

## Chapter Three: Creating an Application

### 3.1 Writing Your First Program

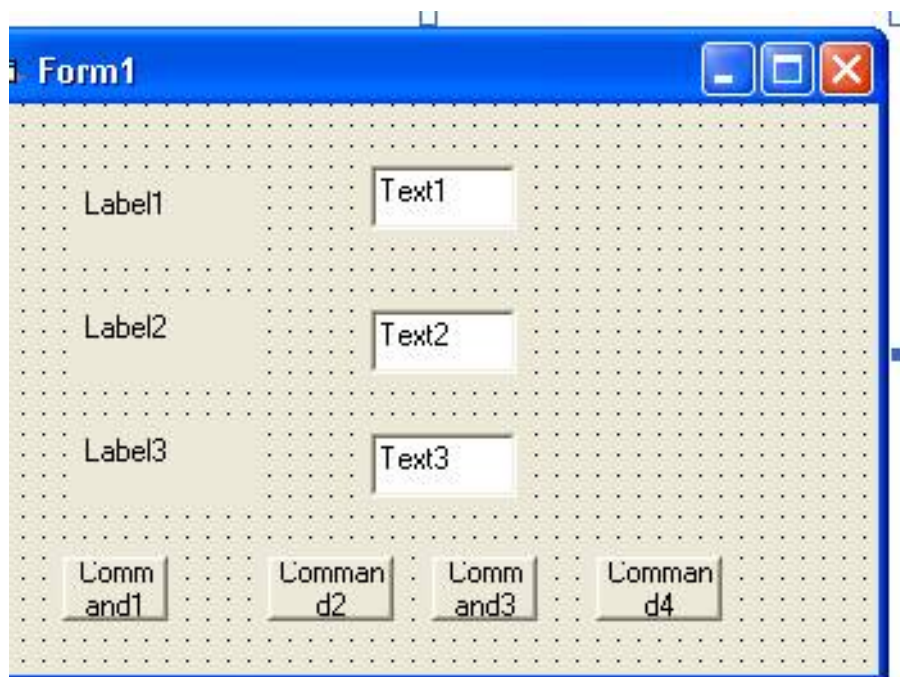
There are four main steps in creating a Windows application. These are:

- Creating the interface for a new program.
- Setting the properties of objects in the interface.
- Writing the code to control events
- Save and run the program.

We will build a little calculator program (first program)

#### 1-Creating the Interface

The interface for this example application needs 4 TextBox to the form, and 4 buttons. (This will be a very basic calculator that can add, subtract, multiply and divide only) and 3 labels see figure 1.

