and a text box. The user can select an item from the list or type a newone in the text box.
DropDownList	This style is a drop-down list from which the user can select one of its items but can't enter a new one. The control displays a single item,and the list is expanded as needed.
Simple	The control includes a text box and a list that doesn't drop down. The user can select from the list or type in the text box

NOTE: Properties,methods, and events are similar to listbox except that deletion of multiselect property, and addition of style property

Example5:.(Figure below show The ComboBox StylesProject)

Shows the code used to fill the combo boxes with values. As you do with a list box, you use the Items collection's Add method to add an item to a combo box. You can use any of the following properties to select **a default item**, which will appear in the text portion of the combo box:

SelectedIndex, SelectedItem, or Text. If no item is selected, the SelectedItem and Text properties contain the empty string, and the SelectedIndex property contains –1 (negative one).

NOTE: to load Combobox or Listbox with items we shall use the load event of form.

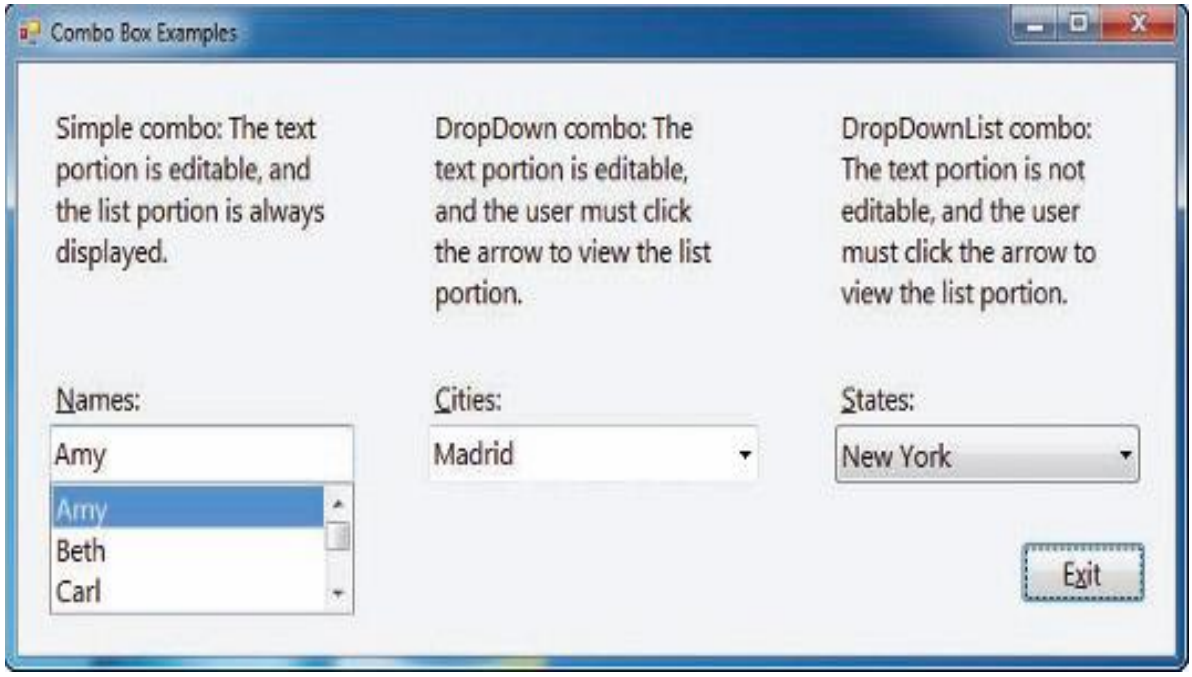


Figure (6.5)

```
Private Sub frmMain_Load(ByVal sender As Object,
ByVal e As System.EventArgs) Handles Me.Load
' fills the combo boxes with values
cboName.Items.Add("Amy")
cboName.Items.Add("Beth")
cboName.Items.Add("Carl")
cboName.Items.Add("Dan")
cboName.Items.Add("Jan")
cboName.SelectedIndex = 0
cboCity.Items.Add("London")
cboCity.Items.Add("Madrid")
cboCity.Items.Add("Paris")
cboCity.SelectedItem = "Madrid"
cboState.Items.Add("Alabama")
cboState.Items.Add("Maine")
cboState.Items.Add("New York")
cboState.Items.Add("South Dakota")
cboState.Text = "New York"
End Sub
```

you can use any of these three properties to select the default item in a combo box

NOTE: If the combo box is a DropDownList style, where the text portion is not editable, you can use the SelectedItem and Text properties interchangeably.

Example 6: to give Squareroot ,Exponential, or power of numbers by putting the numbers combobox1 and chose the operation user want toimplement from and display result in text1box.

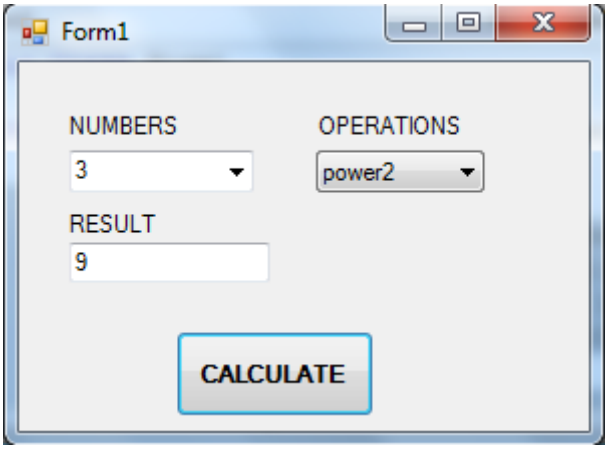


Figure (6.6)

Put the following code in the command button(calculate):

```
If ComboBox2.SelectedIndex = 0 Then
    TextBox1.Text = Math.Sqrt(Val(ComboBox1.Text))
ElseIf ComboBox2.SelectedIndex = 1 Then
    TextBox1.Text = Math.Exp(Val(ComboBox1.Text))
Else
    TextBox1.Text = Val(ComboBox1.Text) * Val(ComboBox1.Text)
End If
```

The ScrollBar and TrackBar Controls

The ScrollBar and TrackBar controls let the user specify a magnitude by moving a selector between its minimum and maximum values. The TrackBar control is similar to the ScrollBar control, but it doesn't cover a continuous range of values. The TrackBar control has a fixed number of tick marks and users can place the slider's indicator to the desired value. In short, the ScrollBar control should be used when the exact value isn't as important as the value's effect on another object or data element. The TrackBar control should be used when the user can type a numeric value and the value your application expects is a number in a specific range—for example, integers between 0 and 100 or a value between 0 and 5 inches in steps of 0.1 inches (0.0, 0.1, 0.2 . . . 5.0). The TrackBar control is preferred to the TextBox control in similar situations because there's

no need for data validation on your part. The user can specify only valid numeric values with the mouse.

HScrollBar and VScrollBar controls

The basic properties of the ScrollBar control, therefore, are properly named Minimum,Maximum, and Value.

Minimum The control's minimum value. The default value is 0, but because this is an Integer value, you can set it to negative values as well.

Maximum The control's maximum value. The default value is 100, but you can set it to any value that you can represent with the Integer data type.

Value The control's current value, specified by the indicator's position.

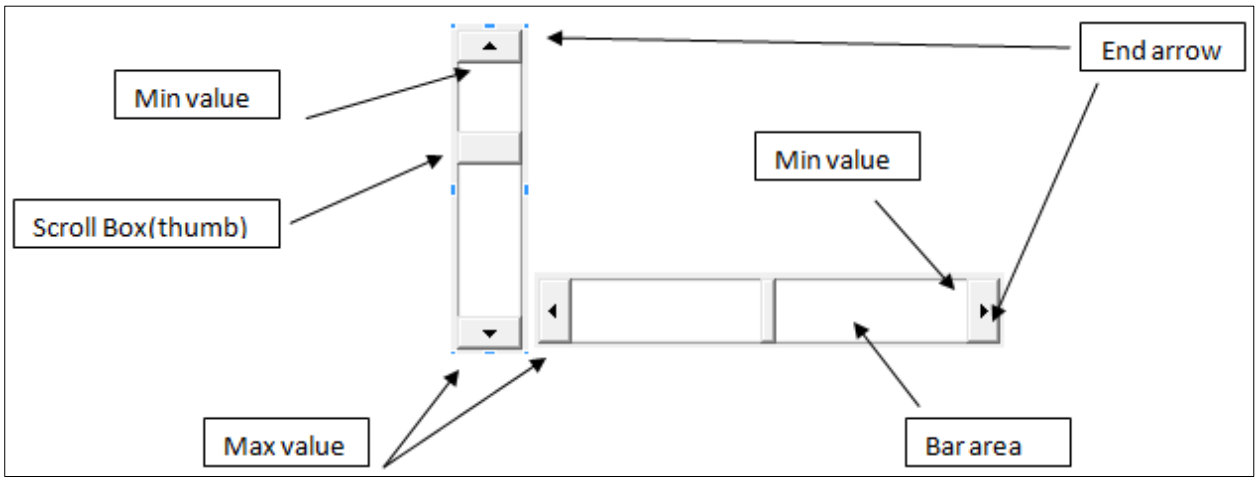


Figure (6.7)

Example7: (vertical scroll bar).this example will convert fehrenheit temrature to Celsius temrature by considering the value of the scroll bar represent the fehernheit temrature and the results will appear on label by using it's caption property.

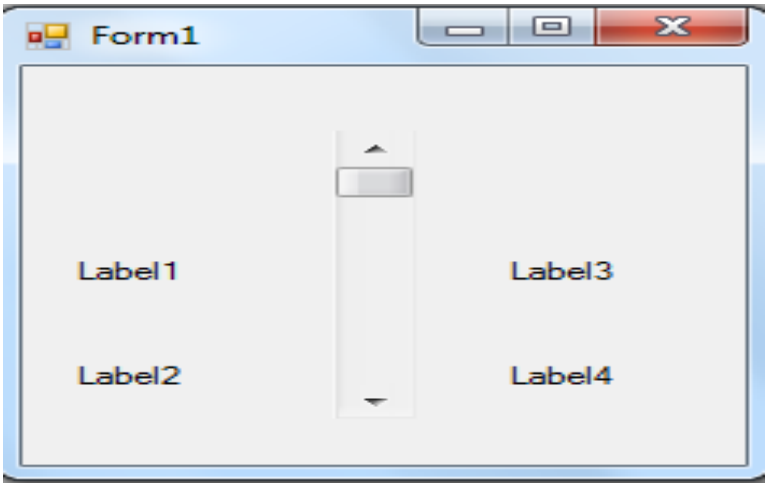


Figure (6.8)

NOTE: Set the max property to 120, and min to -60, then write this code.

Write this statement in Class Form code section:

```
Dim temf, tempc As Integer
```

Then write the following code in the Scroll event of the VscrollBar1 control

```
Private Sub VScrollBar1_Scroll()
temf = VScrollBar1.Value: Label2.Text = Str(temf)
tempc = CInt((temf - 32) * 5 / 9):Label4.Text = Str(tempc)
End Sub ,
```

Finally run the program and test the result.

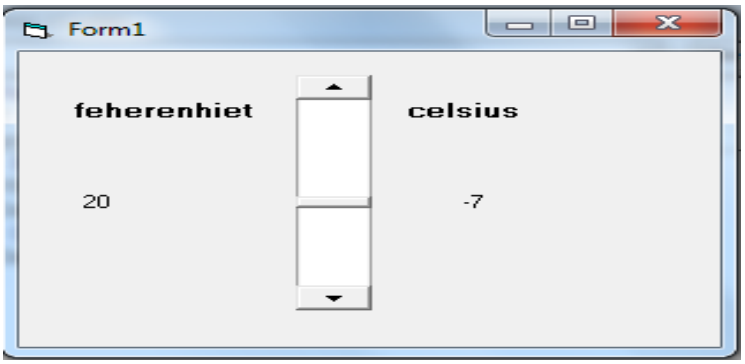


Figure (6.9)

The TrackBar Control

The TrackBar control is similar to the ScrollBar control, but it lacks the granularity of ScrollBar.
Suppose that you want the user of an application to supply a value in a specific range, such as the speed of a moving object. Moreover, you don't want to allow extreme precision; you need only a few distinct settings. The user can set the control's value by sliding the indicator or by clicking on either side of an indicator. Figure below The Inches application demonstrates the use of the TrackBar control in specifying an exact value in specific range

Figure (6.10)

Granularity determines how specific you want to be in measuring. In measuring distances between towns, a granularity of a mile is quite adequate. In measuring (or specifying) the dimensions of a building, the granularity could be on the order of a foot or an inch. The TrackBar control lets you set the type of granularity that's necessary for your application. Similar to the ScrollBar control, SmallChange and LargeChange properties are available. SmallChange is the smallest increment by which the Slider value can change. The user can change the slider by the SmallChange value only by sliding the indicator. (Unlike with the ScrollBar control, there are no arrows at the two ends of the Slider control.) To change the Slider's value by LargeChange, the user can click on either side of the indicator.

Procedure

Using the FindString and FindString-Exact methods to find the string in textbox that match the list item, if entered string by user match the list item, then display message "found", if partially match then display "partially found", else display "not found"

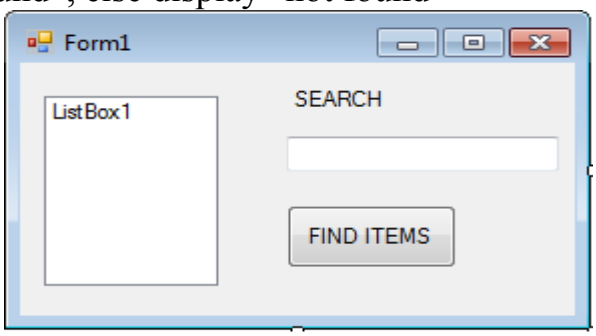


Figure (6.12)

Discussion

1- Users can select one color and one size of the T-shirt example (example2) mentioned above.

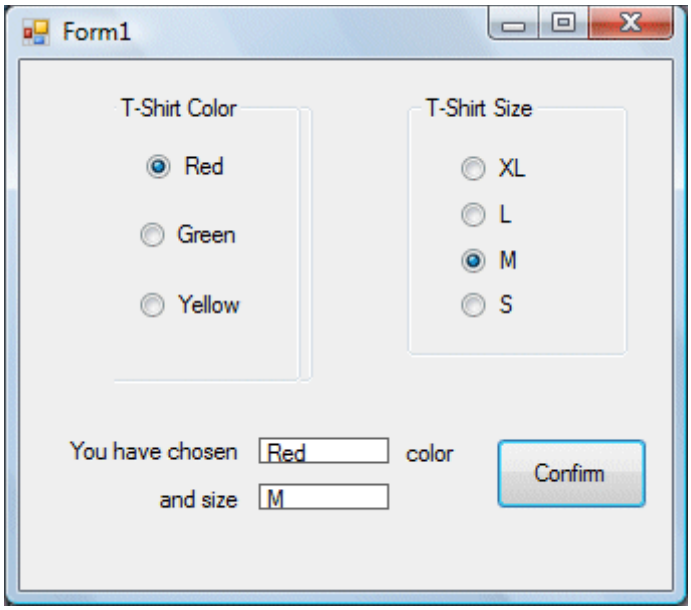


Figure (6.11)

2- Repeat example7 with event change of vertical scroll bar.

Experiment No. (7)

Procedures and Functions + Error Handling

Object

Using procedure and function which are useful for condensing repeated operations.

Theory

Visual Basic programs can be broken into smaller logical components(small piece of the code) called Procedures There are two types of Sub procedures in Visual Basic: event procedures and independent Sub procedures. An event procedure is a Sub procedure that is associated with a specific object and event, such as a button's Click event or An independent Sub procedure, is a procedure that is independent of any object and event(your own Sub procedure). An independent Sub procedure is processed only when called (invoked) from code. Procedures are useful for condensing repeated operations such as the frequently used calculations, text and control manipulation etc.

The benefits of using procedures in programming are:

It is easier to debug a program with procedures, which breaks a program into discrete logical limits make it easy to understand). Procedures used in one program can act as building blocks for other programs with slight modifications(independent Sub procedures are used extensively in large and complex programs, which typically are written by a team of programmers). Together, subroutines and functions are sometimes called *routines* or *procedures.* They arealso sometimes called *methods*.

NOTE: A Procedure can be Sub, Function or Property Procedure.

syntax

A sub procedure can be placed in **standard**, **class** and form **modules**. Each time the procedure is called, the statements between Sub and End Sub are executed. The syntax for a sub procedure is as follows:

```
[Private | Public] Sub Procedurename [(parameterList)]
[statements]
End Sub
```

Under *Scope*, *Public* is selected to create a procedure that can be invoked outside the module, or Private to create a procedure that can be invoked only from within the module.

Syntax of the Call statement

Call *procedureName***(**[*argumentList*]**)**

Steps
1. Click a blank line in the Code Editor window. The blank line can be anywhere
between the Public Class and End Class clauses. However, it must be outside any other Sub or Function procedure.
2. Type the Sub procedure header and then press Enter. The Code Editor automatically enters the End Sub clause for you.

```
Private Sub ClearLabels()
lblFirstName.Text = String.Empty
lblLastName.Text = String.Empty
lblAge.Text = String.Empty
End Sub
```

Note: the procedure can input and return vlaues through its arguments,and not has an output data type,also procedure may not has input or output arguments like:

Example1

```
Sub testproc(x as integer,y as string)
Sub testnew()
```

Example2:

```
Sub newsub(valu as integer)
Msgbox "the number is=" & valu
End sub
```

So the input to this procedure is the vlue of the integer variable value

Example3::

```
Sub calc(x,y,s1,s2,s3 as integer,s4 as single)
S1=x+y
S2=x-y
S3=x*y
S4=x/y
End sub
```

It's clear from this example that the procedure's arguments(x and y) considered as input values to the procedure And in the same procedure consider the arguments(s1,s2,s3,s4) as outputs arguments.

Function Procedures

Functions are like sub procedures, except they return a value to the calling procedure. useful for taking one or more pieces of data, called *arguments* and performing some tasks with them. Then the functions returns a value that indicates the results of the tasks complete within the function.

Syntax

Private Function *procedureName*(**[***parameterList***]**) **As** *dataType*
statements
Return *expression*
End Function

The statement's syntax is **Return *expression***, where *expression* represents the one and only value that will be returned to the statement invoking the function. The data type of the *expression* must agree with the data typespecified in the As *dataType* section of the function's header.

Steps

1. Click a blank line in the Code Editor window. The blank line can be anywhere between the Public Class and End Class clauses. However, it must be outside any other Sub or Function procedure.
2. Type the Function procedure header and then press Enter. The Code Editor automatically enters the End Function clause for you.

Example 4

```
Private Function GetNewPrice(ByVal dblOld As Double) As Double
' increases current price by 5% and returns new price
Dim dblNew As Double
dblNew = dblOld * 1.05
Return dblNew
End Function
```

Example 5

```
Private Function GetNewPrice(ByVal dblOld As Double) As Double
' increases current price by 5% and returns new price
Return dblOld * 1.05
End Function
```

NOTE: You can invoke a function from one or more places in an application's code.

NOTE: A Function procedure returns a value and a Sub Procedure does not return a value.

NOTE: In the function declaration,must declare the output datatype of the function which is differ from sub procedure that function return just one output(result) By it's Return statement.

NOTE: Method — A generic name for a named set of commands. In Visual Basic, both subs and functions are types of methods.

Example6::

```
Function test(valu as integer) as string
Return "the age is" & valu
End function
```

*To invoke the function from the body of program put this code in button-click as follows

```
Dim st as string
St=test(20)
MsgBox(st)
```

Important Note::there are two ways to passing the arguments value to and from the procedure to the main program

Every variable has both a value and a unique address that represents its location in the computer's internal memory. Visual Basic allows you to pass either a copy of the variable's value or the variable's address to the receiving procedure. Passing a copy of the variable's value is referred to as passing by value. Passing a variable's address is referred to as passing by reference. The method you choose—*by value* or *by reference*—depends on whether you want the receiving procedure to have access to the variable in memory. In other words, it depends on whether you want

to allow the receiving procedure to change the variable's contents. To illustrate, assume you have a savings account at a local bank. During a conversation with your friend Melissa, you mention the amount of money you have in the account. Sharing this information with Melissa is similar to passing a variable *by value*. Knowing your account balance does not give Melissa access to your bank account. It merely provides information thatshe can use to compare to the balance in her savings account. The savings account example also provides an illustration of passing information *by reference*. To deposit money to or withdraw money from your account, you must provide the bank teller with your account number. The account number represents the location of your account at the bank and allows the teller to change the account balance. Giving the teller your bank account number is similar to passing a variable *by reference*. The account number allows the teller to change the contents of your bank account, similar to the way the variable's address allows the receiving procedure to change the contents of the variable.

First: passing by reference or some times called by address of the argument.in this way the argument vlue will be changed by the calling program and the called procedure

Example7:: sub test(x,y)

Second: passing by value,in this way the argument will retain it's value in the calling program(main program),but will changed in the called program.