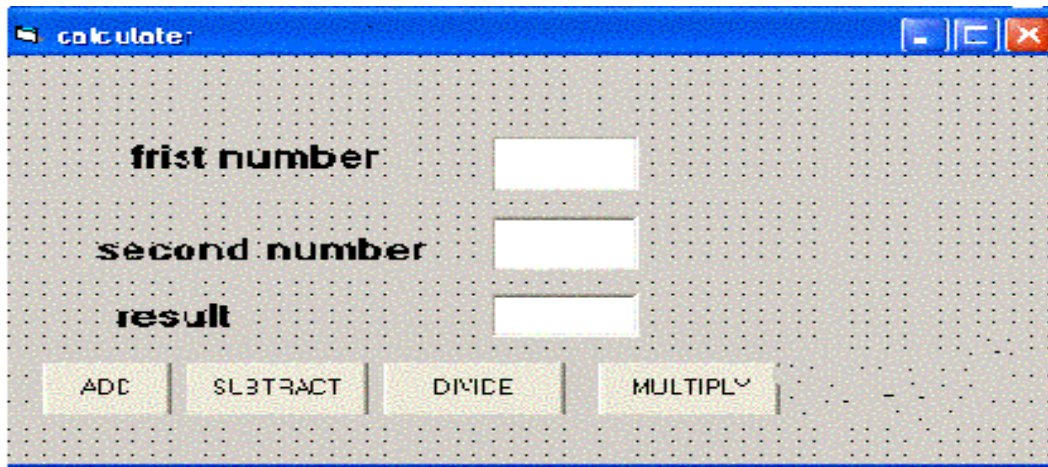


## 2-Setting Properties

Set the following properties for the controls used in the application

Object	Property	Setting
lable1	caption	first number
	BackStyle	0-transpernt
	Font	Bold –14 size
Lable2	caption	second number
	BackStyle	0-transpernt
	Font	Bold –14 size
Lable3	caption	result
	BackStyle	0-transpernt
	Font	Bold –14 size
Text1	name	txtinput1
	Text	blank
Text2	name	txtinput2
	Text	blank
Text3	name	txtresult
	Text	blank
Cammand1	name	cmdAdd
	Caption	add
Cammand2	name	cmdSub
	Caption	Subtract
Cammand3	name	cmdDivide
	Caption	Divide
Cammand4	name	cmdMultiplay
	Caption	Multiplay

The screen should then look like this



### 3- Writing Code

In addition to the properties associated with the objects in an application, there will usually be some written code to determine what should happen when an event occurs. For example, when the user clicks on a command button, the code associated with the button will be executed. See the code window for commands buttons.

```
Private Sub cmdAdd_click ()  
    TxtResult.text= Val(txtinput1.text) + Val(txtinput2.text)  
End Sub
```

```
-----  
Private Sub cmdSub_click ()  
    TxtResult.text= Val(txtinput1.text) - Val(txtinput2.text)  
End Sub
```

```
-----  
Private Sub cmdDivide_click ()  
    TxtResult.text= Val(txtinput1.text) / Val(txtinput2.text)  
End Sub
```

---

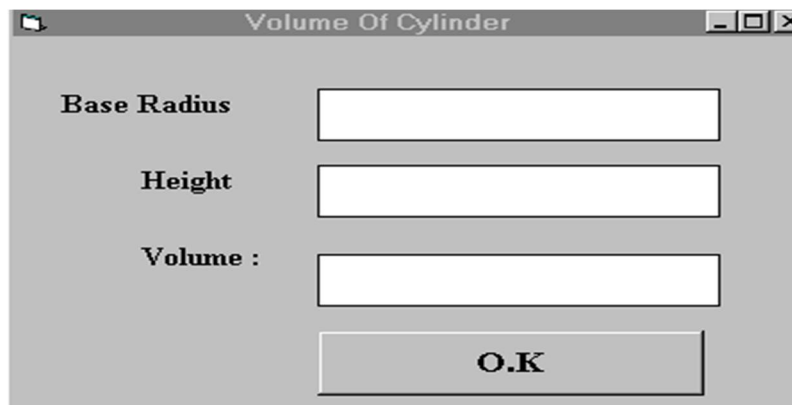
```
Private Sub cmdMultiplay_ click ()  
    TxtResult.text= Val(txtinput1.text) * Val(txtinput2.text)  
End Sub
```

#### 4- Saving the Application

Before testing the application, it is a good idea to save your work. To do this, choose Save Project from the File menu or click on the Save Project button in the toolbar. You will be prompted to save the form and then the project as a whole under their default names (Form1 and Project1) or names of your choosing.

#### 3.2 The second program

This program is a simple program that calculate the volume of a cylinder. Let design the interface:



First of all, go to the properties window and change the **form caption** to **Volume Of Cylinder**. Then draw three **label boxes** and change their **captions** to **Base Radius**, **height** and **volume** respectively. After that, draw three **Text Boxes** and clear its **text** contents so that you get three **empty boxes**. **Named** the **text boxes** as **radius**, **height1** and **volume** respectively. Lastly, insert a **command button** and change its **caption** to **O.K.** and its **name** to **OK**. Now, doubleclick on the O.K button and enter the codes between **Private Sub OK\_Click()** and **End Sub**

**Private Sub OK\_Click( )**

r = Val(radius .Text)

h = Val(height1.Text)

pi = 22 / 7

v = pi \* (r ^ 2) \* h

volume.Text= v

**End Sub**

When you run the program, you should be able to see the interface as shown above. If you enter a value each in the radius box and the height box, then click OK, the value of the Volume will be displayed in the volume box. Let me describe the steps using pseudocodes as follows:

***Procedure for clicking the OK button to calculate the volume of cylinder***

***get the value of r from the radius text box***

***get the value of h from the height 1 text box***

***assign a constant value 22/7 to pi***

***calculate the volume using formula***

***output the results to the Volume text box***

***End of Procedure***

The syntax ***radius.Text*** consists of two parts, radius is the **name** of text box while **Text** is the textual contents of the text box. Generally, the syntax is:

**Object.Property**

In our example, the objects are ***radius***, ***height1*** and ***volume***, each having ***text*** as their ***property***. Object and property is separated by a period(or dot).The contents of a text box can only be displayed in textual form, or in programming term,as

string. To convert the contents of a text box to a numeric value so that mathematical operations can be performed , you have to use the function *Val*. Finally, In order to display the results in a text box, we have to perform the reverse procedure, that is, to convert the numeric value back to the textual form, using the function *Str\$*.

**OK\_Click** defines what kind of action the sub procedure OK will response .Here, the action is mouse click. There are other kind of actions like keypress, keyup, keydown and etc.

**A program:** - is a set of instructions that make the computer do something such as perform accounting.

**A project:** - is a collection of files you create that comprises your Windows application.

**An application:-**is a collection of one or more files that compile into an executable program.

**A form module:** - is a module file that holds one or more forms and the code that goes with each form.

**A standard module:** - is a file that holds code not related to a form.

**Note:** - A form module is code connected to a specific form. Every application has at least one form, so every application contains at least one form module. When you add a new form to an application, Visual Basic adds a new form module to go with the form.

### 3.3 Variables and Constants

#### 3.3.1 Variables

Variables are used by Visual Basic to hold information needed by your application. Rules used in naming variables:

- ❖ **No more than 40 characters**
  - They may include letters, numbers, and underscore ( \_ )
- ❖ **The first character must be a letter**
  - You cannot use a reserved word (word needed by Visual Basic)
- ❖ **No spacing is allowed**
- ❖ **Period is not permitted**

**Examples of valid and invalid variable names are displayed in Table below**

Valid Name	Invalid Name
My_Car	My.Car
ThisYear	1NewBoy
Long_Name_Can_beUSE	He&HisFather      *& is not acceptable

### Data Types

Data falls into three broad categories: **numeric, string, and special.**

**A string:** - is a series of characters that you treat as a single entity. VB supports both fixed-length and variable-length strings.

Table below describe data type.

DataTypes	Description and Range
Boolean	A data type that takes on one of two values only: True or False.
Byte	Positive numeric values without decimals that range from 0 to 255.
Currency	Data that holds dollar amounts from \$922,337,203,685,477.5808 to - \$922,337,203,685,477.5807.
Date	Holds date and time values.
Double	Numeric values that range from -1.79769313486232E+308 to 1.79769313486232E+308.
Integer	Numeric values with no decimal point or fraction that range from – 32,768 to 32,767.
Long	Integer values with a range beyond that of Integer data values. Long data values range from -2,147,483,648 to 2,147,483,647. Long data values consume more memory storage than integer values, and they are less efficient.
Single	Numeric values that range from -3.402823E+38 to 3.402823E+38.
String	Data that consists of 0 to 65,400 characters of alphanumeric data. Alphanumeric means that the data can be both alphabetic and numeric. String data values may also contain special characters such as ^, %, and @. Both fixed-length strings and variable-length strings exist
Variant	Data of any data type and used for control and other values for which the data type is unknown



### Data Type Suffix

You can use the data type suffix characters from Table below to specify the data type for numeric literal .Therefore, if you type 86&, Visual Basic treats the number 86 as a long integer.