Example8:: sub test(byval x,byref y), it means that x will be passing by its value, and y passing by its address

NOTE::if we want to exit procedure or function for any reason or condition, we can write in the body of the procedure Exit sub,,,,or exit function

Example9 to explain passing by reference and by value, this example just give an explination to the concept of passing variables by using two variables i, j as example as in the following form:

```
Public Sub sample(ByVal i As Integer, ByRef j As Integer)
    i = i * 2
    j = j * 2
    MsgBox("sample procedure   " & i & "   " & j)
End Sub
```

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

        Dim x, y As Integer
        x = 3
        y = 2
        MsgBox("in main program x  y equal to " & x & "   " & y)
        Call sample(x, y)
        MsgBox("after first call x  y equal to   " & x & "    " & y)
        Call sample(x, y)
        MsgBox("after second call x  y equal to   " & x & "    " & y)

    End Sub
```

Run the program, then you will get the following results

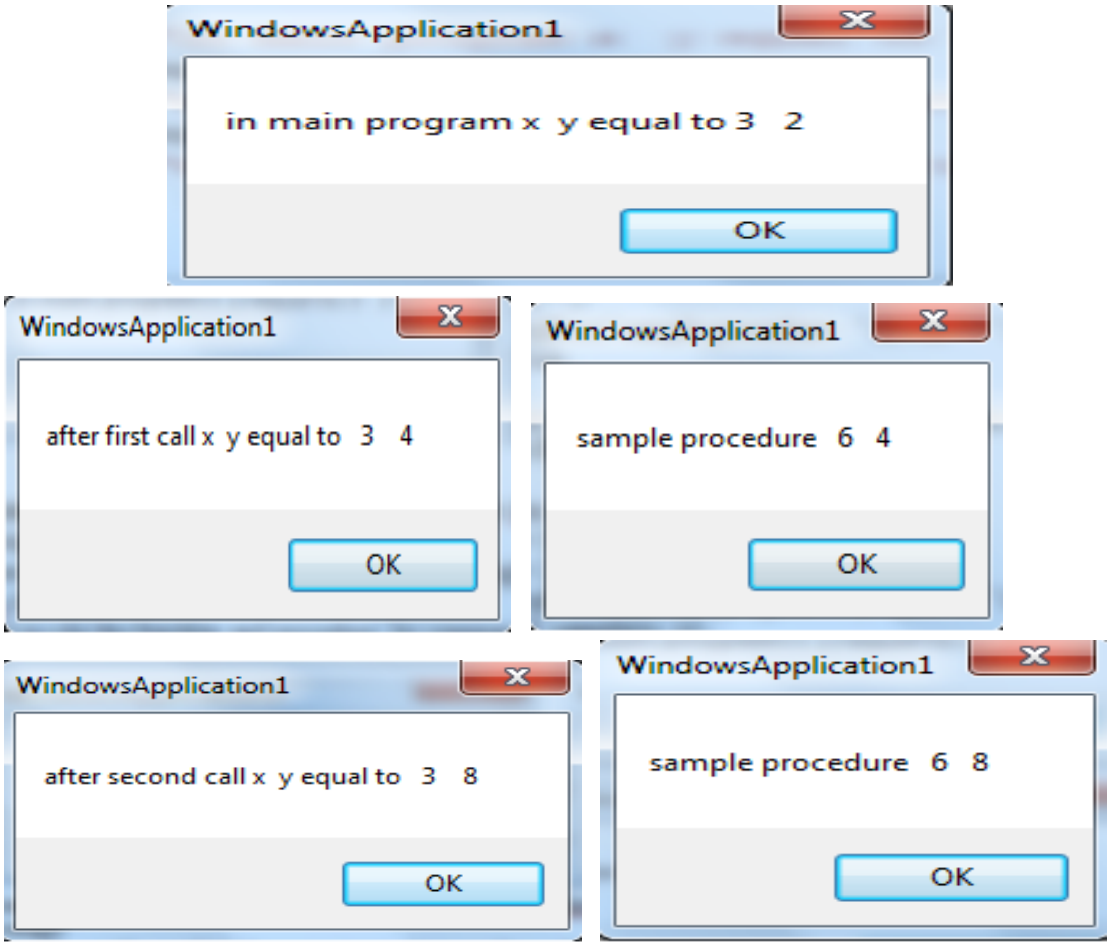


Figure (7.1)

Example 10 of using addsub procedure to sum three dgrees of student,and avg function to give the average Here the code explain how to input degrees through the procedure then call it in the body of the avg function And invoke the function and procedure by command1 code

```
Public Sub addsub(ByRef x As Integer, ByRef y As Integer, ByRef z As
Integer, ByRef sum As Integer)
    x = Val(TextBox1.Text)
    y = Val(TextBox2.Text)
    z = Val(TextBox3.Text)
    sum = x + y + z
End Sub
Function avg() As Single
    Dim s As Integer
    Dim x, y, z As Integer
    Call addsub(x, y, z, s)
    MsgBox("the sum of the three numbers is   " & s)
    avg = s / 3
    'NOTE: also we can get the result of avg function  by return
statement
    ' Return (s / 3)
End Function

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim result As Single
    result = avg()
    MsgBox(" the average of three numbers is   " & result)
End Sub
```

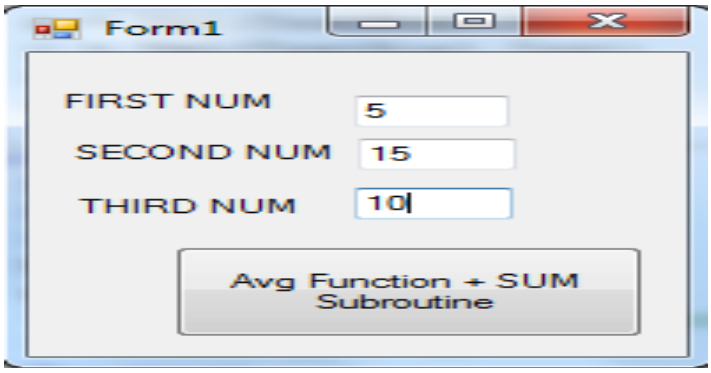


Figure (7.2)

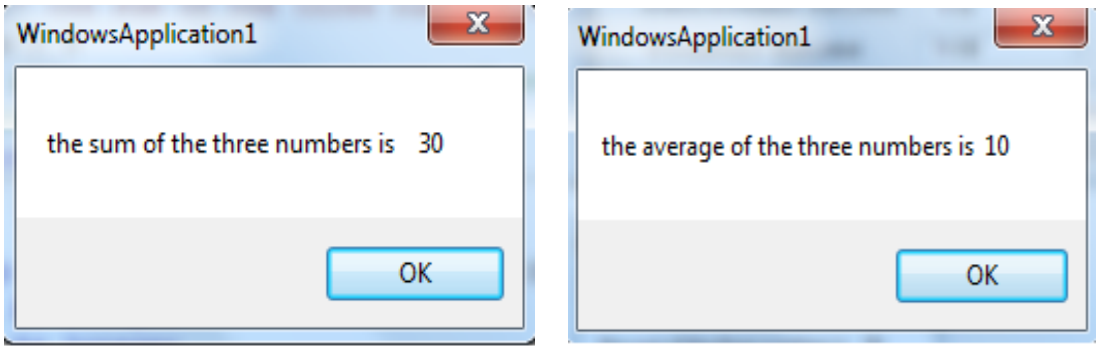


Figure (7.3)

NOTE::avg() function not take any inputs but return output} In the following example for the same program see how can input being through avgfunction like this Avg(d1 as integer,d2 as integer,d3 as integer) and finally give the same result as the previous program

```
Function avg(ByRef x As Integer, ByRef y As Integer, ByRef z As Integer)
As Single
    avg = (x + y + z) / 3
  End Function

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim d1 As Integer
    Dim d2 As Integer
    Dim d3 As Integer
    Dim result As Single
    d1 = Val(TextBox1.Text)
    d2 = Val(TextBox2.Text)
    d3 = Val(TextBox3.Text)
    result = avg(d1, d2, d3)
    MsgBox("the result is    " & result)
  End Sub
```

Error Handling(Exception handling)

Errors often occur due to incorrect input from the user. Error handling can help make the program error-free, An error-free program can run efficiently, Therefore a good programmer should be more alert to the parts of program that could trigger errors and should write errors handling code to help the user in managing the errors.

There are three error types:

1- **syntax error**:occure when mistype some phrase in the language or forget to close parantheses.
2- **Run_time error**(divide by zero),(or open some file which is not exist), enter wrong data(char not number in mathematics operation)…….etc.
3- **Logical error**(semantic error):occure when the program syntax is true but the result of the program after running is not correct or inexpected.(example, endless loop).

In VB can write sub routine to handle the run-time error by writing routine to correct the Expected errors in our program if occur.

NOTE: VB2010 has improved a lot in built-in errors handling compared to Visual Basic 6. For example, when the user attempts to divide a number by zero, Vb2010 will not return an error message but gives the 'infinity' as the answer.

Using On Error GoTo Syantax

Visual Basic 2010 still supports the VB6 errors handling syntax, that is the On Error GoTo *program_label* structure. Although it has a more advanced error handling method, we shall deal with that later. We shall now learn how to write errors handling code in VB2010. The syntax for errors handling (in VB6) is

- On Error GoTo *program_label*

where program_label is the section of code that is designed by the programmer to handle the error committed by the user. Once an error is detected, the program will jump to the program_label section for error handling.

Example 11: Division by Zero

In this example, we will deal with the error of entering non-numeric data into the textboxes that suppose to hold numeric values. The program_label here is error_hanldler. when the user enter a non-numeric values into the textboxes, the error message will display the text "One of the entries is not a number! Try again!". If no error occur, it will display the correct answer. Try it out yourself.

The Code :

```
Public Class Form1

Private Sub CmdCalculate_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles CmdCalculate.Click

Lbl_ErrorMsg.Visible = False

Dim firstNum, secondNum As Double

On Error GoTo error_handler

firstNum = Txt_FirstNumber.Text

secondNum = Txt_SecondNumber.Text

Lbl_Answer.Text = firstNum / secondNum

Exit Sub 'To prevent error handling even the inputs are valid

error_handler:

Lbl_Answer.Text = "Error"

Lbl_ErrorMsg.Visible = True

Lbl_ErrorMsg.Text = " One of the entries is not a number! Try again!"

End Sub

End Class
```

The Output of the previous code is as the following figure:

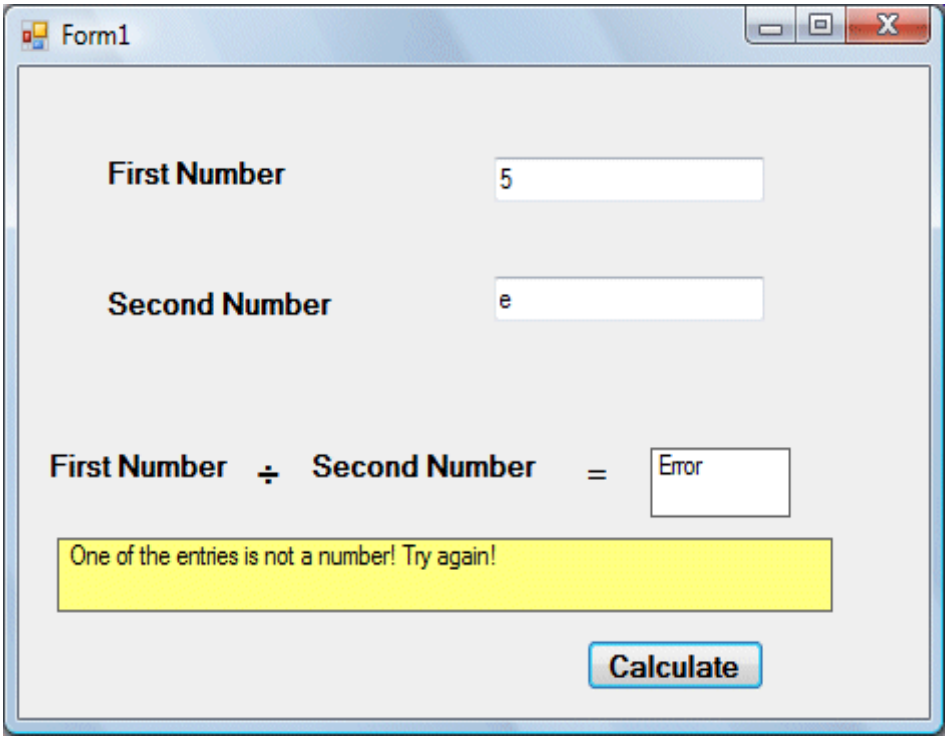


Figure (7.4)

NOTE: In the previous program just exit sub at the error, now we can correct the error and let the program to continue running in correct way By the statement::(Resume :to rtry the operation caused the error) Resume Next:return to the line in the program next to the line error, and Resume label::program return to line that you label it(give it some label).

Errors Handling(VB2010) using Try.....Catch....End Try Structure

VB2010 has adopted a new approach in handling errors, or rather exceptions handling. It is supposed to be more efficient than the old On Error Goto method, where it can handles various types of errors within the Try...Catch...End Try structure.
The structure looks like this

Try

statements

Catch *exception*_variable as Exception

statements to deal with exceptions

End Try

Example12:

This is a modification of Example 11. Instead of using On Error GoTo method, we shall use the Try...Catch...End Try method. In this example, the Catch statement will catch the exception when the user enters a non-numeric data and return the error message. If there is no exception, there will not any action from the Catch statement and the program returns the correct answer.

The code

```
Public Class Form1

Private Sub CmdCalculate_Click(ByVal sender As System.Object, ByVal eAs
 System.EventArgs) Handles CmdCalculate.Click

Lbl_ErrorMsg.Visible = False

Dim firstNum, secondNum, answer As Double

Try

firstNum = Txt_FirstNumber.Text

secondNum = Txt_SecondNumber.Text

answer = firstNum / secondNum

Lbl_Answer.Text = answer

Catch ex As Exception

Lbl_Answer.Text = "Error"

Lbl_ErrorMsg.Visible = True

Lbl_ErrorMsg.Text = " One of the entries is not a number! Try again!"

End Try

End Sub

End Class
```

The output

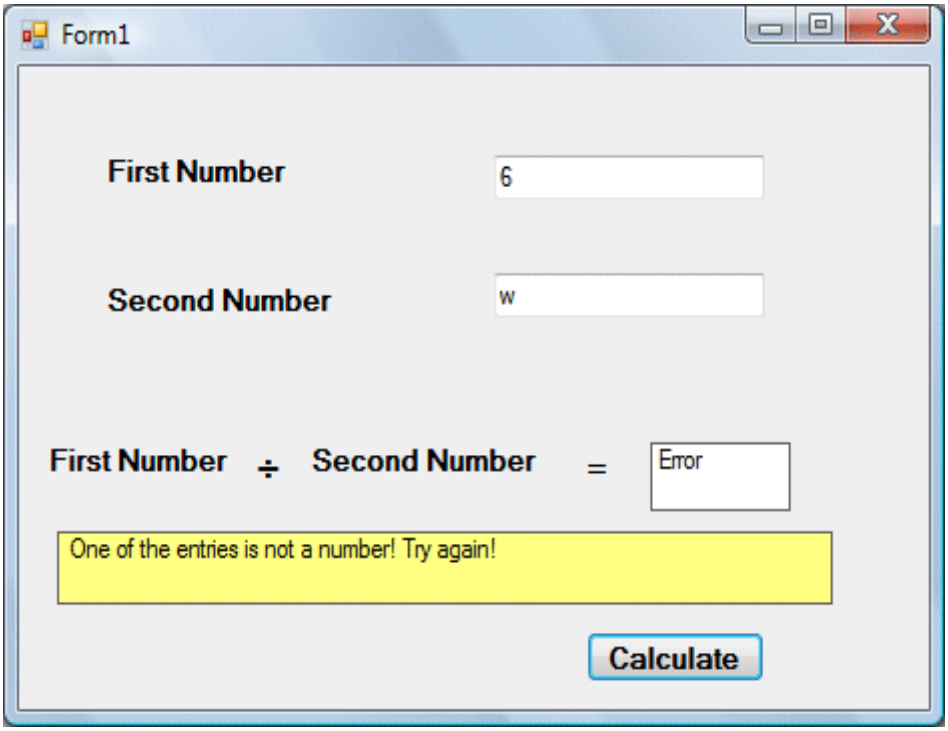


Figure (7.5)

Sections of the Try...End Try Structure

Table (7.1)

Section	Description
Try	The Try section is where you place code that might cause an exception. You can place all of a procedure's code within the Try section, or just a few lines.
Catch	Code within the Catch section executes only when an exception occurs; it's the code you write to catch the exception.
Finally	Code within the Finally section occurs when the code within the Try and/or Catch sections completes. This section is where you place your *cleanup code*—code that you always want executed, regardless of whether an exception occurs.

NOTE: If no exception occurs, code within the Catch section is ignored.

NOTE: When all statements within the Finally section finish executing, execution jumps to the statement immediately following End Try.

NOTE: Like other code structures, Visual Basic has a statement that can be used to exit a Try...End Try structure at any time: Exit Try. Note, however, that if you use Exit Try, code jumps to the Finally section and then continues with the statement immediately following the End Try statement.

Dealing with an Exception

You'll probably also want to tell the user what type of exception occurred. To do this, you must have a way of knowing what exception was thrown. This is also important if you intend to write code to deal with specific exceptions. The Catch statement enables you to specify a variable to hold a reference to an Exception object. Using an Exception object, you can get information about the exception. The following is **the syntax** used to place the exception in an Exception object:

```
Catch variablename As Exception
```

Modify your Catch section to match the following:

```
Catch ex As Exception
    MessageBox.Show("An error has occurred: " & ex.Message)
```

NOTE: the exception object has many properties, here we just mention the message property. The Message property of the Exception object contains the text that describes the specific exception that occurred.

Handling an Anticipated Exception

At times, you'll anticipate a specific exception being thrown. For example, you might write code that attempts to open a file when the file does not exist. In such an instance, you'll probably want the program to perform certain actions when this exception is thrown. When you anticipate a specific exception, you can create a Catch section designed specifically to deal with that one exception. Recall from the previous section that you can retrieve information about the current exception by using a Catch statement, such as

Catch objException **As Exception**

By creating a generic Exception variable, this Catch statement catches any and all exceptions thrown by statements within the Try section. To catch a specific exception, change the data type of the exception variable to a specific exception type. Remember the code you wrote earlier that caused a System.InvalidCastException when an attempt was made to pass an empty string to the CLng() function? You could have used a Try...End Try structure to deal with the exception, using code such as this:

```
Dim lngAnswer As Long
Try
lngAnswer = 100 / CLng(txtInput.Text)
MessageBox.Show("100/" & txtInput.Text & " is " & lngAnswer)
Catch objException As System.InvalidCastException
MessageBox.Show("You must enter something in the text box.")
Catch objException As Exception
MessageBox.Show("Caught an exception that wasn't an invalid cast.")
End Try
```

Notice that this structure has two Catch statements. The first Catch statement is designed to catch only an overflow exception; it doesn't catch exceptions of any other type. The second Catch statement doesn't care what type of exception is thrown; it catches all of them. The second Catch statement acts as a catchall for any exceptions that aren't overflow exceptions because Catch sections are evaluated from top to bottom, much as Case statements are in the Select...Case structure. You could add more Catch sections to catch other specific exceptions if the situation calls for it.

Using Multiple Catch Statements

If you want to handle a number of different exceptions, you can use multiple Catch blocks. In the following example, different Catch blocks handle overflow exceptions, out-of-memory exceptions, and array index out-of-range exceptions (which occur when an array index has been set to a negative value or to a value greater than the upper bound of the array). Using multiple Catch blocks like this gives you a good handle on handling exceptions of various types. Each Catch block serves as a

different exception handler, and you can place exception-specific code in each such block.

Example13:

```
Try

Some statements
   Catch e As OverflowException

Specific code that handle the overflow exception(error)

Catch e As OutOfMemoryException

Specific code that handle the outofmemory exception(error)


Catch e As IndexOutOfRangeException

 Specific code that handle the indexoutofrange exception(error)

End try
```

Procedure

1- Write VB program to build sub routine (call it geometry), to calculate the area and primeter of circle:::geometry(radius as single,primeter as single,area as single)…..hint primetercircle=2*pi*radius…….area=pi*(radius)^2
2- Write VB program to implement function that arrange the student degrees ascendant, and then print in labels the largest and smallest degree ,finally print the average of the student.

Discussion

1- Write VB program to divide two numbers,if user enter wrong data then calling the error handling routin using (ON ERROR GOTO method) with resume statement and then execute the same program with resume next.
2- Repeat previous example with message property, Run the project and click the button.
3- Write sample of VB program that implement more than one exception type to handle the errors may be occurred according to these exceptions, such as the exception types mentioned in the last example.

Experiment No. (8)

Modules and Classes + File Processing

Object

Understanding three kind of modules which are FORM, STANDARD and CLASS.

Theory

Code in Visual Basic is stored in the form of modules. The three kind of modules are Form Modules, Standard Modules and Class Modules.

Class A definition of an object Includes properties (variables) and methods, which can be Subs or Functions.

Instance — When a class is created, the resulting object is an instance of the class's definition.

New Creates an instance of an object
Me A reference to the instance of the object within which a method is executing
Namespace A collection of classes that provide related capabilities For example, the System.Drawing namespace contains classes associated with graphics

Sdtandard or code Module A code block that isn't a class but which can contain Sub and Function methods Used when only a **single copy** of code or data is needed in memory

VB2010 allows users to write programs that break down into modules and are knows as classes or types. An object can be created out of a class and it is known as an instance of the class, example is student class is a subclass of the human class while your son John is an instance of the student class.

NOTE: form is class by default.

VB2010 is a full Object Oriented Programming Language have three core technologies namely encapsulation, inheritance and polymorphism.

Encapsulation refers to the creation of self-contained modules that bind processing functions to the data. These user-defined data types are called classes. Each class contains data as well as a set of methods which manipulate the data. The data components of a class are called instance variables and one instance of a class is an object. For example, in a library system, a class could be member, and John and Sharon could be two instances (two objects) of the library class.

Inheritances
Classes are created according to hierarchies, and inheritance allows the structure and methods in one class to be passed down the hierarchy. That means less programming is required when adding functions to complex systems. If a step is added at the bottom of a hierarchy, then only the processing and data associated with that unique step needs to be added. Everything else about that step is **inherited**.

NOTE: The ability to reuse existing objects is considered a major advantage of object technology.

Polymorphism
Object-oriented programming allows procedures about objects to be created whose exact type is not known until runtime. For example, a screen cursor may change its shape from an arrow to a line depending on the program mode. The routine to move the cursor on screen in response to mouse movement would be written for "cursor," and polymorphism allows that cursor to take on whatever shape is required at runtime. It also allows new shapes to be easily integrated.

Creating a Class

You define a class using the **Class statement**, which you enter in a class file.

Class statement

Syntax

Public Class *className*
attributes section
behaviors section
End Class

Adding a class file to an open project:

1. Click Project on the menu bar and then click Add Class. The Add New Item dialog box opens with Class selected in the middle column of the dialog box.
2. Type the name of the class followed by a period and the letters vb in the Name box, and then click the Add button.

NOTE: can create class by right click on the project name in the solution explorer and then choose ADD⟶then choose class.

Instantiating an object from a class (using the class)

After you define a class, it then can be used to instantiate one or more objects.

Syntax – Version 1

{**Dim** | **Private**} *variableName* **As** *className*
variableName **=** **New** *className*

Syntax – Version 2

{**Dim** | **Private**} *variableName* **As New** *className*

className is the name of the class, and
variableName is the name of a variable that will represent the object.

Example 1 (using syntax version 1)

Private hoursInfo As TimeCard
hoursInfo = New TimeCard
the Private instruction creates a TimeCard variable named hoursInfo; the assignment statement instantiates a TimeCard object and assigns it to the hoursInfo variable

Example 2 (using syntax version 2)

Dim hoursInfo As New TimeCard
the Dim instruction creates a TimeCard variable named hoursInfo and also instantiates a TimeCard object, which it assigns to the hoursInfo variable

CreatingPropertyProcedures

Write in general declaration of (class module code) window

Public firstnam as string

Public lastnam as string

Public birthdat as date

Properties are attributes or characteristics of an object. **For example**, a **contact** object could have a birthdat,firstnam,lastnam properties. At any time, we can change the properties of the contact by assigning a value to the properties.

Using the contact Class(how to use the class???)

Once we've created an instance (object) of a class, we can use the methods and properties of the class. we can use the Object Browser in Visual Basic to view the properties, methods, and events that are defined for a class.

Make instance of the class

Dim ocontact as contact

***Make this instance active**

Set ocontact=**new** contact

***To destroy the class at the end of execution**

Set ocontact=nothing

Example 3:Now make test on the class contact Place a command button on Form1. In the Click event for the command1 button (caption=**class test**), type the following:

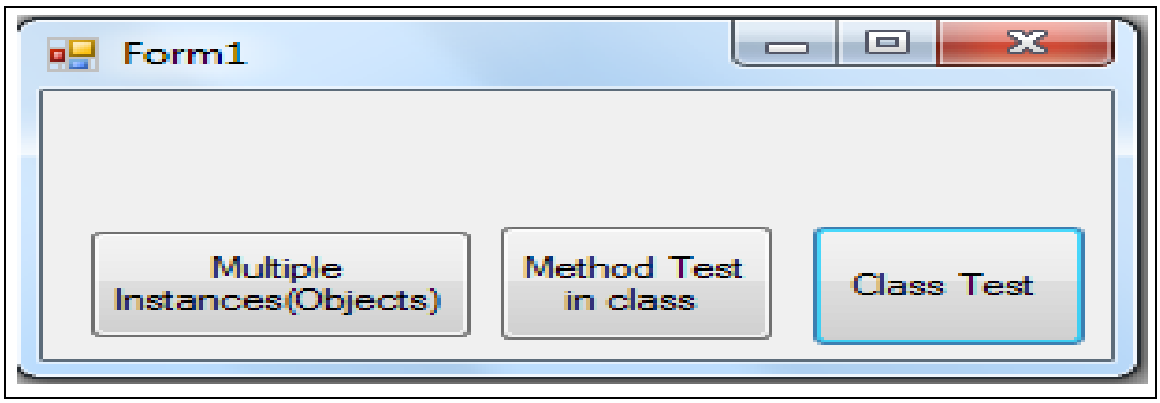


Figure 8.1

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Dim ocontact As contact
        ocontact = New contact

        With ocontact
           .firstnam = "aaa"
           .lastnam = "bbb"
           .birthdat = #5/2/1990#
        End With
        MsgBox(ocontact.firstnam & "  " & ocontact.lastnam & " " &
ocontact.birthdat)
        ocontact = Nothing
    End Sub
```

After running the program the result will be:

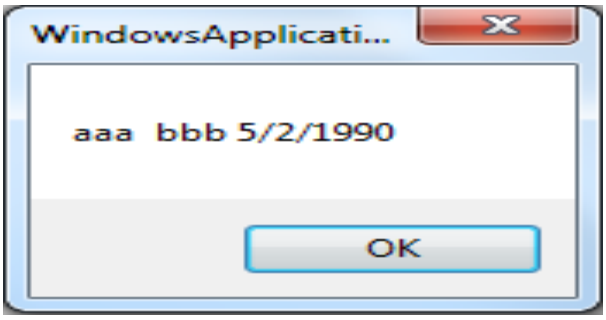


Figure 8.2

Creating Methods

Now we can create any method we need and call it as method of the class

Write in the contact class module window code

Public Sub speak()

MsgBox "my name is " & firstnam & " " & lastnam & " " & "i was born on" & birthdat

End Sub

This method name is speak. And we can call it from the code window of the form module.

Write in the command2 click event(caption=**method test** in class),

```
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
        Dim ocontact As contact
        ocontact = New contact
        With ocontact
           .firstnam = "aaa"
           .lastnam = "bbb"
           .birthdat = #5/2/1990#
        End With
        ocontact.speak()
        ocontact = Nothing

    End Sub
```

After running the program the result will be:

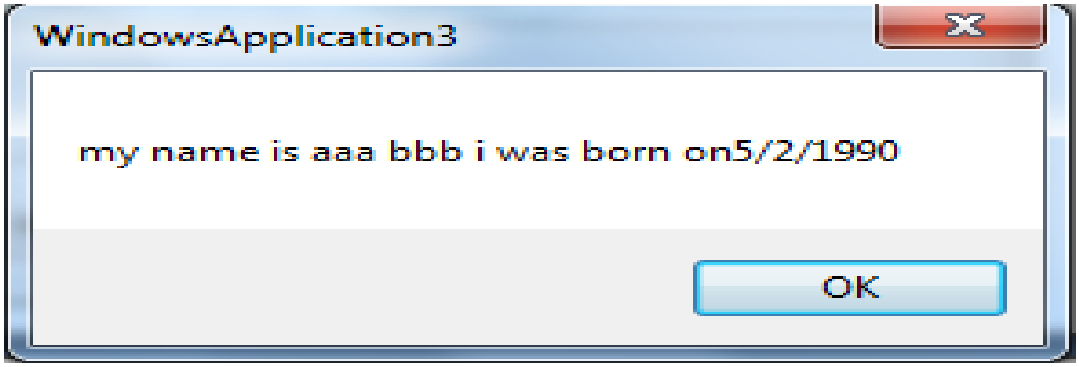


Figure 8.3

Creating multiple instances(object copies)

One of true **benefits** of working with class modules is that instantiate multiple copies of same class,assign each instance its own properties.the following example demonstrates the use of **two different copies of contact objects** in same procedure of form module.

Write this code in the command3 click event(caption=**multiple instances (objects)**)

```
Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button3.Click
        Dim ocontact1 As contact
        Dim ocontact2 As contact

        ocontact1 = New contact
        ocontact2 = New contact
        With ocontact1
           .firstnam = "aaa"
```

```
    .lastnam = "bbb"
    .birthdat = #5/2/1990#
End With
With ocontact2
    .firstnam = "mmm"
    .lastnam = "nnn"
    .birthdat = #2/2/1990#
End With

ocontact1.speak()
ocontact2.speak()
ocontact1 = Nothing
ocontact2 = Nothing

End Sub
```

After running the program, the result will be:

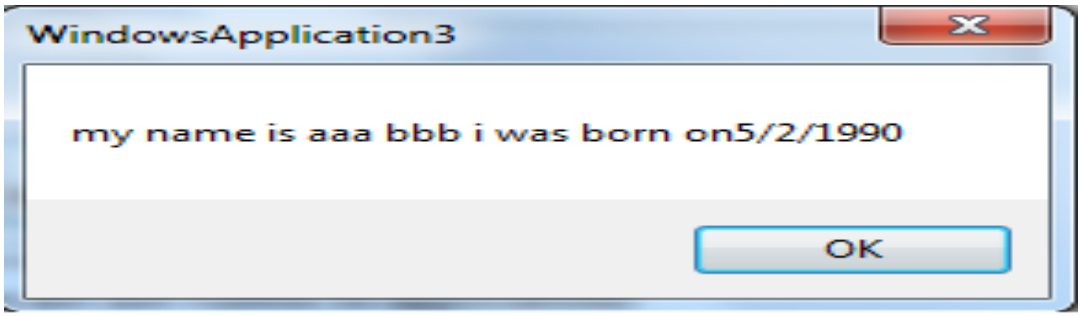


Figure 8.4

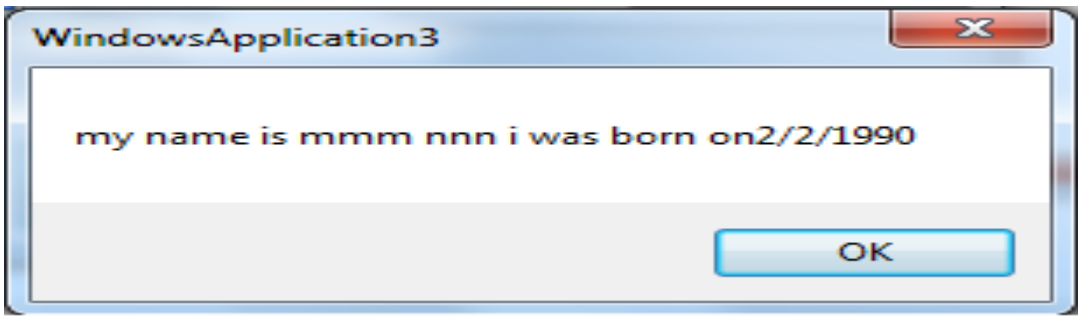


Figure 8.5

PUBLIC Variables

When a variable in a class is declared using the Public keyword, it can be accessed by any application that contains an instance of the class.

To enter the Public variables in the class definition:

1. Enter the following three Public statements:

Public Length **As Double**
Public Width **As Double**

Public Depth **As Double**

Example4: this example calculates the number of gallons required to fill the pool. The class statement, the GetGallons function, and the btnCalc control's Click event procedure. The class statement groups together the three dimensions of a rectangular pool: length, width, and depth. The event procedure declares a class variable and then fills the variable's members with values. It then passes the class variable to the GetGallons function, which calculates and returns the number of gallons required to fill the pool. The event procedure displays the returned value in the lblGallons control.

*In the class code window write the following:

```
Public Class RectangularPool
Public Length As Double
Public Width As Double
Public Depth As Double
End Class
```

entered in the RectangularPool.vb file

*in the form code window write the following:

```
Public Function GetGallons(ByVal pool As RectangularPool) As
Double
‘ calculates and returns the number of gallons
Const dblGAL_PER_CUBIC_FOOT As Double = 7.48
Return pool.Length * pool.Width * pool.Depth *
dblGAL_PER_CUBIC_FOOT
End Function

Private Sub btnCalc_Click(ByVal sender As Object,
ByVal e As System.EventArgs) Handles btnCalc.Click
‘ displays the number of gallons
Dim customerPool As New RectangularPool
Dim dblGallons As Double
Double.TryParse(txtLength.Text, customerPool.Length)
Double.TryParse(txtWidth.Text, customerPool.Width)
Double.TryParse(txtDepth.Text, customerPool.Depth)
dblGallons = GetGallons(customerPool)
lblGallons.Text = dblGallons.ToString(“N0”)
txtLength.Focus()
End Sub
```

receives a RectangularPool object *by value*

instantiates a RectangularPool object and assigns it to the customerPool variable

fills the Public variables withvalues

passes the customerPool variable to the GetGallons function

After running the program set length=60,width=30,depth=5, then you will get the result= 67.320

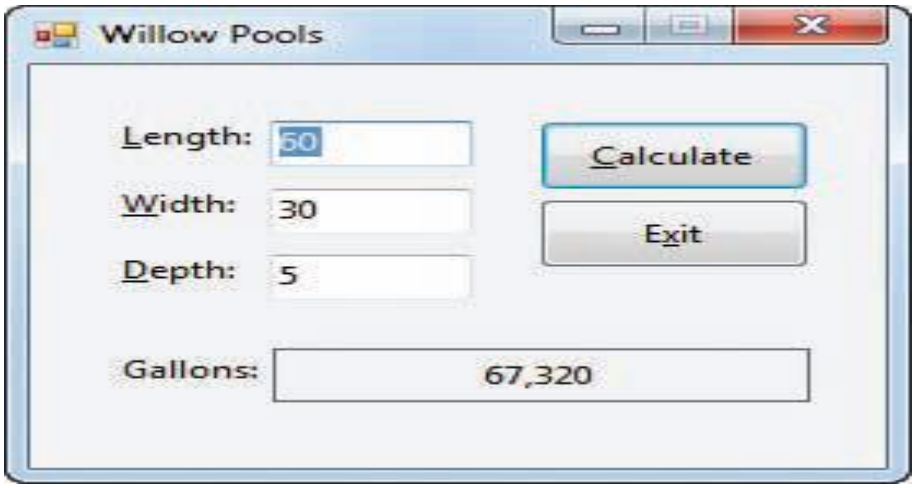


Figure 8.6

Private Variables and Property Procedures

Private variables are not visible to applications that contain an instance of the class A class's Private variables can be used only by instructions within the class itself

Example5:

Private _dblLength **As Double**
Private _dblWidth **As Double**

For an application to assign data to or retrieve data from a Private variable in a class, it must use a **Public property**. In other words, an application cannot directly refer to a Private variable in a class. Rather, it must refer to the variable indirectly, through the use of a Public property.

NOTE:initializes the Private variables directly uses the Public properties to initialize the Private variables indirectly

A Public Property procedure creates a property that is visible to any application that contains an instance of the class.

Property procedure
Syntax
Public [**ReadOnly** | **WriteOnly**] **Property** *propertyName*[**(***parameterList***)**] **As** *dataType*
Get
[*instructions*]
Return *privateVariable*
End Get

Set(ByVal value As *dataType***)**
[*instructions*]
privateVariable = {**value** | *defaultValue*}
End Set
End Property

Example 6 – an application can both retrieve and set the Side property's value

```
Private _intSide As Integer
Public Property Side As Integer
Get
Return _intSide
End Get
Set(ByVal value As Integer)
If value > 0 Then
_intSide = value
Else
_intSide = 0
End If
End Set
End Property
```

Example 7– an application can retrieve, but not set, the Bonus property's value

```
Private _dblBonus As Double
Public ReadOnly Property Bonus As Double
Get
Return _dblBonus
End Get
End Property
```

Example 8 – an application can set, but not retrieve, the AnnualSales property's value

```
Private _decAnnualSales As Decimal
Public WriteOnly Property AnnualSales As Decimal
Set(ByVal value As Decimal)
_decAnnualSales = value
End Set
End Property
```

A constructor is a class method, always **named New**, whose sole purpose is to initialize the class's Private variables. Constructors never return a value, so they are always **Sub procedures** rather than **Function procedures**.

Constructor

Syntax

Public Sub New([*parameterList*]**)**
instructions to initialize the class's Private variables
End Sub

Example 9 (default constructor)

```
Public Sub New()
_dblLength = 0
_dblWidth = 0
End Sub
```

initializes the Private variables directly

Example 10 (parameterized constructor)

```
Public Sub New(ByVal dblL As Double,
ByVal dblW As Double)
Length = dblL
Width = dblW
End Sub
```

uses the Public properties to initialize the Private variables indirectly

Method(sub or function) that is not a constructor

Syntax

Public {**Sub** | **Function**} *methodName***(**[*parameterList*]**)** [**As** *dataType*]
instructions
End {**Sub** | **Function**}

Example 11 (coded as a Function procedure)

```
Public Function GetArea() As Double
Return _dblLength * _dblWidth
End Function
```

Example 12 (coded as a Sub procedure)

```
Public Sub GetArea(ByRef dblA As Double)
dblA = _dblLength * _dblWidth
End Sub
```

Modules (also called a Code Module or Standard Module)

A code module contains only code – declarations, procedures and function – that are used by other files in the project. They are not associated with a class or form and contain no objects or event procedures. Like forms, they are saved in the project as a file with a *.vb* extension.

When you create a large application with multiple forms, many times you will find that you will need to access data or functions/procedures from several different forms. Using a code module to store this information that all files have potential access to avoids duplication of code and makes it easier to maintain the application if you need to modify or update shared information.

Module Names and Module Files - The content of a module begins with the Module statement and ends with an End Module statement. **The syntax is**:

```
Module ModuleName
    [contents of module]
End Module
```

Adding a Module

Follow these steps to add a module to your project:

1. Click the Add New Button on the Toolbar or click Project on the Menu bar, and then click Add Module. The Add New Item dialog box will appear.
2. Under Templates, select Module.
3. Change the default name in the Name text box to the name you want to name the module file. Again, don't forget to type the .vb extension.
4. Click the Add button. A new empty module will be added to the project. It should appear in the Solution Explorer window with all the rest of the files.

Once you have added a code module, you add code for sub procedures and functions to it, just like you would do for a form.

Module-Level Variables

A variable declared inside a module but not inside a procedure or function is called a **module-level** variable. The same rules about the scoped of class-level variables in a form apply to module-level variables in a module.

1. A module-level variable in a standard module is accessible to any procedure or function in the standard module.
2. If a module-level variable is declared with the Dim or Private keywords, the variable is NOT accessible to statements outside the module. This type of variable is said to have **module scope**.
3. If a module-level variable is declared with the Public keyword, it is accessible to all other forms and statements outside the module. This type of variable is said to have **global scope**. Many programmers prefix global variables with the letter g_ to document that it is a global variable.

Example 13: create module call it module1, then built in it subroutine that display message when we call it form window as following:

```
Module1
Sub display()
MsgBox ("first program in module")
End module
```

Then we can call it from form code window by writing this statement

```
Module1.display()
```

FILE PROCESSING IN VB.NET

In addition to getting data from the keyboard and sending data to the computer screen, an application also can get data from and send data to a file on a disk. Getting data from a file is referred to as "reading from the file," and sending data to a file is referred to as "writing to the file." Files to which data is written are called output files, because the files store the output produced by an application. Files that are read by the computer are called input files, because an application uses the data in these files as input. Most input and output files are composed of lines of text that are both read and written sequentially. In other words, they are read and written in consecutive order, one line at a time, beginning with the first line in the file and ending with the last line in the file. Such files are referred to as sequential access files, because of the manner in which the lines of text are accessed. They also are called text files, because they are composed of lines of text. Examples of text stored in sequential access files include an employee list, a memo, and a sales report.

Three basic file types are native to Windows (and MS-DOS before it) and can be processed by application programs. These are:

Table 8.1

text files	Text files generally consist of "plain text" that is readable to the human eye. In a business data processing context, a text file consists of a series of lines (or "records"), each of which contains a set of fields. A data file of this nature is also known as a "flat file".
binary files	Binary files are files which contain data elements that are stored in native machine format. Examples would be executable files (such as .EXE or .DLL files) and files produced by applications such as Excel and Word (.XLS and .DOC files). This would also include data files (record-oriented files) that store one or more fields (such as an Integer field) in native machine format.
random files	Random files are data files that support "direct access" by record number. This means that the application can go directly to the desired record in a file rather than reading through all the preceding records to get to the desired record (as would be required with a text file).

NOTE: Random files were most commonly used prior to the advent of desktop database systems such as MS-Access, but they can still be used today.

NOTE: there is also sequential file which processed sequentially line by line.

VB.NET provides functionality to process text and binary files through the **System.IO** namespace and also supports re-tooled versions of earlier file processing functions through the **Microsoft.VisualBasic** namespace. **Random files** can also be processed, but only via the MS.VB namespace.

Text File Processing

To process text files using the **System.IO namespace**, you use the **FileStream** object to refer to the file being processed, and the **StreamReader** object if processing the file for input or the **StreamWriter** object if processing the file for output. You then use the methods of the appropriate object to process the file. Methods of the StreamReader object include **Peek**, **Read**, **ReadLine**, **ReadToEnd**, and **Close**. Methods of the StreamWriter object include **Write**, **WriteLine**, and **Close**.

To process text files using the **Microsoft.VisualBasic namespace**, the **FileOpen** and **FileClose** functions are used to open and close the file, respectively. If processing the file for input, the **Input**, **LineInput**, **InputString**, and **EOF** are among the functions you might use. If processing the file for output, **Write**, **WriteLine**, **Print**, and **PrintLine** are among the functions you might use.

Random File Processing

To process random files, you must use the Microsoft.VisualBasic namespace, and the same functions that are used process binary files are used to process random files: **FileOpen**, **FileClose**, **FileGet**, and **FilePut**. However, the arguments to these functions in this case will be appropriate to random file processing.

The FileStream Class

The FileStream class is used to create an object that will reference the file that will be processed by the program. The syntax for declaring a FileStream object, as it will be used in the examples that follow, is:

Dim *variable* As New FileStream(*path, mode, access [,share]*)

Where

Table 8.2

path	is a string that refers to the full path and filename of the file to be processed. As in the example above, you can use a reference like **My.Application.Info.DirectoryPath & "\somefile.txt"** or any other path/file

reference, such as **"C:\SomeDirectory\SomeFile.dat"**.

mode is an enumeration that specifies how the file should be open or created. The possible values are:

FileMode.Append	Opens the file if it exists and starts writing new data at the end of the file (preserving what is already there). If the file does not exist, it is created.
FileMode.Create	Creates a new file; if the file already exists, it is overwritten.
FileMode.CreateNew	Creates a new file; if the file already exists, an exception is thrown.
FileMode.Open	Opens an existing file; if the file does not exist, an exception is thrown.
FileMode.OpenOrCreate	Opens a file if it exists, or creates a new file if it does not exist.
FileMode.Truncate	Opens an existing file and truncates it (i.e., deletes any data that was previously there).

access is an enumeration that specifies how the file can be accessed. The possible values are:

FileAccess.Read	Data can be read from the file, but not written to it.
FileAccess.ReadWrite	Data can be read from and written to the file.
FileAccess.Write	Data can be written to the file, but not read from it.

share is an enumeration that specifies restrictions on how other processes can access the file. The possible values are:

FileShare.None	Other processes may neither read from nor write to the file
FileShare. ReadWrite	Any process can read from or write to the file.
FileShare.Write	Other processes may write to the file.
FileShare.Read	Other processes may read from the file.

The StreamWriter Class

In Visual Basic, you use a StreamWriter object to write a stream of characters to a sequential access file.

Declaring a StreamWriter variable

Syntax

{**Dim** | **Private**} *streamWriterVariableName* **As IO.StreamWriter**

Example14

```
Dim outFile As IO.StreamWriter
```

declares a StreamWriter variable named outFile

Creating a StreamWriter object

Syntax

IO.File.*method***(***fileName***)**

Table 8.3

Method	**Description**
CreateText	opens a sequential access file for output
AppendText	opens a sequential access file for append

Example 15

outFile = IO.File.CreateText("F:\Chap22\pay.txt")
opens the pay.txt file for output; creates a StreamWriter object and assigns it to the outFile variable

Example 16

outFile = IO.File.AppendText("report.txt")
opens the report.txt file for append; creates a StreamWriter object and assigns it to the outFile variable

After opening a file for either output or append, you can begin writing data to it. You write data to a sequential access file using either the Write method or the WriteLine method.