Data Type	Suffix
Boolean	None
Integer	%
Long (Integer)	&
Single (Floating)	!
Double (Floating)	#
Currency	@
Date	None
Object	None
String	\$
Variant	None

**Ex:**

**TextValue\$ = "This is a string"**

creates a string variable, while

**Amount% = 300**

creates an integer variable.

**Note:-**

1. All String literal data contains surrounding quotation marks. . The following are literal that take the String data type:

"Sams"                      "123 E. Sycamore St."                      "91829"  
 "#\$%^&\*"    "[Adam]"    "Happy birthday!"                      ""

The last string is called an **empty string** or a **null string** because the quotation marks are together without even a space between them.

2. You must embed date and time literal (Visual Basic uses the Date data type to hold these values) inside pound signs (#). Depending on your International settings, you can specify the date or time in just about any valid date or time format, as in the following:

#12-Jan-1999#    #14:56#    #2:56 PM#    #December 5, 1998#

**Literals:** - are values that you assign to a data.

**Constant:** - is another name for literal.

### Variables Hold Data

To hold data that might change due to calculations or state changes within the application, you must declare variables. A variable is a named location that holds data.

Variables, unlike literals, can change. In other words, you can store a number in a variable early in the program and then change that number later in the program. The variable acts like a box that holds a value.

### Variable Declaration

There are three ways for a variable to be typed (declared):

1. Default

2. Implicit

3. Explicit

- ❖ If variables are **not implicitly** or **explicitly typed**, they are assigned the variant type by default. The variant data type is a special type used by Visual Basic that can contain **numeric**, **string**, or **another date data**.
- ❖ To **implicitly type** a variable, use the corresponding suffix shown above in the data type table. **For example**,

*TextValue\$ = "This is a string"*

creates a string variable, while

*Amount% = 300*

creates an integer variable.

- ❖ To **explicitly type** a variable, you must first determine its scope. **There are four levels of scope:**

1. Procedure level
2. Procedure level, static
3. Form and module level
4. Global level

- 1- Within a procedure, variables are declared using the **Dim statement**:

For example

*Dim MyInt as Integer*

Dim MyDouble as Double

Dim MyString, YourString as String

You use the **Dim statement** to declare variables (Dim stands for dimension). The Dim statement defines variables. Dim describes the data type and also assigns a name to the variable. **Format of the Dim statement:**

**Dim VarName As DataType**

If you use Dim in Procedure level, a variable will only exist within the procedure in which it was declared. (As an example, if we'd declare a variable within a button-click event, as soon as all code within that event is processed, the variable will cease to exist, and the data inside the variable will be lost).

- 2- To make a procedure level variable store a value (retain a value) and remain active within the procedure it's created as long as the program is running. Replace the Dim keyword with Static:

**Static MyInt as Integer**

**Static MyDouble as Double**

- 3- Form level variables retain their value and are available to all procedures within that form (global in form). Form level variables are declared in the top of a form module in the section called general section that appears before all event procedures) in the form's code window. The Dim keyword is used:

**Dim MyInt as Integer**

**Dim MyDate as Date**

- 4- Global level variables retain their value and are available to all procedures within an application. Global level variables are declared in top of a standard

module (in a section called the general section that appears before all procedures)

Use the **Global** or **Public** keyword: for example

**Global** MyInt **as** Integer

**Global** MyDate **as** Date

Or

**Public** strInput **as** String

**Note:-**

Standard module variables are almost always **globally** defined with Public or global so that other modules within a project you add the standard module to can access the variables.

We can write special statement called the **Option Explicit** statement at the very top of a **form module** or at the top of a **standard module** The **Option Explicit** statement forces us to declare all variables used in the form module or standard module .To do this follow these steps

- 1- Double-click anywhere on the form to open the code window. Or, select 'View Code' from the project window.
- 2- Click the down arrow in the Object box and select the object named (general). The Procedure box will show (declarations). Here, write Option Explicit and you declare all variables in the general declarations area of your form. This makes them available to all the form procedures:

**Example:-****Option Explicit**

Dim StartTime As Variant

Dim ElapsedTime As Variant

In general declarations area of form

**Example:-***General section***Example of Variable Scope:****Global X as Integer**

Form1

**Dim Y as Integer**

Sub Routine1()

**Dim A as Double**

..