Writing data to a sequential access file

Syntax

*streamWriterVariableName***.Write(***data***)**
*streamWriterVariableName***.WriteLine(***data***)**

Example 17

outFile.Write("Hello")

Result

Hello|

the next character will appear immediately after the letter o

Example 18

outFile.WriteLine("Hello")

Result

Hello
|

the next character will Appear on the next line

*The **syntax** to close an output sequential access file is

streamWriterVariableName.Close().

The StreamReader Class

The **StreamReader** class is used to read a stream of characters. In the .NET framework, a **stream**, in general terms, is the flow of data from one location to

another. In the case of these examples, a stream refers to the text file that will be processed by the sample programs. The syntax for declaring a StreamReader object is:

Dim *variable* As New StreamReader(*stream*)

Where

stream is an object representing the stream of characters to be read (in this case a text file)

Declaring a StreamReader variable

Syntax

{**Dim** | **Private**} *streamReaderVariableName* **As IO.StreamReader**

Example 19

Dim inFile As IO.StreamReader

declares a StreamReader variable named inFile

Creating a StreamReader object

Syntax

IO.File.OpenText(*fileName***)**

Example 20

inFile = IO.File.OpenText("report.txt")
opens the report.txt file for input; creates a StreamReader object and assigns it to the inFile variable.

Determining whether a sequential access file exists

Syntax

IO.File.Exists(*fileName***)**

Example 21

If IO.File.Exists("report.txt") = True Then
determines whether the report.txt file exists in the current project's bin\Debug folder; you also can write the If clause as If IO.File.Exists("report.txt") Then

ReadLine method

Syntax

streamReaderVariableName.**ReadLine**

Example 22

strMessage = inFile.ReadLine
reads a line of text from the sequential access file associated with the inFile variable and assigns the line, excluding the newline character, to the strMessage variable.

Peek method

Syntax

streamReaderVariableName.**Peek**

Example 23

```
Do Until inFile.Peek = –1
strLineOfText = inFile.ReadLine
MessageBox.Show(strLineOfText)
Loop
```

reads each line of text from the sequential access file associated with the inFile variable, line by line; each line (excluding the newline character) is assigned to the strLineOfText variable and is then displayed in a message box

Commonly used methods of the StreamReader class are (Table 8.4):

Peek	Looks ahead to the next available character in the input stream without actually advancing to that next position. If no character is available, Peek returns -1. (This method is convenient to test for end-of-file.)
Read	Reads the next character from the input stream.
ReadLine	Reads the next line of characters from the input stream into a string.
ReadToEnd	Reads the data from the current position in the input stream to the end of the stream into a string. (This method is often used to read the entire contents of a file.)
Close	Closes the StreamReader and its associated FileStream object.

Example24: this is program to explain how to write and read from sequential file a text box for entering a name or any text. The Write to File button will write the name to a sequential access file. The Read from File button will read each name from the sequential access file and display each in the readtext box. (The textbox1 control's Multiline and ReadOnly properties are set to True, and its ScrollBars property is set to Vertical.)

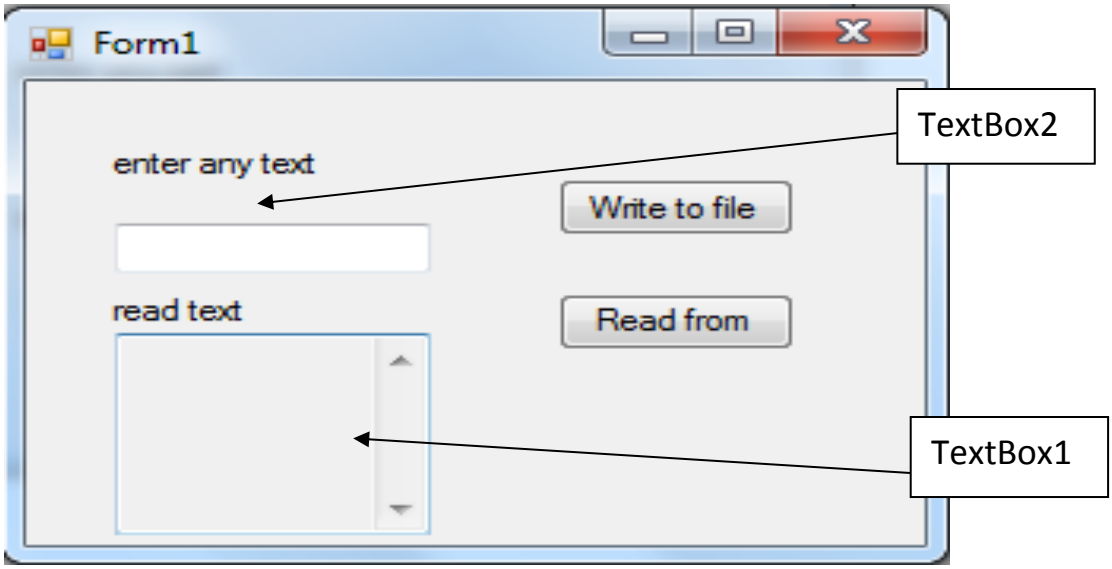


Figure 8.7

Write the following code in the write to file button:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        ' writes a name to a sequential access file
        ' declare a StreamWriter variable
        Dim outFile As IO.StreamWriter
        ' open the file for append
        outFile = IO.File.AppendText("contestants.txt")
        ' write the name on a separate line in the file
        outFile.WriteLine(TextBox2.Text)
        ' close the file
        outFile.Close()
        ' clear the Name box and then set the focus
        TextBox2.Text = String.Empty
        TextBox2.Focus()

    End Sub
```

Write the following code in the read from the file button:

```
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
        ' reads names from a sequential access file
        ' and displays them in the interface
        ' declare variables
        Dim inFile As IO.StreamReader
        Dim strName As String
        ' clear previous names from the Contestants box
        TextBox1.Text = String.Empty
        ' determine whether the file exists
        If IO.File.Exists("contestants.txt") = True Then
            ' open the file for input
            inFile = IO.File.OpenText("contestants.txt")
            ' process the loop instructions until the end of the file
            Do Until inFile.Peek = -1
                ' read a name
                strName = inFile.ReadLine
                ' display the name
                TextBox1.Text = TextBox1.Text & strName & ControlChars.NewLine
            Loop
            ' close the file
            inFile.Close()
        Else
```

```
        MessageBox.Show("Can't find the contestants.txt file", "Game Show
Contestants", MessageBoxButtons.OK, MessageBoxIcon.Information)
    End If

End Sub
```

The **MS.VB namespace functions** are summarized below **(Table 8.5),** followed by more detailed explanations.

Table (8.5)

Function	Description
FreeFile	Returns an **Integer** value representing the next file number available for use by the **FileOpen** function.
FileOpen	Opens a file for input or output. (Analagous to the **Open** statement in classic VB.)
Input	Reads a comma-delimited field from a text file and assigns it to a variable. (Analagous to the **Input** statement in classic VB.)
LineInput	Reads a single line from an open sequential file and assigns it to a **String** variable. (Analagous to the **Line Input** statement in classic VB; also analagous to the **ReadLine** method of the StreamReader object.)
InputString	Reads a specified number of characters from a text or binary file into a **String** variable. (Analagous to the **Input** function in classic VB; similar to the **ReadToEnd** method of the StreamReader object.)
EOF	Returns a Boolean value indicating whether or not the end of a file opened for input or random access has been reached.
Write and **WriteLine**	Writes data to a text file. Automatically writes the data in comma-delimited format, enclosing string fields in quotes when necessary. (Analagous to the **Write** statement in classic VB.)
Print and **PrintLine**	Writes data to a text file. (similar to the **WriteLine** method of the StreamWriter object, but with additional functionality.)
FileClose	Closes an open file.

We will now explore some of these functions in more detail.

FileOpen

Syntax:

FileOpen(FileNumber, FileName, Mode [, Access [, Share]])

The parameters for **FileOpen** are as follows:

FileNumber Required. Any valid file number.

***FileName*:** Required. **String** expression that specifies a valid file name — may include directory or folder, and drive.

***Mode*:** Required. Enum specifying the file mode. Possible values are:

Table 8.6

OpenMode.Append	Opens a file for output (writing). If the file does not exist, it will be created; if it does exist, records will be added to the file after the last record in the file
OpenMode.Input	Opens a file for input (reading). The specified file must exist.
OpenMode.Output	Opens a file for output (writing). If it does not exist, it will be created; if it does exist, its previous contents will be overwritten.

Access Optional. Enum specifying the operations permitted on the open file. These

parameters is restrict the user ability to change the file by prevent him from

writing, just reading or make him able to read and write and so on.

***Share*:** Optional. Enum specifying the operations restricted on the open file by other processes. These parameters used in network environment to decide whether other processes can fully reach(read or write) the file, or we can lockwrite(prevent user from write on file mean make it readonly), or lockread(user can only able to write and so on.

Example 25:

```
FileOpen(1,"C:\ProgramFiles\EmpMaint\EMPLOYEE.TXT", penMode.Input)
```

FreeFile

function to supply you with a file number that is not already in use by the system. To use it, declare an integer variable, then assign FreeFile to it, as follows:

```
Dim intEmpFileNbr As Integer

intEmpFileNbr = FreeFile
```

In the Open statement (and any other statement that refers to this file), use the integer variable For example:

FileOpen(intEmpFileNbr, "C:\Program Files\EmpMaint\EMPLOYEE.TXT", OpenMode.Input)

FileClose

Close the opened file and free all system resources used by this file

Syntax: fileclose(filenumber)

Example 26: to open text file name it employee1, to write employee name and employee number, then read the content into TextBox. Make properties multiline for the text3 =true, readonly, and vscroll.

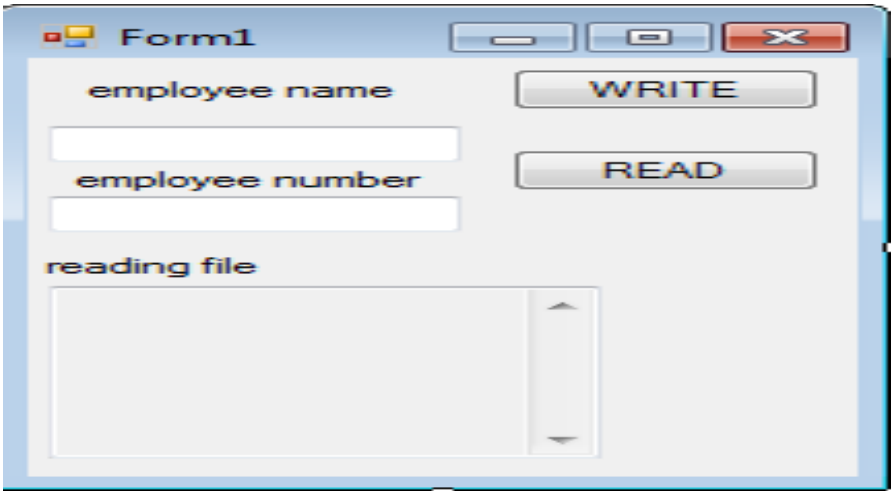


Figure 8.8

Here is the code for write button

```
Dim intEmpFileNbr As Integer

intEmpFileNbr = FreeFile()
    FileOpen(intEmpFileNbr, "D:\employee1.txt", OpenMode.Append)
    WriteLine(intEmpFileNbr, TextBox1.Text, Val(TextBox2.Text))

    FileClose(intEmpFileNbr)
```

Write in Read button this code:

```
Dim inred As Integer
```

```
inred = FreeFile()
    FileOpen(inred, "D:\employee.txt", OpenMode.Input)
    TextBox3.Text = LineInput(inred)

    FileClose(inred)
```

Procedure

A- Built class to calculate the number of square yards needed for some room with the cost of the required carpet. **Hint:** you can use A Class that Contains Private Variables, Public Properties, and Methods if you want. Pseudocode for the Calculate button's Click event procedure

1. Instantiate a Rectangle object to represent the floor
2. Declare variables to store the price per square yard, required number of square yards, and carpet cost
3. Assign the input data to the appropriate properties and variable
4. Calculate the required number of square yards by dividing the floor's area by 9
5. Calculate the carpet cost by multiplying the price per square yard by the required number of square yards
6. Display the required number of square yards and the carpet cost

B- Repeat the geometry subroutine by using subroutine built in module then call subroutine in the form window.

C- Write VB.NET program to write and read from file usin readtoend method with other methods you learned from this lecture.

Discussion

1- Repeat example24 with using MS.VB namespace functions.
2- What are the differences between the Classes with Standard Modules.

Experiment No. (9)

DATABASES in Visual Basic.NET

Theory

What Is a Database?

A *database* is a container for storing relational, structured information. A relational database is one that stores information in tables composed of columns and rows. What makes a database unique is that it is designed to preserve relationships and make data easily retrievable.

A field is a single item of information about a person, place, or thing—such
as a name, a salary amount…etc.

A record is a group of related fi elds that contain all of the necessary data about a specifi c person, place, or thing.

A Table is a group of related records.

DBMS

Databases are maintained by special programs, such as Microsoft Office Access and SQL Server, oracle…etc. These programs are called *database management systems* (DBMSs). data organized in tables with relationships between tables.

SQL

To access or update the data stored in the database, you use a special language, the Structured Query Language (SQL).all major DBMSs support it. SQL is a **nonprocedural** language, which means that SQL doesn't provide traditional programming structures such as If statements or loops. Instead, it's a language for specifying the operation you want to perform against a database at a high level.

NOTE: By the way, the SQL version of SQL Server is called T-SQL, which stands for Transact-SQL.T-SQL is a superset of SQL and provides advanced programming features that are not available with SQL.

NOTE: you can use Access as well as non-Microsoft databases such as Oracle as DBMS engines.

NORMALIZATION

The reason for breaking the information we want to store in a database into separate tables is to avoid duplication of information The process of breaking the data into related tables that eliminate all possible forms of information duplication is called *normalization*.

Example1: if you have table of authors and their books you really have the same author with more books names, so you need to break the information to two tables one for authors and the other for books then make relationship between the two tables.

***A** relational database can contain one or more tables **for example** we can build database call it dbauthor which contain two tables one call it authors and the other is books, then we can make relationship between the two tables by some unique field has the same name and same data type in the two tables but in the authors table this field must has **primary key** and in the books table the field has **foreign key,** also authors table call it the **parent**(or master) table, but the books table call it the **child**(or the details) table.

Example2: in this example will learn how to built database give it name dbauthor and two tables(authors and books) by using **Microsoft office access as DBMS.**

1-open Microsoft office access, then choose blankdatabse option, then name the database (dbauthor) the file name space, then click the create button.
2-point the table1 by the mouse then choose design view in the tool bar, finally name the table authors.
3-create three fields (id,authorname,title), and make the datatype of the id field is autonumbering which will be the primary key field.

Primary key

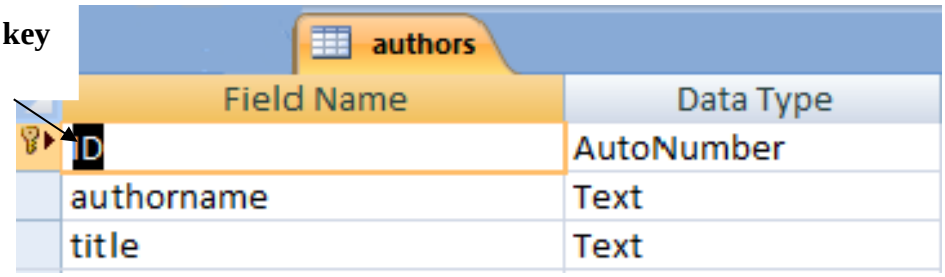


Figure 9.1

4- now built the second table, open create menu, choose table in the tool bar of the create menu, now point the table2 by the mouse then click design view from the tool bar.
5- create five fields(id, booktitle, pages, price, bookdate), before closing the table put the mouse on the id field and right click on the primary key icon to remove this key because we shall make this field as foreign key, make datatype of this id field is autonumber like id field in the author table.

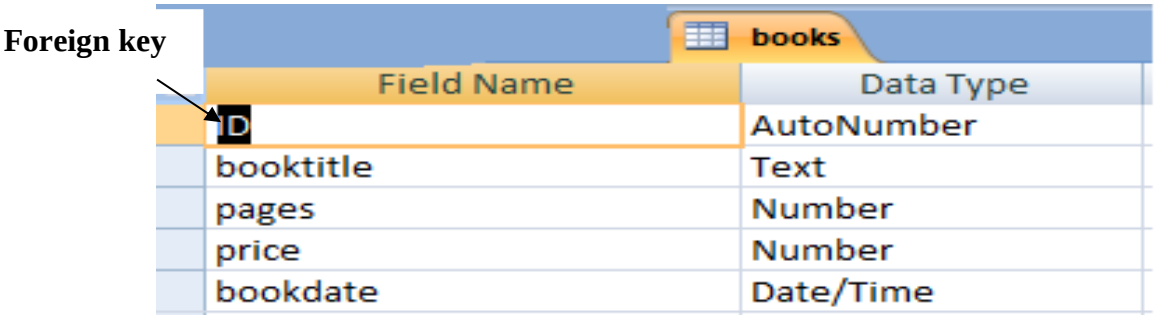


Figure 9.2

6-open database tools menu, then choose relationships from tool bar of the menu, click on authors and books tables and click add.

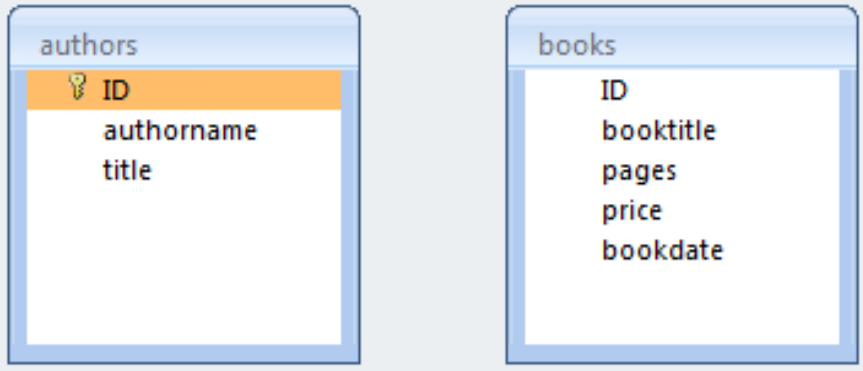


Figure 9.3

7-now establishing the relationship by continuous click on primary key id field in author table to id field of books table you shall get the following dialogue box:

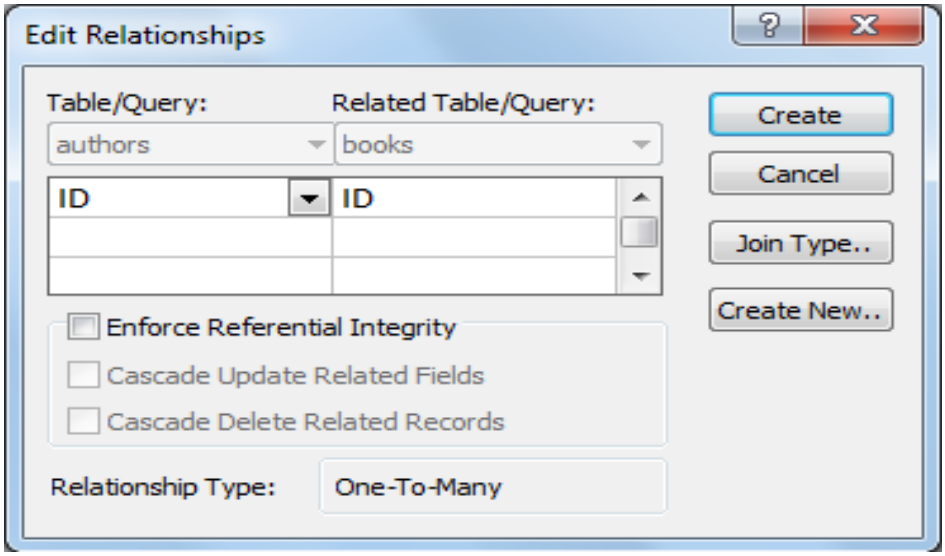


Figure 9.4

8- click on create button you will get the following relationship diagram:

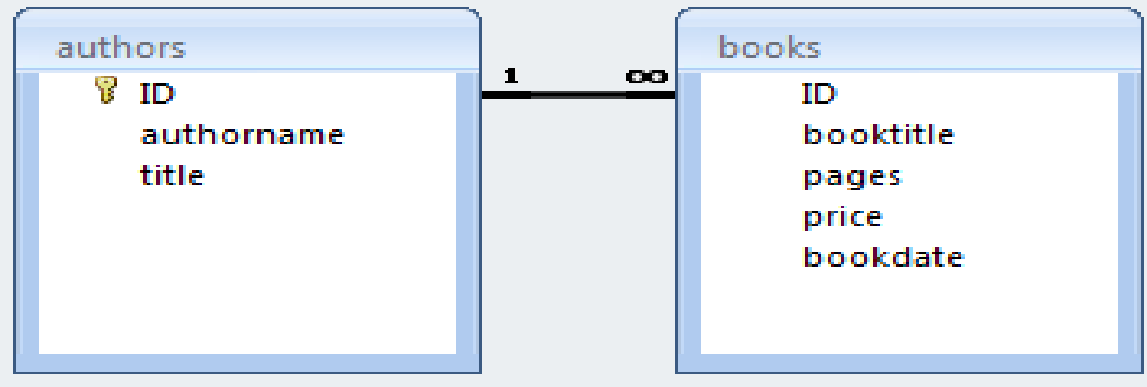


Figure 9.5

Visual Database Tools

To simplify the development of database applications, Visual Studio 2010 comes with some visual tools, the most important of which are briefly described in **Table9.1** and then discussed in the following sections.

Table 9.1: Visual database tools

Name	Description
Server Explorer	This is the most prominent tool. Server Explorer is the toolbox for database applications, in the sense that it contains all the basic tools for connecting to databases and manipulating their objects.
Query Builder	This is a tool for creating SQL queries (statements that retrieve the data you want from a database or update the data in the database). SQL is a language in its own right, and we'll discuss it later in this chapter. Query Builder lets you specify the operations you want to perform on the tables of a database with point-and-click operations. In the background, Query Builder builds the appropriate SQL statement and executes it against the database.
Database Designer And Tables Designer	These tools allow you to work with an entire database or its tables. When you work with the database, you can add new tables, establish relationships between the tables, and so on. When you work with individual tables, you can anipulate the structure of the tables, edit their data, and add constraints. You can use these tools to manipulate a complicated object — the database — with point-and-click operations.

Connecting VB Application to a Microsoft Access Database

Open Visual Studio, and choose View menu _ Server Explorer to display Server Explorer window. Right-click Data Connections, and choose Add Connection from the context menu. Server Explorer will display the Add Connection dialogbox shown in Figure below.
If Microsoft Access Database File (OLE DB) does not appear in the Data source box, click the **Change** button to open the Change Data Source dialog box, click **Microsoft Access Database File**, and then click the **OK** button.
Click the **Browse** button in the Add Connection dialog box. Open the **C:\users\computer\mydocuments** then click **dbauthors.**
accdb in the list of file names. Click the **Open** button. Figure below shows the completed Add Connection dialog box.

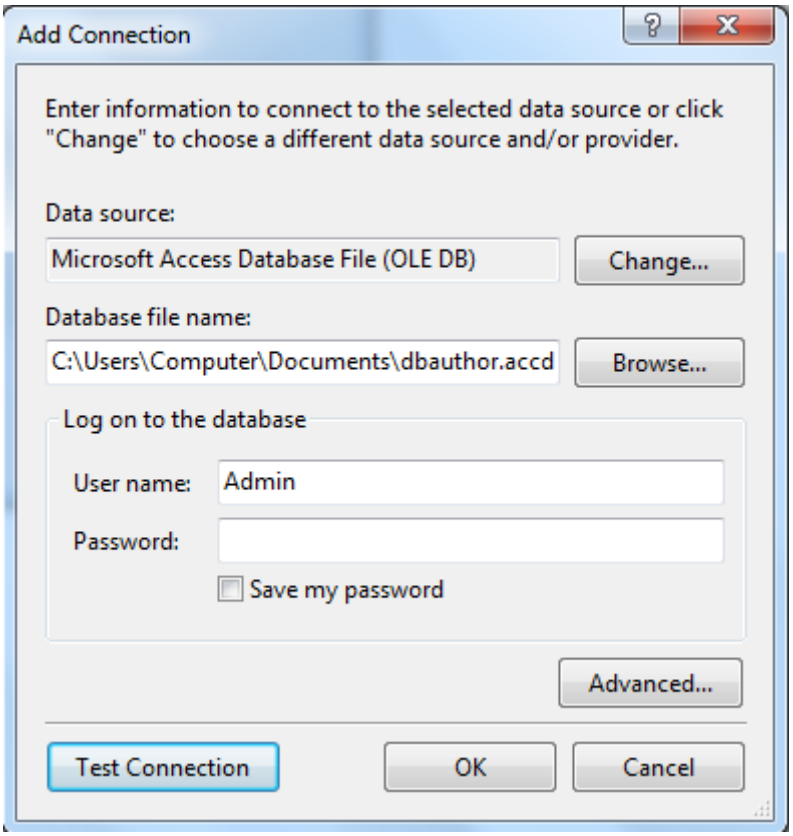


Figure 9.6

Now click **test** connection button, then click ok.

Note: Visual Basic Express Edition 2008 limits the data files that are accessible to SQL Server and Microsoft Access files. Visual Basic Professional Edition enables access to all data sources. Also VB2010 is best choice.

Very Important Note: if you want Using SQL Server Management Studio

One of the applications installed with SQL Server is SQL Server Management Studio. To start it, choose Start _ Programs _ SQL Server _ SQL Server Management Studio.

When this application starts, you see the Connect To Server dialog box(as in figure above). Choose Database Engine in the Server Type field so you can work with databases on your system. Select the server you want to use in the Server Name field. Provide your credentials, and click Connect.

In order to access the data stored in a database, an application needs to be connected to the database. The computer makes a copy of the specified data and stores the copy in its internal memory. The copy of the data you want to access is called a **dataset**.

now to make a **dataset** for the current application:

click **Data** menu, then choose **show data sources,** the data sources window will appear on the right of the project screen.
Click **Data** menu, then choose **add new data source,** dialogue box will appear, click **Database** then click **Next.**
Now if **dbauthor.accdb** not appears in the **Choose Your Data Connection**
Screen, then you must click the **Newconnection** button to specify the database location that you will use it as datasource in your project(application).

Click **Next,** then the following message will appear

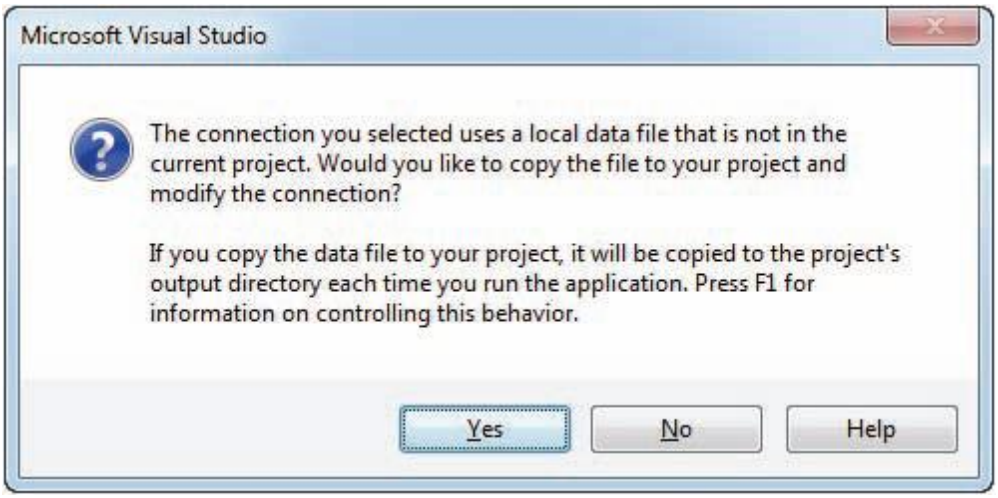


Figure 9.7

Click **Yes** to add the dbauthor.accdb file to the application's project folder.

Now Click the **Next** button to display the **Choose Your Database Objects** screen. You use this screen to select the table and/or field objects to include in the dataset, which is automatically named dbauthorDataSet.

Finally check all the tables and their fields as following:

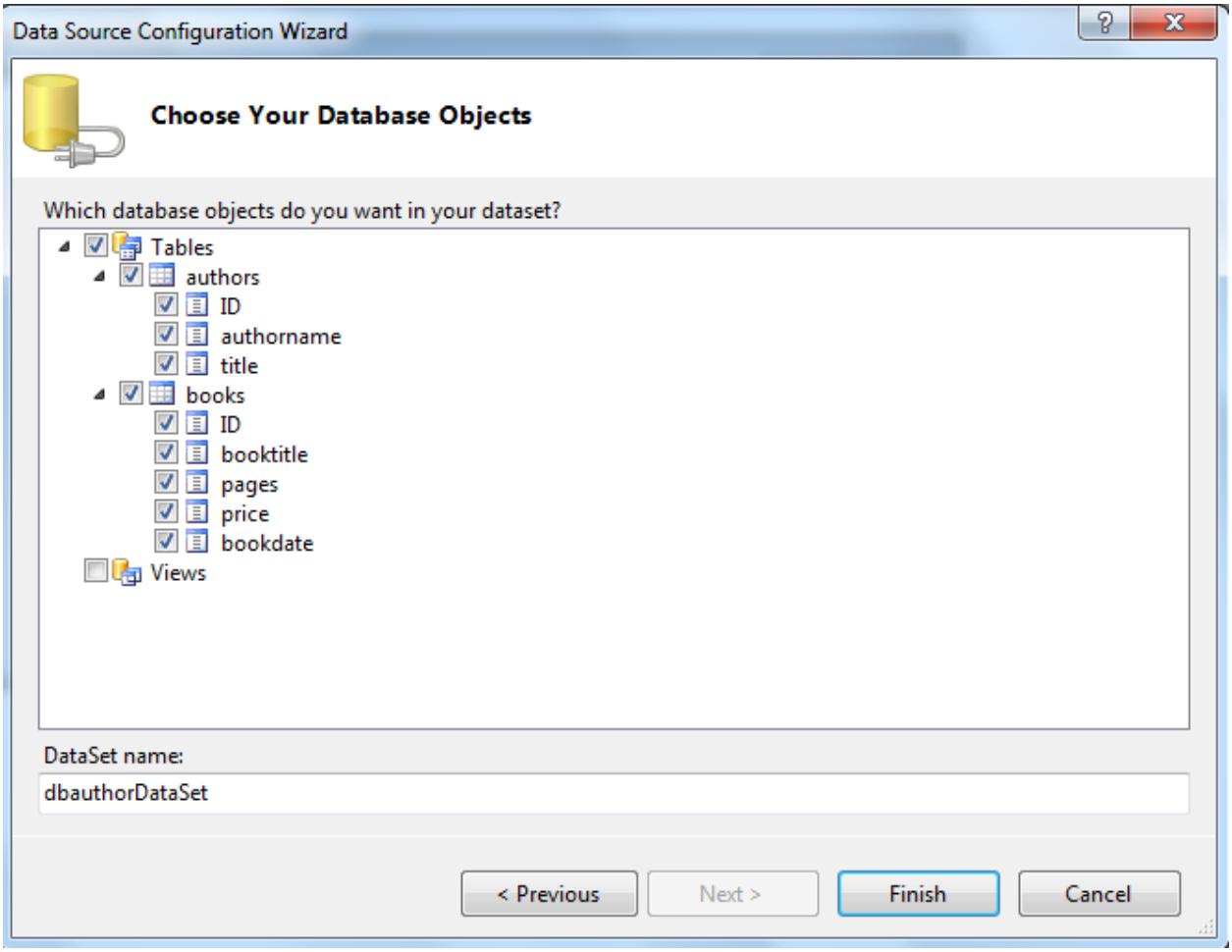


Figure 9.8

Click the **Finish** button. The computer adds the dbauthorDataSet to the Data Sources as shown below

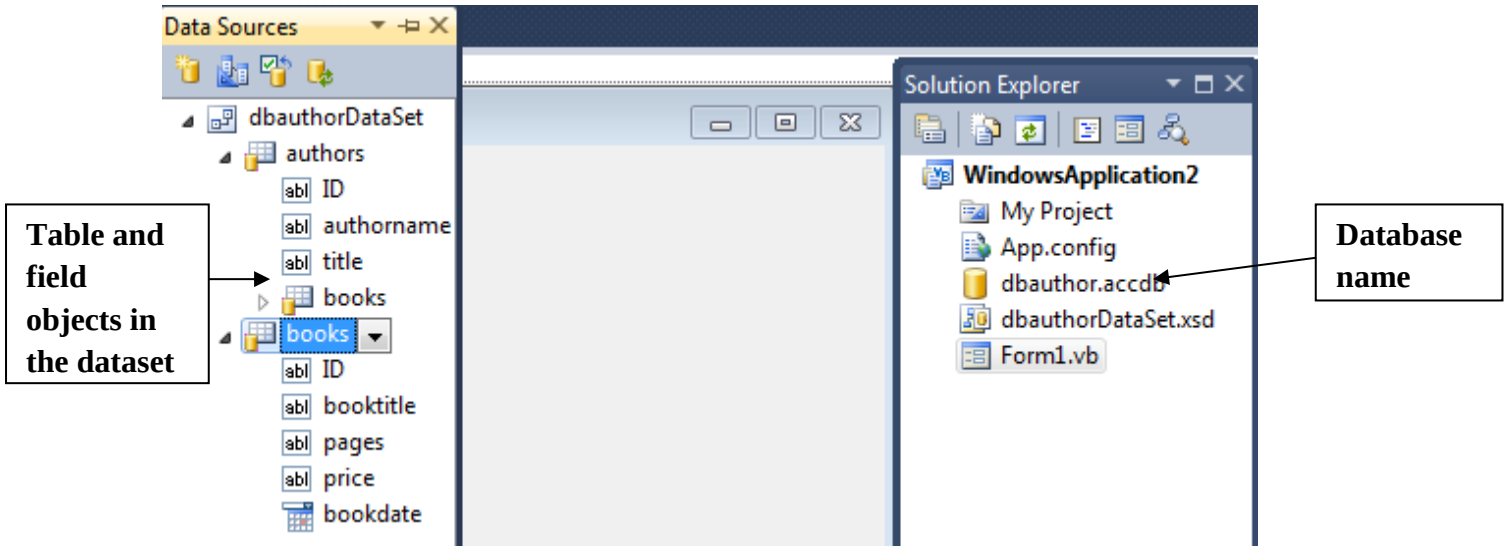


Figure 9.9

To view the contents of the dbauthorDataSet:

1. Click the **form** to make it the active window. Click **Data** on the menu bar and then click **Preview Data** to open the Preview Data dialogbox.
2. Click the **Preview** button. As Figure shown below

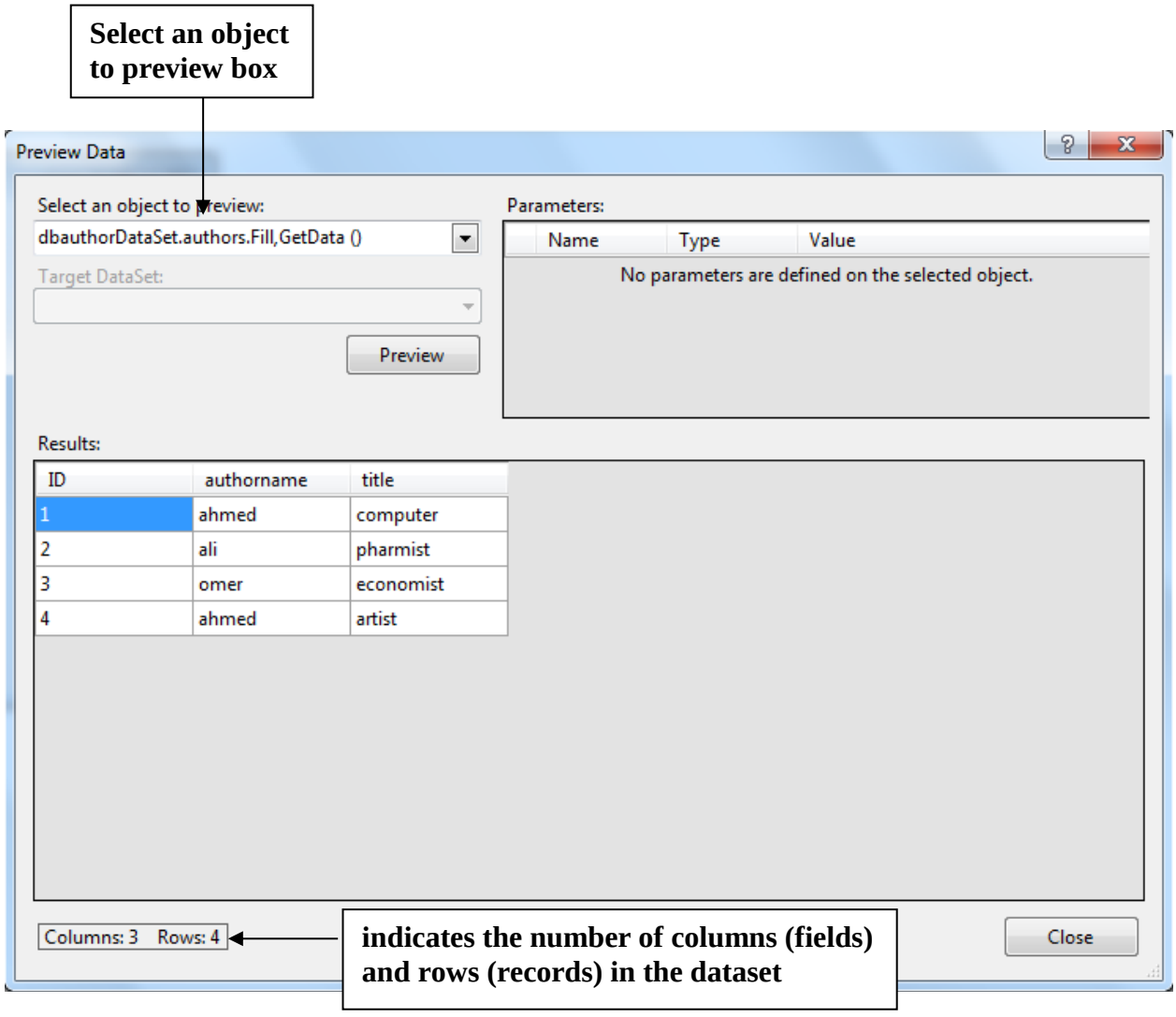


Figure 9.10

Binding the Objects in a Dataset

For the user to view the contents of a dataset while an application is running, you need to connect one or more objects in the dataset to one or more controls in the interface. Connecting an object to a control is called **binding**, and the connected controls are called **bound controls.**

To bind an object to an existing control:

you can click the control on the form and then use the Properties window to set the appropriate property or properties. (Refer to the *Binding to an Existing Control* section in this lesson.)
For example, you use the DataSource property to bind a DataGridView control. However, you use the DataSource and DisplayMember properties to bind a ListBox control. To bind label and textbox controls, you use the DataBindings/Text property.

NOTE: another method for binding by dragging the objects.
In the Data Sources window, click the object you want to bind. Drag the object to the control on the form and then release the mouse button.

A **DataGridView control** displays the table data in a row and column format, similar to a spreadsheet. Each row in the control represents a record, and each column represents a field.

NOTE:also we can use textboxes to binding them to fields of the table without using DataGridView control.

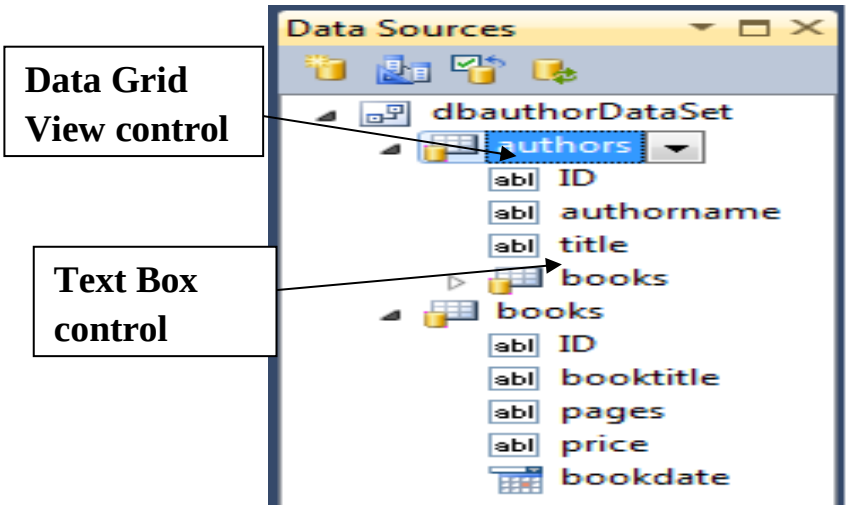


Figure 9.11

Drag the **authors** object from the **Data Sources** window to the form and then release the mouse button. The computer adds a **DataGridView** control to the form, and it binds the **authors** object to the control. See Figure below (Result of dragging the table(authors) object to the form)

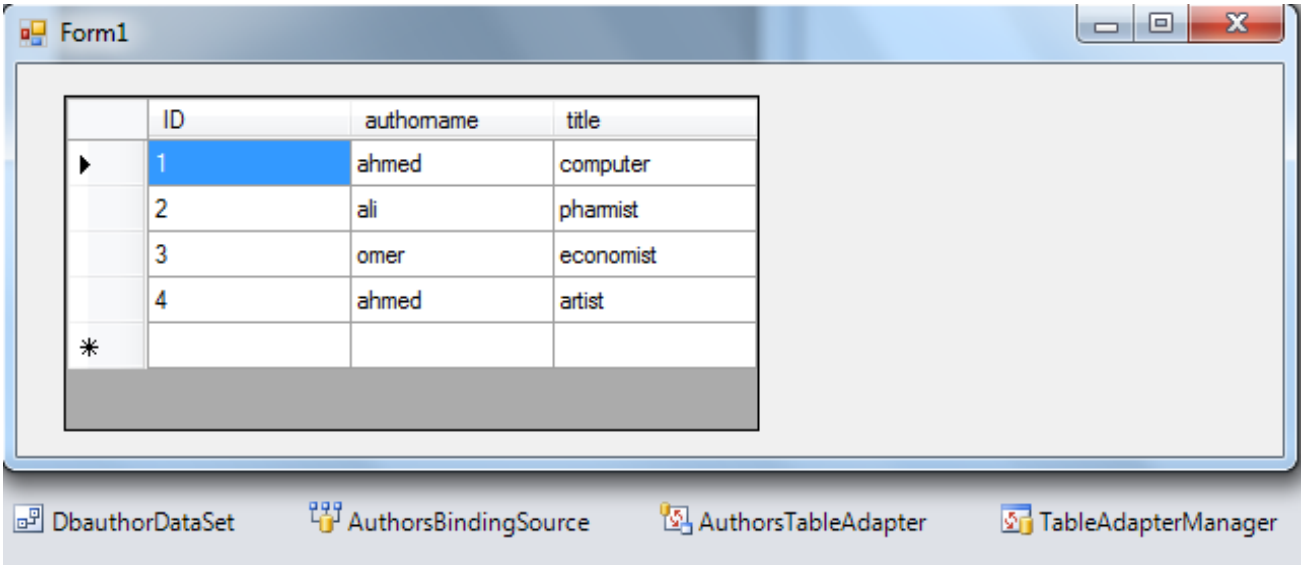


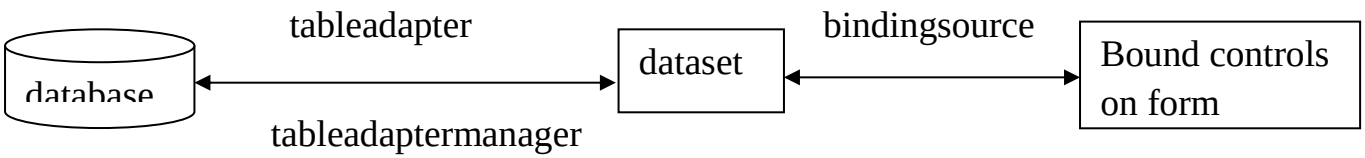
Figure 9.12

We see in the bottom of the above window the following three objects:

The **TableAdapter object:** is responsible for retrieving the appropriate information from the database and storing it in the DataSet.

TableAdapterManager object to save the changes, because it can handle saving data to multiple tables in the DataSet.

BindingSource object provides the connection between the DataSet and the bound controls on the form.



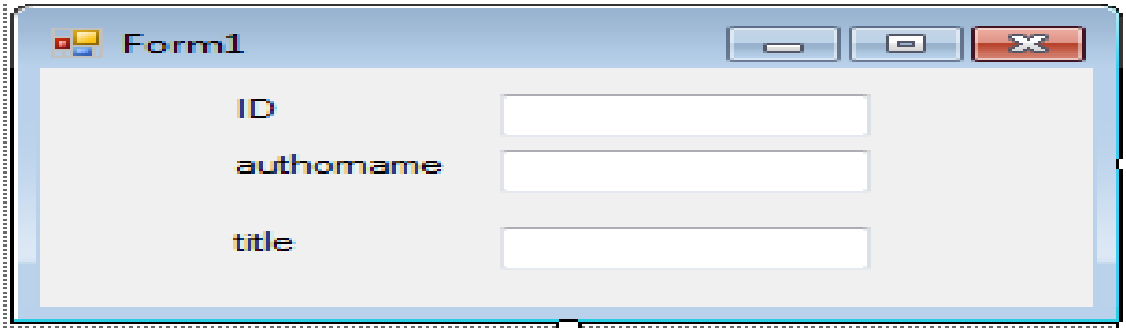


Figure 9.13

Moving between records

BindingSource object stores the position of the record pointer in its **Position property**. The first record is in position 0

Syntax

bindingSourceName.**Position**

Example 3

intRecordNum = BindingSourcename.Position
assigns the current record's position to the intRecordNum variable

Example 4

authorsBindingSource.Position = 4
moves the record pointer to the fifth record in the dataset

BindingSource object's Move methods. The **Move methods** move the record pointer to the first, last, next, or previous record in the dataset.

Syntax

bindingSourceName.**MoveFirst()**
bindingSourceName.**MoveLast()**
bindingSourceName.**MoveNext()**
bindingSourceName.**MovePrevious()**

Example 5

authorsBindingSource.MoveFirst()
.

VERY IMPORTANT NOTE: if DBMS is SQL server, then To connect the application to a database, the Connection object must know the name of the server on which the database resides, the name of the database itself, and the credentials that will allow it to establish a connection to the database. These credentials are either a user name and password or a Windows account that has been granted rights to the database.

In the Server Name field, type the name of your machine, or select one with the mouse. Click the down arrow in the Select Or Enter A Database Name field.

Executing SQL Statements with access database

SQL is a universal language for manipulating data in database tables. Every DBMS supports it. SQL is a nonprocedural language, which means that SQL doesn't provide traditional programming structures such as If statements or loops.

If you are not familiar with SQL, I suggest that you follow the examples in this experiment and experiment with the sample databases. To follow the examples, you have two options:

4- SQL Server Management Studio (**SSMS**) (**used with SQL-server**). SSMS helps you manage databases in various ways, including creating queries to extract data.
5- and **Query Designer** of **Visual Studio**. Query Designer is an editor for SQL statements that also allows you to execute them and see the results.

NOTE: In addition to Query Designer, you can also use **Query Builder**, which is part of SSMS and Visual Studio. Query Builder lets you build the statements with visual tools, and you don't have to know the syntax of SQL in order to create queries with Query Builder.

Using Visual Studio

To execute the same queries with Visual Studio, open the Server Explorer window, and right-click the name of the database against which you want to execute the query. From the context menu, choose New Query, and a new query window will open.then choose the table name you want to add then click Add button, finally The Query builder will appear as in figure shown below.

The Query builder has four panes:

Diagram pane: displays the tables involved in the query, their fields, and the relationships between the tables — if any.

Gride Pane: shows the fields that will be included in the output of the query. This pane is Query Builder, the tool that lets you design queries visually.

the SQL pane, you see the SQL statement produced by the visual tools.

the Results pane, contains a grid with the query's output.

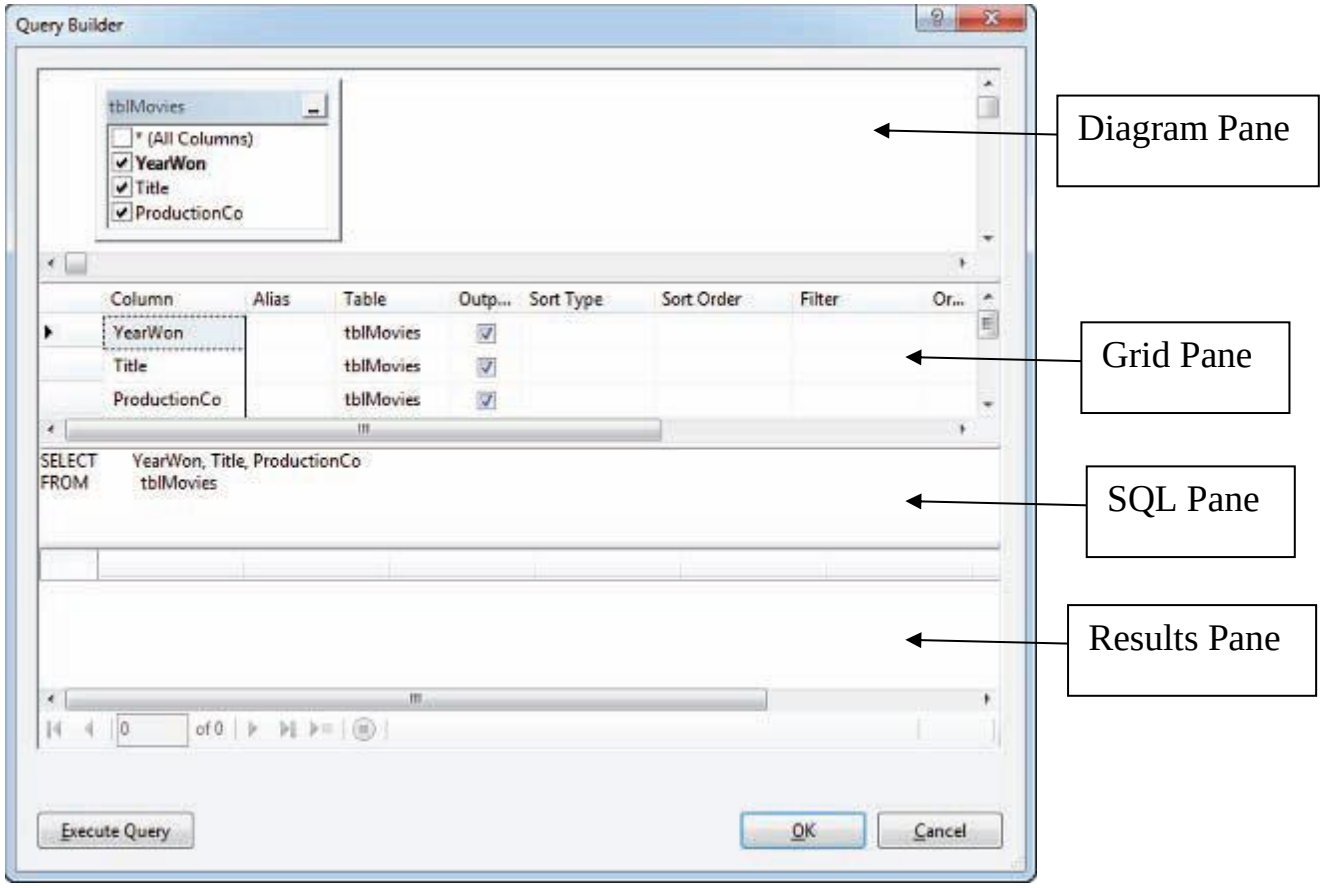


Figure 9.14

Now we shall using the authors and books tables of the database dbauthor to execute the SQL statements.

SELECT query

To retrieve the fields of some table we write

SELECT feildsnames FROM tablename

Example 6 to select all the fields of the authors table write the following in the SQL pane:

SELECT * FROM authors

Then click Execute Query Button. You shall see the result in the results pane.

Limiting the Selection with *WHERE*

To limit the selection with some criteria, like select all records with field value=100.

Example 7: if we want to retrieve authors that have the name ali of the field (authorname), write in the SQL pane the following

SELECT * FROM authors WHERE authorname = 'ali'

to combine multiple conditions we can use OR, AND.

Example 8: to retrieve the fields with author names ali or omer, write the following:

SELECT * FROM authors WHERE authorname= ‘ali’ OR ‘omer’

Combining Data from Multiple Tables

It is possible to retrieve data from two or more tables by using a single statement
To retrieve all fields from the tables (authors and books) write the following:
SELECT * FROM authors, books

Skipping Duplicates with *DISTINCT*

The DISTINCT keyword eliminates any duplicates retrieved by the SELECT statement. For example if the authors table has more than one author have the same name (in our table we have two authors have the name ahmed), to retrieve all records in the field authorname without duplicating, write the following in the SQL pane:

SELECT DISTINCT authorname FROM authors

The *LIKE* Operator

The LIKE operator recognizes several pattern-matching characters to match one or more characters, numeric digits, ranges of letters, and so on. Below is some characters used with LIKE keyword.

Table 9.2

% Matches any number of characters. The pattern program% will find *program*, *programming*, *programmer*, and so on. The pattern %program% will locate strings that contain the words *program, programming, nonprogrammer*, and so on.

_ (Underscore character.) Matches any single alphabetic character. The pattern b_y will find *boy* and *bay*, but not *boysenberry.*

[ ] Matches any single character within the brackets. The pattern Santa [YI]nez will find both *Santa Ynez* and *Santa Inez.*

[^] Matches any character not in the brackets. The pattern %q[^ u]% will find words that contain the character *q* not followed by *u* (they are misspelled words).

[-] Matches any one of a range of characters. The characters must be consecutive in the alphabet and specified in ascending order (*A* to *Z*, not *Z* to *A*). The pattern [a-c]% will find all words that begin with *a, b,* or *c* (in lowercase or uppercase).

Matches any single numeric character. The pattern D1## will find *D100* and *D139*, but not *D1000* or *D10.*

Example 9: to retrieve all records that authorname field begin with letter **A**

SELECT * FROM authors WHERE authorname LIKE '%a%'

Null Values and the *ISNULL* Function

A common operation for manipulating and maintaining databases is to locate null values in
fields. The expressions IS NULL and IS NOT NULL find field values that are (or are not) null.

Example 10: SELECT * FROM authors WHERE authorname IS NOT NULL

Sorting the Rows with *ORDER BY*

The rows of a query are not in any particular order. To request that the rows be returned in a
specific order, use the ORDER BY clause, which has this syntax:
ORDER BY col1, col2, . . .
You can specify any number of columns in the ORDER BY list. The output of the query is
ordered according to the values of the first column.

Example 11: retrieve all records ordered according to the authorname field
SELECT * FROM authors ORDER BY authorname

Working with Calculated Fields

In addition to column names, you can specify calculated columns in the SELECT statement.

Example 12: if we assume each author sale 2 books, then to calculate the totalsum of each edition and put the result in calculated column name it totalsum.
SELECT books.price * 2 AS totalsum FROM books

Calculating Aggregates

SQL supports some aggregate functions, which act on selected fields of all the rows returned by the query. The basic aggregate functions listed below

Table 9.3

Function	Returns
COUNT()	The number (count) of values in a specified column
SUM()	The sum of values in a specified column
AVG()	The average of the values in a specified column
MIN()	The smallest value in a specified column
MAX()	The largest value in a specified column

Example13: to get the highest price of books
SELECT MAX(price) FROM books
To get the highest price of the books of authorname ahmed (ID=1)

SELECT MAX(price) FROM books WHERE (ID = 1)

Grouping Rows

The GROUP BY clause groups all the rows with the same values in the specified column and forces the aggregate functions to act on each group separately.

Example 14: if you want sum all prices of books for each author, then the prices summed according to the grouped ID as following:
SELECT ID, SUM(price) FROM books GROUP BY ID

Limiting Groups with HAVING

The HAVING clause limits the groups that will appear at the cursor. In a way, it is similar to the
WHERE clause, but the HAVING clause is used with aggregate functions and the GROUP BY clause,
and the expression used with the HAVING clause usually involves one or more aggregates.

Example 15: to return all ID's that have sum(price) more than 100
SELECT ID, SUM(price) FROM books GROUP BY ID HAVING (SUM(price) > 100)

Deleting Rows

The DELETE statement deletes one or more rows from a table; its syntax is as follows:

DELETE table_name WHERE criteria

Example 16: to delete books edited before 1980 write the following:
DELETE books WHERE bookdate < '1/1/1980'

Inserting New Rows

The INSERT statement inserts new rows in a table; its syntax is as follows:

INSERT table_name (column_names) VALUES (values)

Example 17: to insert values to some fields:
INSERT tablename (field1,field2,…) VALUES (valu1,valu2,…)

NOTE: If the values come from a table, you can replace the VALUES keyword with a SELECT statement:
INSERT INTO SelectedProducts
SELECT * FROM Products WHERE CategoryID = 4

Editing Existing Rows

The UPDATE statement edits a row's fields; its syntax is the following:
UPDATE table_name SET field1 = value1, field2 = value2,. . .

WHERE criteria

Example 18:to change the authorname of the authors table where ID=4 (old value=ahmed, new value=naji)

UPDATE authors SET authorname = 'naji' WHERE (ID = 4)

LINQ Queries

LINQ stands for *Language Integrated Query*, Perform simple LINQ queries.

Whereas SQL can be used only with database tables, LINQ can be applied to collections, XML files, and SQL tables. It's a uniform language for querying data from several data sources. And it is part of all .NET languages. The LINQ query starts after the equals sign with the From keyword, which is followed by a variable that represents the current item in the collection

NOTE: Notice that the syntax does not require you to specify the data type of the variable in the Dim statement. Instead, the syntax allows the computer to infer the data type from the value being assigned to the variable, so do this by entering the **Option Infer On** statement in the **General Declarations** section of the Code Editor window.

Basic LINQ syntax and examples for selecting and arranging records in a dataset

Using LINQ to select and arrange records in a dataset

Basic syntax

Dim *variableName* **= From** *elementName* **In** *dataset.table*
[**Where** *condition*]
[**Order By** *elementName.fieldName1* [**Ascending** | **Descending**]
[**,** *elementName.fieldNameN* [**Ascending** | **Descending**]]]
Select *elementName*

Example 19

Dim records = From authors In DbauthorDataSet.authors Select authors
selects all of the records in the dataset

Example 20

Dim records = From authors In DbauthorDataSet.authors
Order By authors.authorname Select authors
selects all of the records in the dataset and arranges them in ascending order by the authorname field

Example 21

Dim records = From authors In DbauthorDataSet.authors
Where authors.authorname.ToUpper Like "a*"
Order By employee.title Descending
Select authors
selects from the dataset only the authors records whose name begins with the

letter a, and arranges them in descending order by the title field

Assigning a LINQ variable's contents to a BindingSource object

Basic syntax

bindingSource.**DataSource** = *variableName*.**AsDataView**

Example 22