..

**End Sub**

Sub Routine2()

**Static B as Double**

.

.

End Sub

Form2

**Dim Z as Single**

Sub Routine3()

**Dim C as String**

..

..

**End Sub**

Procedure Routine1 has access to X, Y, and A (loses value upon termination)

Procedure Routine2 has access to X, Y, and B (retains value)

Procedure Routine3 has access to X, Z, and C (loses value)

### **WARNING:-**

Never declare two variables with the same name in the same location. That is, you cannot declare two variables with the name **intNumber** in the same event procedure.

**Global variables:** - are variables that are available to the entire module or to the entire application.

**Local variables:** - are variables that are available only to the procedure in which you define the variables.

### **Examples about Dim Statement**

- ❖ The following statements define Integer, Single, and Double variables:

**Dim** intLength **As** Integer

**Dim** sngPrice **As** Single

**Dim** dblStructure **As** Double

- ❖ If you want to write a program that stores the user's text box entry for the first name, you would define a string like this:

**Dim** strFirstName **As** String

This strFirstName string can hold any string from 0 to 65,500 characters long.

- ❖ The following Dim statement demonstrates how you can add the \* StringLength option when you want to define fixed-length strings:

**Dim** strTitle **As** String \* 20

**strTitle**:- is the name of a String variable that can hold a string from **0** to **20** characters long. If the program attempts to store a string value that is longer than 20 characters in **strTitle**, Visual Basic truncates the string and stores only the first 20 characters.

❖ when you define Variant variables. This Dim statement:

**Dim** varValue **As** Variant

does exactly the same thing as this:

**Dim** varValue

**Note: -**

If you begin calling a variable one name, you must stay with that name for the entire program. curSale is not the same variable name as curSales. Use **Option Explicit** to guard against such common variable-naming errors.

❖ Instead of listing each variable definition on separate lines like this:

**Dim** A **As** Integer

**Dim** B **As** Double

**Dim** C **As** Integer

**Dim** D **As** String

**Dim** E **As** String

You can combine variables of the same data type on one line. Here's an

**Example:-**

**Dim** A **As** Integer, C **As** Integer

**Dim** B **As** Double

**Dim** D **As** String, E **As** String



### 3.3.2 Constants

Well, as we know now, variables are used for temporarily storing data. What though if we would have a variable that stays the same within the entire of the application, and is used all-through the application? We can use a specific type of variable called a *Constant* to achieve exactly this.

A constant is declare like so:

**Const** cstrFilename = any data type

As you see, we immediately assign the value of the constant to it. This is because the value will be constant. It will **not** change.

You can also define your own constants for use in Visual Basic. The format for defining a constant named PI with a value 3.14159 is:

Const PI = 3.14159

User-defined constants should be written in all upper case letters to distinguish them from variables. The scope of constants is established the same way a variables' scope is. That is, if defined within a procedure, they are local to the procedure. If defined in the general declarations of a form, they are global to the form. To make constants global to an application, use the format:

**Global Const PI = 3.14159**

within the general declarations area of a module

### Putting Data in Variables

So far you have learned how to define variables but not how to store data in them.

Use the **assignment statement** when you want to put data values into variables.

Here is the **format of the assignment statement**.

**VarName = Expression**

Where VarName is a variable name that you have defined using the Dim statement, followed by the **assignment operator (=)**, followed by **Expression** which can be a **literal**, **another variable**, or a **mathematical expression**.