```
Dim records = From authors In DbauthorDataSet.authors Select authors
    AuthorsBindingSource.DataSource = records.AsDataView

authorsBindingSource.DataSource = records.AsDataView
```

assigns the contents of the records variable to the authorsBindingSource object

Example 23: To use LINQ to select specific records in the authorname field in the authors table implement the following program

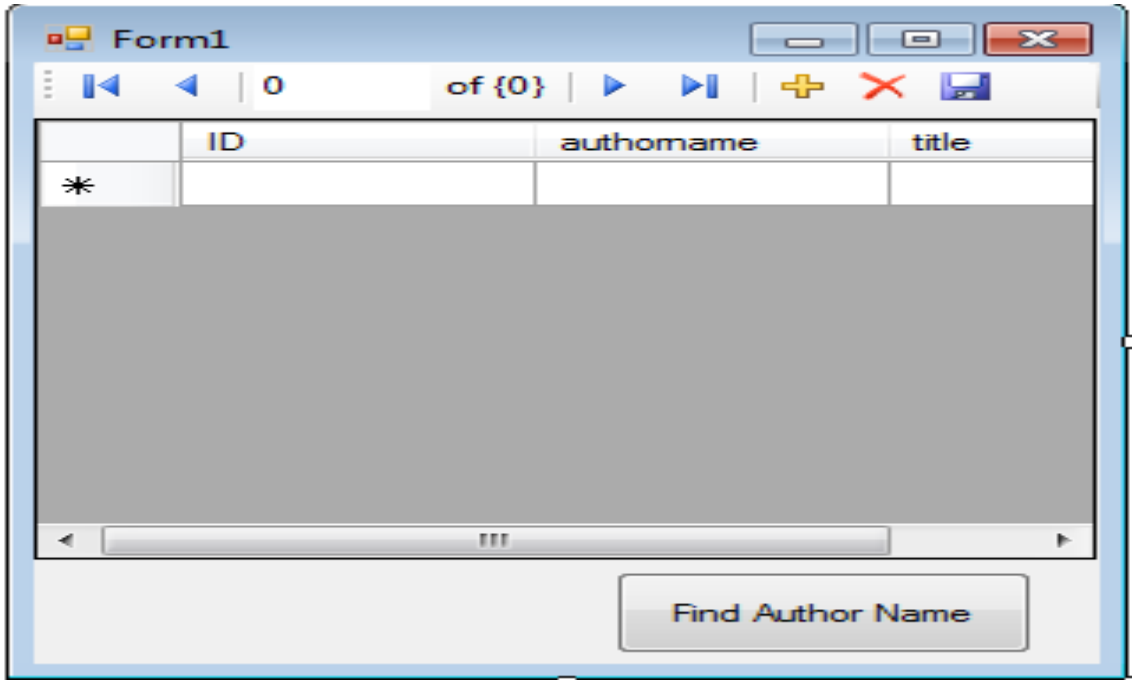


Figure 9.15

Write the following in general code window

```
Option Explicit On
Option Strict On
Option Infer On
```

Then write this code in the button control

```
Const strPROMPT As String = "One or more letters " &
    "(leave empty to retrieve all records):"
    ' get the author name
    Dim strFindName As String =
```

```
InputBox(strPROMPT, "Find Last Name").ToUpper
' select records matching the author name
Dim records = From authors In DbauthorDataSet.authors
Where authors.authorname.ToUpper Like strFindName & "*"
Select authors
AuthorsBindingSource.DataSource = records.AsDataView
```

Run the program, if you enter letter ah for example, the result will be all the records that have the author name begin with ah such as ahmed.

NOTE: Like strFindName & "*", this statement to select any values in field matching the first letters you enterd.

Using the LINQ Aggregate Operators

LINQ provides several aggregate operators that you can use when querying a dataset. Th e most commonly used aggregate operators are Average, Count, Max, Min, and Sum. An **aggregate operator** returns a single value from a group of values.

LINQ aggregate operators

Syntax

Dim *variableName* [**As** *dataType*] **=**
Aggregate *elementName* **In** *dataset.table*
[**Where** *condition*]
Select *elementName.fieldName*
Into *aggregateOperator***()**

Example 24

```
Dim dblAvgRate As Double =
Aggregate books In dbauthorDataSet.books
Select books.price Into Average()
```

calculates the average of the prices rates in the dataset and assigns the result to the dblAvgRate variable

Example 25

```
Dim intCounter As Double =
Aggregate books In dbauthorDataSet.books
Where books.ID = 2
Select books.title Into Count()
```

counts the number of authors books editions whose ID number is 2 and assigns the result to the intCounter variable.

Access Databases and SQL

Syntax and examples of adding a record to a dataset

Adding a record to a dataset

Syntax

*dataSetName.tableName.**Add**tableName***Row(***valueField1*[**,**

valueField2 . . . , valueFieldN]**)**

Example 26

BooksDataSet.tblBooks.AddtblBooksRow(txtTitle.Text,
txtAuthor.Text)
adds a record to the BooksDataSet

Example 27

CDDataSet.tblCds.AddtblCdsRow("02", "Colors", 12.99)
adds a record to the CDDataSet

the TableAdapter object's **Update method**

For the changes made to a dataset to be permanent, you need to save the changes to the database associated with the dataset.

Saving dataset changes to a database

Syntax
*tableAdapterName***.Update(***dataSetName.tableName***)**

Example 28

Try
TblBooksTableAdapter.Update(BooksDataSet.tblBooks)
Catch ex As Exception
MessageBox.Show(ex.Message, "Books",
MessageBoxButtons.OK,
MessageBoxIcon.Information)
End Try
saves (to the tblBooks table in the Books database) the changes made to the BooksDataSet

BindingSource object's Sort method

Sorting the records in a dataset

Syntax
*bindingSourceName***.Sort =** *fi eldName*

Example 29

TblBooksBindingSource.Sort = "Author"
sorts the records by the Author field

Deleting Records from a Dataset

Before deleting any record we must locate the record in the dataset first, thene delete it.

NOTE: Before deleting the record, the procedure should display a message that asks the user to confirm the deletion. You will use the MessageBox.Show method to both display the message and get the user's response.

Syntax and examples of locating a record in a dataset

Locating a record in a dataset

Syntax
dataRowVariable =
dataSetName.*tableName*.**FindBy***fi eldName*(*value*)

Example 30

```
Dim row As DataRow
row = BooksDataSet.tblBooks.FindById(123)
```

The assignment statement searches the dataset for the record whose Id fi eld contains 123, and then assigns the record to the row variable.

Syntax and an example of deleting a record from a dataset

Deleting a record from a dataset

Syntax
dataRowVariable.**Delete()**

Example 31

```
Dim row As DataRow
row = BooksDataSet.tblBooks.FindByTitle("Money")
row.Delete()
```

The Delete method deletes the record associated with the row variable.

Note:built new database bdauthor with one table authors without primary keys to use it in the following example.

Example 32: now draw the following user interface
and write the code for the add record and delete record command buttons.

Note: there are two group box used in this example, one for add and the other for delete.

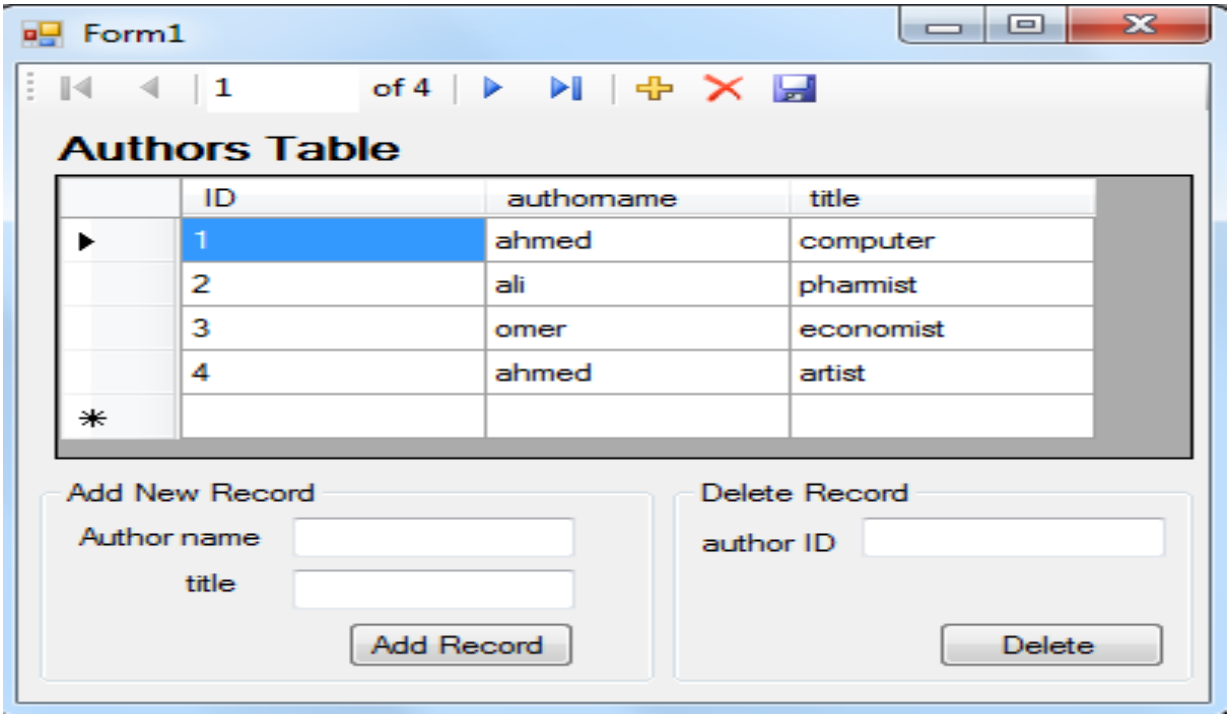


Figure 9.16

Write the following code in Add record button:
DbauthorDataSet.authors.AddauthorsRow(TextBox1.Text,TextBox2.Text)
AuthorsTableAdapter.Update(DbauthorDataSet.authors)

Write the following code in Delete button:

```
Dim id As Integer
    Dim response As Integer
    Integer.TryParse(TextBox3.Text, id)
    response = MsgBox("do you want to delete author name" & TextBox3.Text, MsgBoxStyle.YesNo)

    If response = vbYes Then
       Dim row As DataRow
       row = DbauthorDataSet.authors.FindByID(id)
       row.Delete()
       AuthorsTableAdapter.Update(DbauthorDataSet.authors)
    End If
```

Saving and invoke Query

Now you will build and save query to invoke it from the visual basic code.
To save query:

Right-click **MoviesDataSet.xsd** in the Solution Explorer window and then click **Open** to open the DataSet Designer window.

Right-click **authorsTableAdapter** in the DataSet Designer window. Point to **Add** on the shortcut menu and then click **Query** to start the TableAdapter Query Configuration Wizard. (If Add does not appear on the shortcut menu, click Add Query instead.) Verify that the Use SQL statements radio button is selected. Click the **Next** button to display the Choose a Query Type screen. Verify that the "SELECT which returns rows" radio button is selected. Click the **Next** button to display the Specify a SQL SELECT statement screen. The "What data should the table load?" box contains the default query, which selects all of the fields and records in the table.

Click the **Query Builder** button to open the Query Builder dialog box. You will create a parameter query that allows the user to display the record for specific author.

Write the following SQL statement in the SQL Pane, then click Ok.

```
SELECT      ID, authorname, title
FROM        authors
WHERE       (authorname = ?)
```

Click the **Next** button to display the Choose Methods to Generate screen. If necessary, select the **Fill a DataTable** and **Return a DataTable** check boxes. Change the Fill a DataTable method's name from FillBy to **FillByname**. Change the Return a DataTable method's name from GetDataBy to **GetDataByname.**
Finally click Next , then click Finish.

Now you will see Method names included in the DataSet Designer window

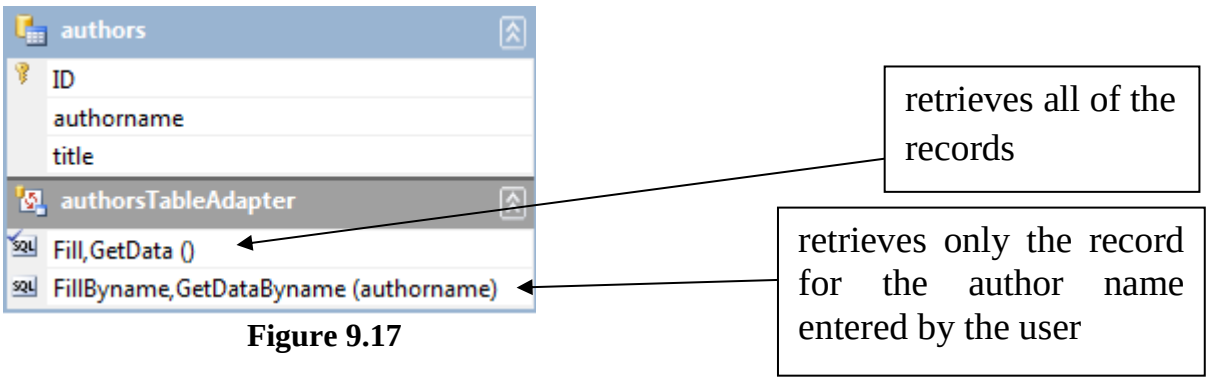


Figure 9.17

Invoking a Query from Code

Now you will invoke the two methods built earlier by the code window.

Example 33: draw the following interface to make user take choice for retrieving either all records(radio button1=ALL) in author table or just record(s) match some author name (radio2=Fill By Name).

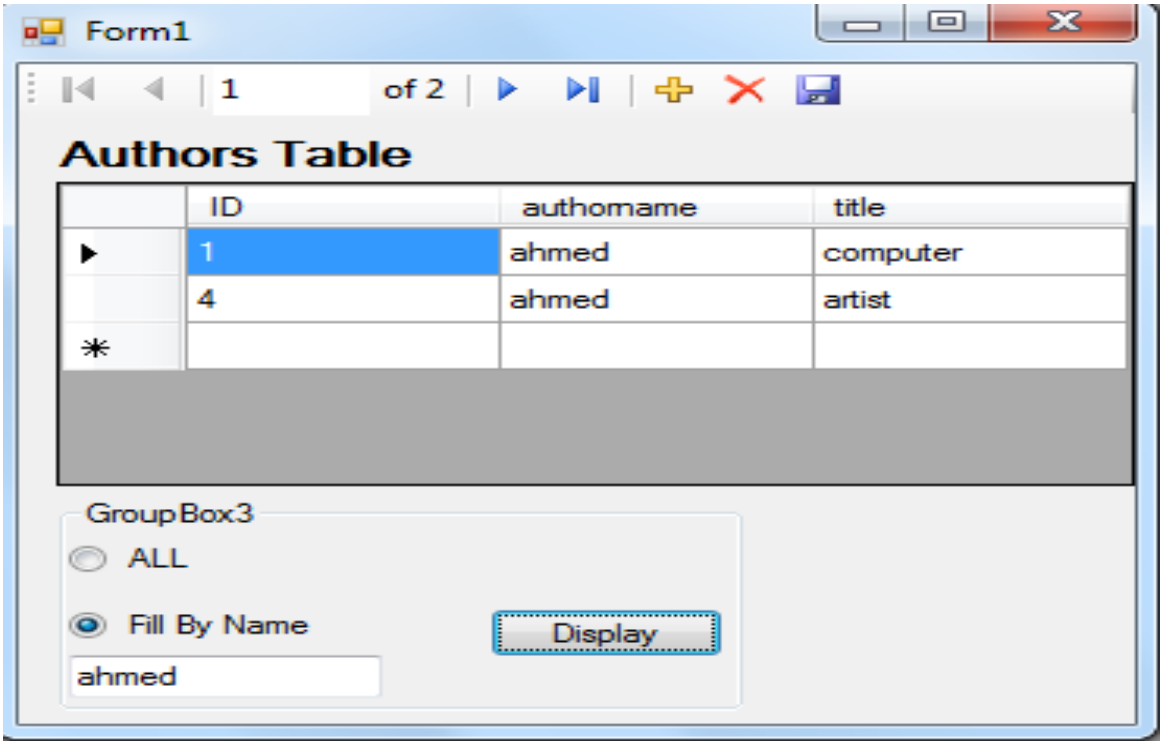


Figure 9.18

Write the following code in Display command button:

```
If RadioButton1.Checked Then
        AuthorsTableAdapter.Fill(DbauthorDataSet.authors)
    Else
        If TextBox1.Text.Trim = String.Empty Then
            MsgBox("please enter the author name")
        Else
            AuthorsTableAdapter.FillByname(DbauthorDataSet.authors,
TextBox1.Text)

        End If
    End If
```

Programming with ADO.NET

it builds on the older ADO (Active Data Objects) technology. The main goal of ADO.NET was to enable developers to easily create distributed, data - sharing applications in the .NET Framework.

ADO.NET provides **DataSet** and **DataTable** objects that are optimized for moving disconnected sets of data across the Internet and intranets, including through firewalls.

At the same time, ADO.NET includes the traditional **connection**, **command,** and **DataAdapter** objects, as well as an object called a **DataReader** .Together, these objects provide the best performance and throughput for retrieving data from a database.

The basic cycle of data driven applications is (retrieve data from DB , present it to the user, allow user to edit data, submit changes to DB).

NOTE: All the ADO.NET objects, except the DataTable, are part of the System.Data namespace. The DataTable is part of System.Xml.

NOTE: for any VB.Net project to add references to any namespaces so that you can use the namespaces without having to type the full namespace qualifier:right click Project name, choose Properties to display the Project Properties. Click the References tab to display the active references for the project, then from the upper list choose any namespace you want to add to your project then click Add button.

Classes of ADO.NET

To summarize, ADO.NET provides three core classes for accessing databases: the **Connection**, **Command**, and **DataReader** classes.
Connection class has three derived calasses (SqlConnection, OracleConnection, and OleDbConnection.)
the Command class has three derived classes: SqlCommand, OracleCommand, and OleDbCommand.
The OleDbConnection and OleDbCommand classes belong to the OleDb namespace, which
you must import into your project via the following statement:

Imports System.Data.OleDb

Datareader reading the results of the query returned from database to your application in a forward-only manner(read rows returned by the query, then close connection), read-only (you can't use it to update the underlying rows)

Connecting to a Database

To access data in a database, you must first establish a connection, using an ADO.NET connection object.

1- To make connection with OLEDB (Microsoft Access database), the OleDbConnection object is used

2 - for optimized access to Microsoft SQL Server, the SqlConnection object is used

Before using this connection, you must specify the data source(database name) to which you want to connect. This is done through the **ConnectionString** property of the ADO.NET connection object.below is the parameters of **ConnectionString** property:

Table 9.4

Parameter	Description
Provider	The name of the data provider (Jet, SQL,and so on) to use.
Data Source	The name of the data source to connect to.
User ID	A valid username to use when connecting to the data source.
Password	A password to use when connecting to the data source.

NOTE: username and password of an account configured by the database administrator for more secured database.

Connection class

*method of initializing a Connection object in your code is the following (**assuming you have the SQL server database and VB application in the same computer machine.)**:

```
Dim CN As New SqlConnection("Data Source = localhost;" &
"Initial Catalog = Northwind; uid = user_name;" &
"password = user_password")
```

localhost is a universal name for the local machine, *Northwind* is the name of the database, and *user_name* and *user_password* are the username and password of an account

*If SQL Server resides on a different computer in the network, use the server computer's name in place of the localhost name

* If the database is running on a remote machine, use the remote machine's IP address. If you're working from home, for example, you can establish a connection to your company's server with a connection string like the following:

```
Data Source = "213.16.178.100; Initial Catalog = BooksDB; uid = xxx; password
= xxx"
```

NOTE: If you're using an IP address to specify the database server, you may also have to include SQL Server's port by specifying an address such as 213.16.178.100, 1433. The default port for SQL Server is 1433, and you can omit it.

If the administrator has changed the default port or has hidden the server's IP address behind another IP address for security purposes, you should contact the administrator to get the server's address.

NOTE: If you're connecting over a local network, you shouldn't have to use an IP address.

If you want to connect to the company server remotely, you will probably have to request the server's IP address and the proper credentials from the server's administrator.

NOTE: If you want to connect to the database by using each user's Windows credentials, you should omit the uid and password keys and instead use the Integrated Security key.

Example 34: Dim CN As New SqlConnection
CN.ConnectionString =
"Data Source = localhost; Initial Catalog = Northwind; " &
"Integrated Security = True"

Example 35: to open connection to Microsoft access database 2007(OLEDB.12.0)

Imports the following namespace, writing in general declaration window

```
Imports System.Data.OleDb
```

In the class section of the form write:

```
Dim myConnection As OleDbConnection = New OleDbConnection
```

Then, Write the following code in the form-load section to setting the connectionstring property

```
myConnection.ConnectionString = "Provider= Microsoft.ACE.OLEDB.12.0;" & _
      "DataSource =C:\Users\Computer\Documents\dbauthor.accdb"

myConnection.Open()
```

Important NOTE: if you using office 2003 , then using (OLEDB.11.0) instead of 12.

The Command Class

the Command class, which allows you to execute SQL statements against the database.

The constructor for an OleDbCommand optionally takes the command to execute SQL statementst, as well as a connection specifying the datasource

Dim OleDbCommandAs New OleDbCommand([CommandText],[Connection])

Methods of Command Class

The ExecuteNonQuery: executes INSERT/DELETE/UPDATE statements that do not return any rows, just an integer value, which is the number of rows affected by the query.

The ExecuteScalar: returns a single value, which is usually the result of an aggregate operation, such as the count of rows meeting some criteria, the sum or average of a column over a number of rows, and so one or more tables.

ExecuteReader: is used with SELECT statements that return rows from one or more tables.

The DataReader class

provides the Read method, which advances the current pointer to the next row in the result set. To read the individual columns of the current row, you use the Item property.

Example 36: the following example is to read the record of specific author, and know the number of the records in the authors recordset(table).

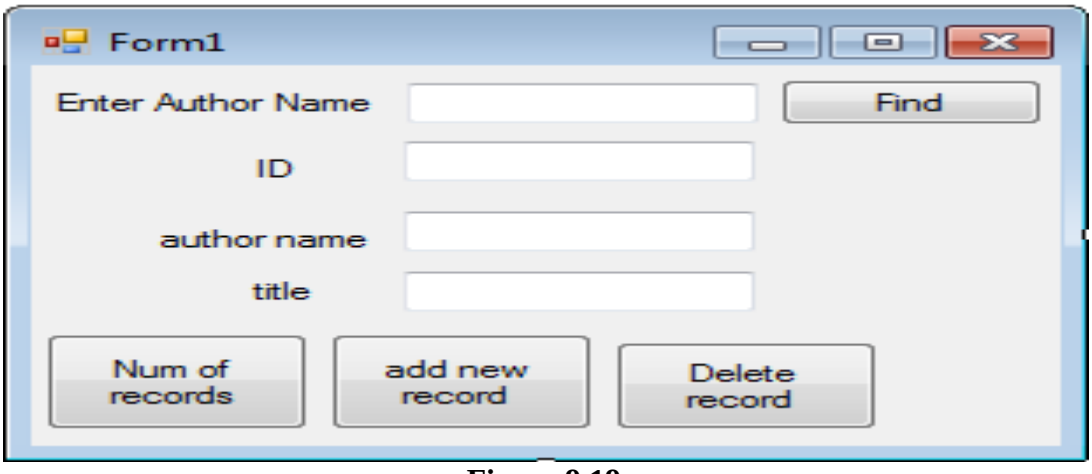


Figure 9.19

In general code window write the following
Imports System.Data.OleDb
in class form section write
Public myConnection As OleDbConnection = New OleDbConnection
Dim provider As String

```
    Dim dataFile As String

    Dim connString As String
    Public dr As OleDbDataReader
```

In form load section write

```
provider = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source ="

        dataFile = "C:\Users\Computer\Documents\dbauthors.accdb"
        connString = provider & dataFile

        myConnection.ConnectionString = connString
```

in Find button write

```
myConnection.Open()

        TextBox2.Clear()
        TextBox3.Clear()
        TextBox4.Clear()

        Dim str As String

        str = "SELECT * FROM authors WHERE (authorname = '" & TextBox1.Text &
"')"

        Dim cm As OleDbCommand = New OleDbCommand(str, myConnection)

        dr = cm.ExecuteReader

        While dr.Read()

            TextBox2.Text = dr("ID").ToString

            TextBox3.Text = dr("authorname").ToString

            TextBox4.Text = dr("title").ToString

        End While

        myConnection.Close()
```

in num of records button write

```
Dim CMD As New OleDbCommand("SELECT COUNT(*) FROM authors",
myConnection)
        Dim count As Integer
        myConnection.Open()
        count = CMD.ExecuteScalar
```

```
    MsgBox(count)
    myConnection.Close()
```

in add new record button write the following code

```
Dim str As String
    Dim ins As Integer
    str = "insert into authors ([ID], [authorname], [title]) values (?, ?, ?)"

    Dim cmd As OleDbCommand = New OleDbCommand(str, myConnection)

    cmd.Parameters.Add(New OleDbParameter("ID", CType(TextBox2.Text,
String)))

    cmd.Parameters.Add(NewOleDbParameter("authorname",
CType(TextBox3.Text, String)))

    cmd.Parameters.Add(New OleDbParameter("title", CType(TextBox4.Text,
String)))

    myConnection.Open()

    ins = cmd.ExecuteNonQuery()
    If ins = 1 Then
       MsgBox("added")
    End If
    myConnection.Close()
```

in delet record button write the following

```
Dim str As String
    Dim n As Integer

    str = "Delete from authors Where authorname = '" & TextBox1.Text & "'"

    Dim c As OleDbCommand = New OleDbCommand(str, myConnection)

    myConnection.Open()

    ' c.ExecuteNonQuery()
    n = c.ExecuteNonQuery
    If n <> 0 Then
       MsgBox("deleted")
    End If

    myConnection.Close()
```

Procedure

1- Write VB.NET program to connect VB project to office database (dbauthor), and display the three fields of the authors table on the form below, Then execute the same example with datagridview control.

2- Repeat previous the previous question with four buttons set their text property to(First,Next,Last,Previous), and execute the four methods(movefirst, movenext,….) of the bindingsource**.**

Discussion

Write vb.net program to implement LINQ aggregation functions on Books table of the dbauthor database.

Experiment No. (1o)

Tool Box Control

Object

Using Web Browser control to allow the user to surf the internet and open web enabled documents like html help files.

WebBrowser Control

The VB.NET Windows.Forms.WebBrowser control can be used within applications to allow the user to surf the internet and open web enabled documents like html help files, word documents etc.

WebBrowser Properties

URL property: Gets or sets the URL of the current Web page. Setting this property navigates the control to the new URL.

AllowNavigation: Gets or sets a value indicating whether the control can navigate to another page after its initial page has been loaded.

The control includes several navigation methods or functions as: Url ,Navigate ,GoBack ,GoForward ,GoHome, GoSearch.

Example 1:Now add a label, TextBox and a button.

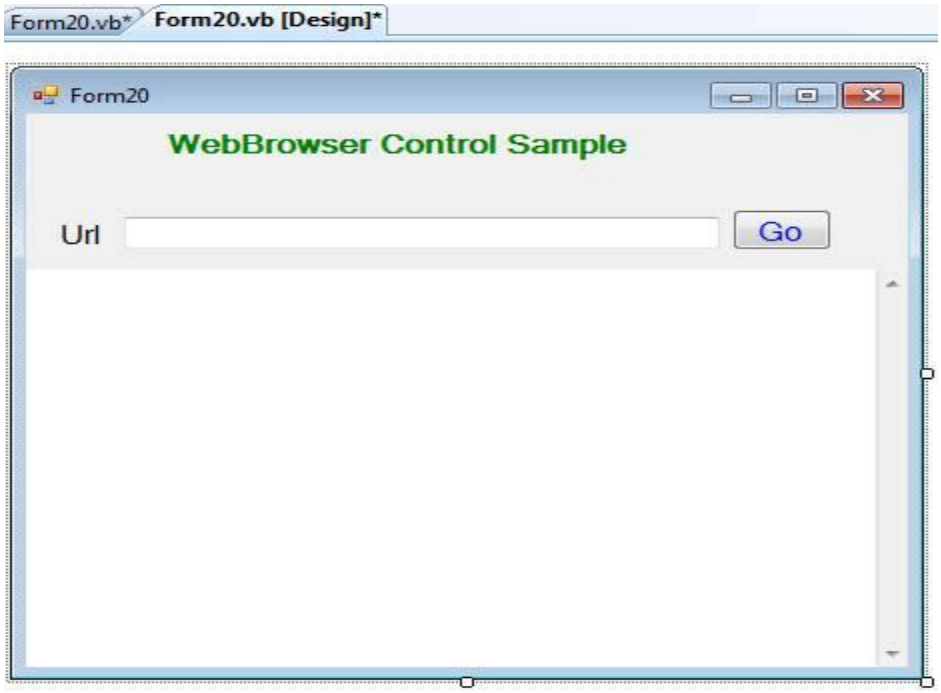


Figure 10.1

Code: Write below code in button click GO:

WebBrowser1.Navigate(TextBox1.Text) ''The **Navigate method** of browser control views a page in the viewer.

Then run the program When you enter URL address in textbox and click Go button then click event will fire and navigate method will open website which URL address is given in TextBox.

Bachgroundworker

If we have an application that has some code that is very time consuming such as large text file parsing and takes several minutes or longer, you may not want your users to wait till the process is finished. We can execute this process in a separate thread also known as a worker or secondary thread.
A BackgroundWorker component executes code in a separate dedicated secondary thread. The BackgroundWorker type in the VB.NET language is useful for creating a background process, one which won't block the

user interface if it takes a long time to run. In most programs, disk or network accesses or database accesses and loading huge images are the slowest and should be put on a background thread if possible.

Methods

BackgroundWorker has two methods, RunWorkerAsync and CancelAsync. The RunWorkerAsync starts the thread and the CancelAsync stops the thread.

Events

DoWork This event is fired when the RunWorkerAsync method is called. In this event hander, we call our code that is being processed in the background where our application is still doing some other work. The DoWork event handler looks like this where we need to call our time-consuming method.

RunWorkerCompleted event occurs when the background operation has completed. The BackgroundWorker component is defined in the System.ComponentModel namespace. To use the BackgroundWorker, you can add an Imports statement or use the namespace in the declaration and initialization statement. Here is an example:

```
Dim worker as new System.ComponentModel.BackgroundWorker
```

Or write in the general declaration of the code window the namespace system.componentmodel

```
Imports system.componentmodel
```

And then just write

```
Dim  anyvariable as backgroundworker
```

Example 2: While BackgroundWorker will be doing some background work, user can still type in the RichTextBox.

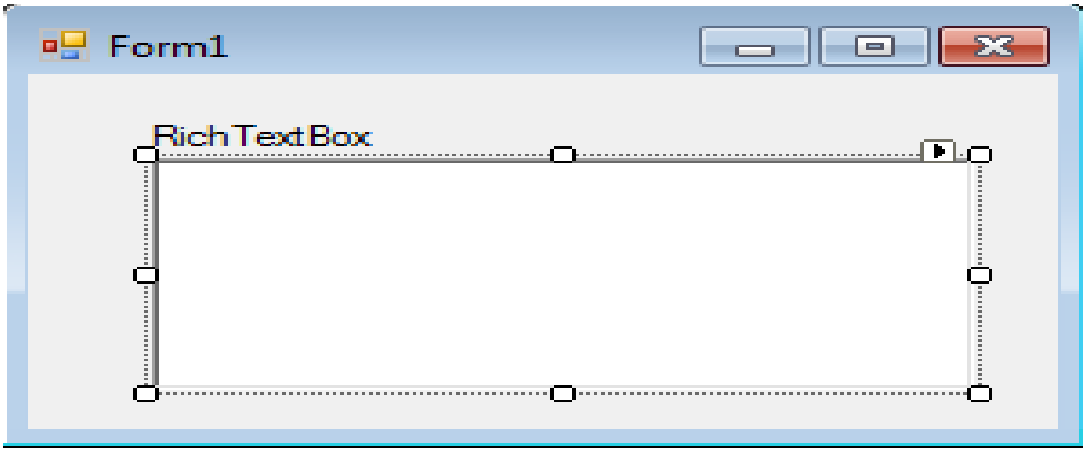


Figure 10.2

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

' Start up the BackgroundWorker1.

    BackgroundWorker1.RunWorkerAsync()

End Sub

Private Sub BackgroundWorker1_DoWork(ByVal sender As System.Object, ByVal e
As System.ComponentModel.DoWorkEventArgs) Handles
BackgroundWorker1.DoWork

' Do some time-consuming work on this thread.

System.Threading.Thread.Sleep(20000)

    MessageBox.Show("I was sleeping for 20 seconds")
End Sub

Private Sub BackgroundWorker1_RunWorkerCompleted(ByVal sender As Object,
ByVal e As System.ComponentModel.RunWorkerCompletedEventArgs) Handles
BackgroundWorker1.RunWorkerCompleted

' Called when the BackgroundWorker is completed.

    MsgBox("the work is complete")
End Sub
```

Errorprovider

An error provider is a better alternative than displaying an error message in a message box, because once a message box is dismissed, the error message is no longer visible. The**ErrorProvider** component displays an error icon ( ) next to the relevant control, such as a text box; when the user positions the mouse pointer over the error icon, a ToolTip appears, showing the error message string

Example 3: input just numbers to the textbox, else the error provider icon will appear

```
Private Sub TextBox1_TextChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles TextBox1.TextChanged
        If Not IsNumeric(TextBox1.Text) Then
            ErrorProvider1.SetError(TextBox1, "Not a numeric value.")
        Else
            ' Clear the error.
            ErrorProvider1.SetError(TextBox1, "")
        End If
    End Sub
```

Filesystemwatcher

Another very useful class, FileSystemWatcher, acts as a watchdog for file system changes and raises an event when a change occurs. You must specify a directory to be monitored. The class can monitor changes to subdirectories and files within the specified directory. If you have Windows 2000, you can even monitor a remote system for changes. (Only remote machines running Windows NT or Windows 2000 are supported at present.)

The option to monitor files with specific extensions can be set using the Filter property of the FileSystemWatcher class. You can also fine-tune FileSystemWatcher to monitor any change in file Attributes, LastAccess, LastWrite, Security, and Size data.

The FileSystemWatcher class raises the events described below.

Table 10.1 FileSystemWatcher events

Event Name	Use
Changed	Fired when a file or directory in the watched path is changed
Created	Fired when a file or directory in the watched path is created
Deleted	Fired when a file or directory in the watched path is deleted
Error	Fired when the internal buffer overflows due to many changes made over a short time, particularly when the buffer size is small

Renamed	Fired when a file or directory in the watched path is renamed

Process

An external process can be launched by a VB.NET application. This starts a program for the user. It runs a command-line program. We see several examples of the **Process.Start** Function. This function is ideal for launching programs.

Note 1: Process.Start launches external programs, such as Word, Excel, a web browser—EXE programs of any kind.

Note 2: It is found in System.Diagnostics.

Example 4: First, here is an example VB.NET program that uses Process.Start to open the file manager on your C:\ drive. When you run this example the root directory folder will open.
now create module name is module1, and write this code:

```
Module Module1
    Sub Main()
        Process.Start("C:\")
    End Sub

End Module
```

Then call the subroutine main in the button click procedure:

```
Module1.Main()
```

Example 5: Program that launches web browser [VB.NET], You can tell Windows to launch a web browser window with a specific URL
Write in the module

```
Module Module1
    Sub openweb()
        SearchGoogle("VB.NET awesome")
    End Sub

    ''' <summary>
    ''' Open the user's default browser and search for the parameter.
    ''' </summary>
    Private Sub SearchGoogle(ByVal t As String)
        Process.Start("http://google.com/search?q=" & t)
    End Sub
End Module
```

Write in the click button:

```
Module1.openweb()
```

The result is the following window:

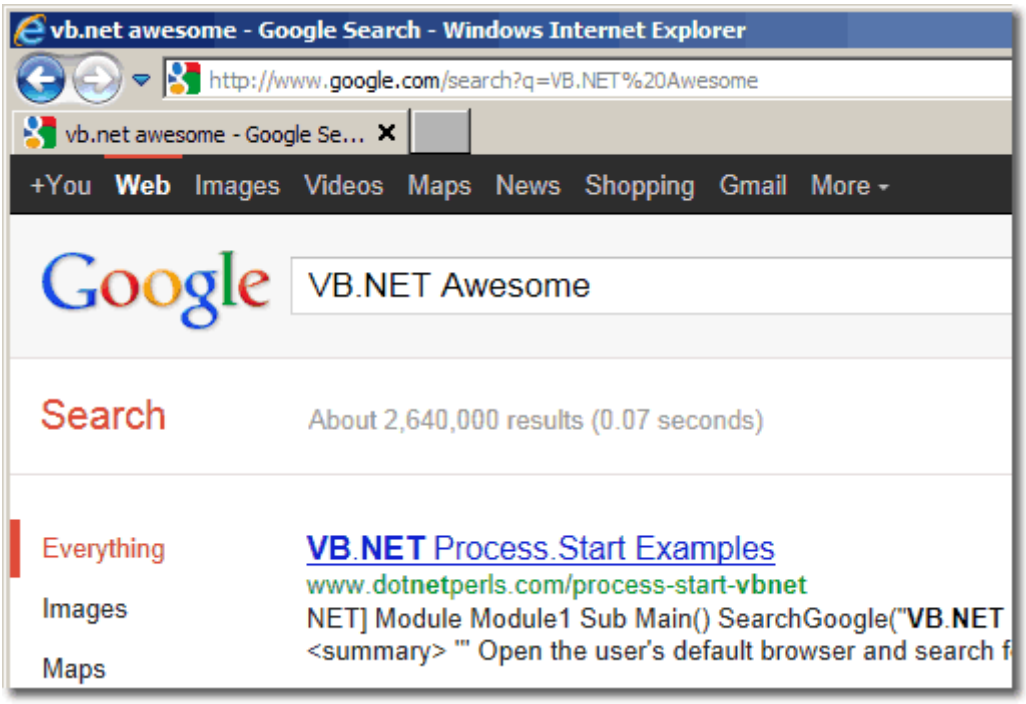


Figure 10.3

Serial port

A port, or *interface*, that can be used for serial communication, in which only 1 bit is transmitted at a time asynchronously in or out the computer.

A serial port is a general-purpose interface that can be used for almost any type of device, including modems, mice, and printers, bar code scanners,display prices, And used in networking.

Operating systems usually use a symbolic
name to refer to the serial ports of a computer.
For example, the Microsoft MS-DOS and Windows environments refer to serial ports as COM ports: COM1, COM2...etc.

Advantages of serial communications

is very cheaper than parallel port. And most of the times there are more chances of corruption of data in parallel port as compared to the serial port. A serial port also have the advantage that it requires very little supporting software from the host system.

Example 6: you'll writing a simple Windows Application in VB.NET that enables simple 2-way communication or data transfer via the serial communications port (COM1). you've just succesfully written a Serial Communications program that utilizes SerialPort class in VB.NET
Drag serialport component on the form
For now the PortName should be COM1. WriteTimeOut and ReadTimeOut at around 500ms.
Drage button name it (open port)

This code essentially open and closes the serial com port by clicking the button. It will also change states as in the text on the button will match the current action that will be allowed. It will also enable or disable the Send Button(added in the next step) to avoid an illegal operation.

```
If Button1.Text Is "Open Port" Then
 SerialPort1.Open()
 Button1.Text = "Close Port"
 Button2.Enabled = True

ElseIf Button1.Text Is "Close Port" Then
 SerialPort1.Close()
 Button1.Text = "Open Port"
 Button2.Enabled = False
End If
```

This code snippet functions as the data sending portion of the application and it'll log to the ListBox that we will add later.

```
 SerialPort1.WriteLine(TextBox1.Text)
ListBox1.Items.Add("Sent: " + TextBox1.Text)
```

12. Next, go back to the design view and we will drag and drop a ListBox into the Form. Resize it as necessary to fit the Form.

13. Drag and drop a TextBox into the Form.

Lastly left click once on the SerialPort at the bottom. Then, go to the Properties pane, click on the lightning symbol. You'll see Misc, DataReceived, ErrorReceived, PinChanged. Double click on DataReceived and fill the sub with this code.

```
ListBox1.Items.Add("Received: " + SerialPort1.ReadLine())
```

This function will be called whenever there are data stored in the input buffer. It is to display incoming data from the Serial Communication Port.

15. You've completed the coding section. Now press Ctrl-F5 to Start Without Debugging. The application should run. Test the application by clicking Open Port, keying in some data in the TextBox and then click Send. Make sure the Rs232 cable is connected between 2 computers

Service controller

A ServiceController component allows us to access and manage Windows Services running on a machine. In this article, I will discuss how to a ServiceController class to load all services and get information about them in a Windows Forms application using Visual Studio 2010.

Adding System.ServiceProcess Reference

A ServiceController represents a Windows Service and defined in the System.ServiceProcess namespace. Before this namespace can be imported, you must add reference to the System.ServiceProcess assembly.

To add reference to an assembly, right click on the project name in Visual Studio and select Add Reference menu item and then browse the assembly you need to add to your application.

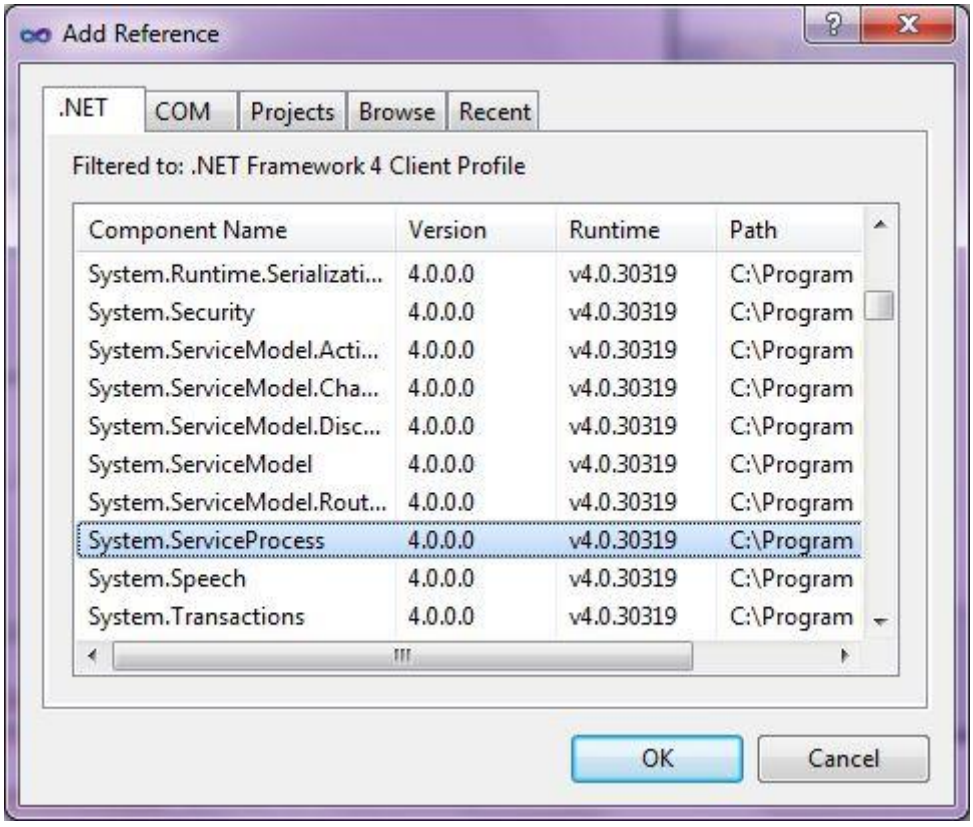


Figure 10.4

Getting all services installed on a machine

ServiceController.GetServices static method returns list of all services running on a machine. The following code snippet gets all the services and displays them in a ListBox.

ServiceController Properties

ServiceName - Service name.
DisplayName - Friendly name of a service.
ServiceType - Type of a service such as device driver, process, or kernel.
Status - Status of a service such as running, stopped, or paused.

Example 7: The following code snippet uses some of these properties. Built the subroutine then call it by the command

```
Private Sub GetAllServices()
    For Each service As ServiceController In ServiceController.GetServices()
      Dim serviceName As String = service.ServiceName
      Dim serviceDisplayName As String = service.DisplayName
      Dim serviceType As String = service.ServiceType.ToString()
      Dim status As String = service.Status.ToString()
      ListBox1.Items.Add(serviceName + "  " + serviceDisplayName +
        serviceType + " " + status)
    Next
  End Sub
```

Write in the command button (services):

Call GetAllServices()
After running the program you will get the following:

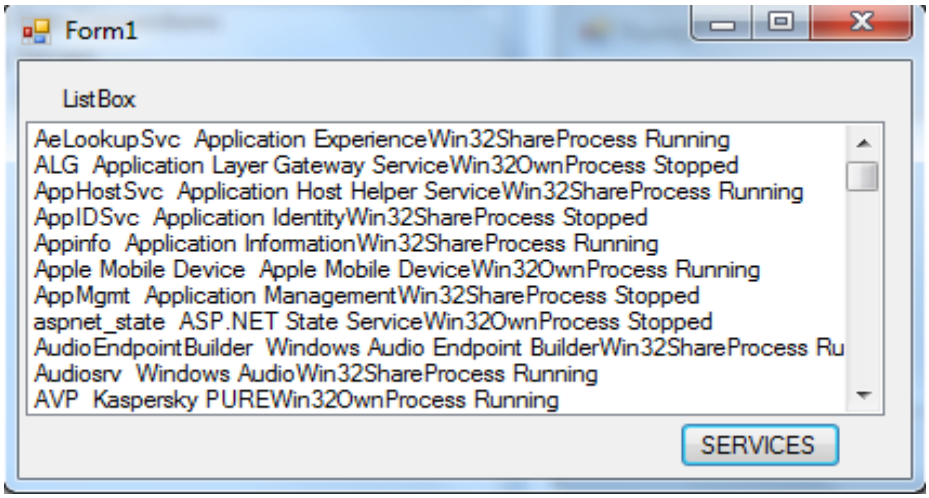


Figure 10.5

Timer

If you need to execute some code after certain interval of time continuously, you can use a timer control., you can specify a subroutine to perform the maintenance or updating code.

NOTE: A Timer control does not have a visual representation and works as a component in the background.

Timer Properties

Enabled property of timer represents if the timer is running. We can set this property to true to start a timer and false to stop a timer.

Interval property represents time on in milliseconds, before the Tick event is raised relative to the last occurrence of the Tick event. One second equals to 1000 milliseconds. So if you want a timer event to be fired every 5 seconds, you need to set Interval property to 5000.

```
Dim Timer1 As New Timer()
Timer1.Interval = 2000
Timer1.Enabled = True
```

Events

If you go to the Events window by clicking little lightning icon, you will see only one Tick event. Double click on it will add the Tick event handler.

Example 8: if you have a ListBox control on a Form and you want to add some items to it, like the current dates and times added every some interval

NOTE: set the interval property to 3000 and the enabled property enabled= true.

```
Private Sub Timer1_Tick(ByVal sender As System.Object, _
                        ByVal e As System.EventArgs) _
                    Handles Timer1.Tick
  ListBox1.Items.Add(DateTime.Now.ToLongTimeString()
+ "," + DateTime.Now.ToLongDateString())
End Sub
```

List View

The ListView control is used to display a list of items. Along with the TreeView control, it allows you to create a Windows Explorer like interface.

The *ListView* control displays a list of items along with icons. The Item property of the ListView control allows you to add and remove items from it. The *SelectedItem* property contains a collection of the selected items. The *MultiSelect* property allows you to set select more than one item in the list view. The *CheckBoxes* property allows you to set check boxes next to the items.

Rich TextBox control

A RichTextBox control is an advanced text box that provides text editing and advanced formatting features including loading rich text format (RTF) files.

Text and TextLength

Text property of a RichTextBox represents the current text of a RichTextBox control. The TextLength property returns the length of a RichTextBox contents.

Maximum Length

You can restrict number of characters in a RichTextBox control by setting MaxLength property. The following code snippet sets the maximum length of a RichTextBox to 50 characters.

```
RichTextBox.MaxLength = 50
```

ReadOnly

You can make a RichTextBox control read-only (non-editable) by setting the ReadOnly property to true. The following code snippet sets the ReadOnly property to true.

```
RichTextBox.ReadOnly = True
```

Enabling and Disabling Shortcuts

ShortcutsEnabled property of the RichTextBox is used to enable or disable shortcuts. By default, shortcuts are enabled.

ShortcutsEnabled property applies to the following shortcut key combinations:

- CTRL+Z
- CTRL+E
- CTRL+C
- CTRL+Y
- CTRL+X
- CTRL+BACKSPACE
- CTRL+V
- CTRL+DELETE
- CTRL+A
- SHIFT+DELETE
- CTRL+L
- SHIFT+INSERT
- CTRL+R

SelectAll and DeselectAll Methods

RichTextBox class provides SelectAll and DeselectAll methods to select and deselect all text of a RichTextBox control.

Picture Box Control

PictureBox control is used to display images in Windows Forms. In this article, I will discuss how to use a PictureBox control to display images in Windows Forms applications.

Display an Image

Image property is used to set an image to be displayed in a PictureBox control. The following code snippet creates a Bitmap from an image and sets the Image property of PictureBox control.

SizeMode

SizeMode property is used to position an image within a PictureBox. It can be Normal, StretchImage, AutoSize, CenterImage, and Zoom.

Example 10: uploade picture from any location in the computer by using open dialogue box control and image property of the picture box.

Upload a picture to PictureBox

First Add a OpenFileDialog to the project

Add the following code to BROWSE button

```
Dim OpenFileDialog1 As New OpenFileDialog

With OpenFileDialog1
.CheckFileExists = True
.ShowReadOnly = False
.Filter = "All Files|*.*|Bitmap Files (*)|*.bmp;*.gif;*.jpg"
.FilterIndex = 2
If .ShowDialog = DialogResult.OK Then

PictureBox1.Image = Image.FromFile(.FileName)

End If
End With
```

Example 11:For this tutorial you need a picturebox, a textbox, and a button.

Add atermark Button: water is embedded graphix or image into another image used for more protection like currency and formal letters.

```
Button1_Click

Dim NF As New Font("Impact", 36, FontStyle.Italic)
```

```
Dim NB As New SolidBrush(Color.FromArgb(64, 192, 255, 255))

NB.Color = Color.FromArgb(128, 0, 0, 0)

PictureBox1.CreateGraphics.DrawString(TextBox1.Text, NF, NB, 15, 135)
```

Figure 10.7

LinkLabel control

A LinkLabel control is a label control that can display a hyperlink. You can make link to web page

```
System.Diagnostics.Process.Start("http://www.google.com")
```

to open some application like start the Solitaire game,

```
System.Diagnostics.Process.Start("sol.exe")
```

or open some picture in some location on the computer

```
System.Diagnostics.Process.Start("C:MyPicture.jpg")
```

You can use the LinkLabel almost like a button by simply putting any code you like in the LinkClicked event instead of the Start method.

Procedure

1- Add multiple columns to listview control, then add content to one row of the control
2- Repeate the previous example but by using textboxes to add items to listview control.

Discussion

Implement web application to browsing webpages. Use methods of webbrowser control

REFERENCES

1- Clearly Visual Basic, DIANE ZAK, Second Edition, 2010

2- PROGRAMMING WITH MICROSOFT VISUAL BASIC 2010, DIANE ZAK, Fifth Edition, 2010

3- MASTERING, Copyright © 2010 by Wiley Publishing, Inc., Indianapolis, Indiana Published simultaneously in Canada

4- Sams teach yourself Visual Basic 2010 in 24 hours complete : starter kit / James Foxall. Library of Congress Cataloging-in-Publication Data: Foxall, James D.

5- Processing Text Files for Input, Processing Text Files for Input.html

6- Visual Basic 2008 / 2010 Tutorials,

support@reflectionforbrain.com, 2012