**An assignment statement:** - is a program statement that puts data into a control, a variable, or another object.

**Examples about (assignment statement)**

1. To store a minimum age value of 18 in an Integer variable named MinAge. The following assignment statement does that:

**MinAge = 18**

2. To store the string welcome in string variable named AB . The following assignment statement dose that .

**AB = “ welcome “**

**Note:-**

The data type of Expression must match the data type of the variable to which you are assigning it. In other words, the following statement is invalid. It would produce an error in Visual Basic programs.

```
Dim AB As String }
AB= 55             Invalid
```

3. You can assign other variables to variables. Consider the following code:

Dim A1 As Single, B1 As Single

A1= 3945.42

B1 = A1

When the third statement finishes, both A1 and B1 have the value 3945.42

4. To assign **variables** to **controls** and **controls** to **variables**. Suppose, for example, that the user types the value 18.34 in a **text box's Text property**. If the **text box's Name property** is **txtFactor**, the following statement stores the value of the text box in a variable named **sngFactorVal**:

```
sngFactorVal = txtFactor.Text
```

#### Note:-

1- Statements normally take up a single line with no terminator. Statements can be stacked by using a colon (:) to separate them. Example:

```
StartTime = Now : EndTime = StartTime + 10
```

2-If a statement is very long, it may be continued to the next line using the continuation character, an underscore (\_). Example:

```
Months = Log(Final * IntRate / Deposit + 1) _  
          / Log(1 + IntRate)
```

#### Comment statements

It begin with the keyword **Rem** or a single quote ('). For example:

```
Rem This is a sub to add two number
```

or

```
' This is a sub to add two number
```

or

```
x = 2 * y '           another way to write a remark or comment
```

### Expressions and Math Operators

In order to compute inputs from users and to generate results, we need to use various mathematical operators. Mathematical operators, as shown in Table below.

**Table 6.1**

Operator	Mathematical function	Example
^	Exponential	$2^4=16$
*	Multiplication	$4*3=12$
/	Division	$12/4=3$
Mod	Modulus(return the remainder from an integer division)	$15 \text{ Mod } 4=3$
\	Integer Division(discards the decimal places)	$19 \setminus 4=4$
+ or &	String concatenation	"Visual"&"Basic"="Visual Basic"

#### Example:-

```
firstName=Text1.Text
```

```
secondName=Text2.Text
```

```
yourName=firstName+secondName
```

```
number1=val(Text3.Text)
```

```
number2=val(Text4.Text)
```

```
number3=num1*(num2^3)
```

```
number4=number3 Mod 2
```

```
number5=number4\number1
```

```
Total=number1+number2+number3+number4+number5
```

```
Average=Total/5
```

**Combining expressions**:- often make Visual Basic computes mathematical results in a predetermined order. Visual Basic always calculates **exponentiation first** if one or more ^ operators appear in the expression. Visual Basic **then computes all multiplication and division**--working from **left to right**--before any **addition and subtraction**.

Visual Basic assigns 13 to intResult in the following assignment:

```
intResult = 3 + 5 * 2
```

At first, you might think that Visual Basic would assign 16 to intResult because 3 + 5 is 8 and 8 \* 2 is 16. However, the rules state that Visual Basic always computes multiplication--and division if division exists in the expression--before addition. Therefore, Visual Basic first computes the value of 5 \* 2, or 10, and next adds 3 to 10 to get 13. Only then does it assign the 13 to Result.

If both multiplication and division appear in the same expression, Visual Basic calculates the intermediate results from left to right. For example, Visual Basic assigns 20 to the following expression:

```
intResult = 8 / 2 + 4 + 3 * 4
```

Visual Basic computes the division first because the division appears to the left of the multiplication. If the multiplication appeared to the left of the division, Visual Basic would have multiplied first. After Visual Basic calculates the intermediate answers for the division and the multiplication, it performs the addition and stores the final answer of 20 in intResult.

**Note:-**

The order of computation has many names. Programmers usually use one of these terms: order of operators, operator precedence, or math hierarchy.

It is possible to override the operator precedence by using **parentheses**. Visual Basic always computes the values inside any **pair of parentheses** before anything else in the expression, even if it means ignoring operator precedence.

The following assignment statement stores 16 in `intResult` because the parentheses force Visual Basic to compute the addition before the multiplication:

```
intResult = (3 + 5) * 2
```