

# Visual Basic and Databases

## 1. Introducing Visual Basic and Databases

### Preview

- In this first chapter, we will do a quick overview of what the course entails. We will discuss what you need to complete the course. We'll take a brief look at what databases are, where they are used, and how Visual Basic is used with databases. And, we'll review the Visual Basic development environment and the steps followed to build an application in Visual Basic.

**Course Objectives**

- ⇒ Understand the benefits of using Microsoft Visual Basic to build a 'front-end' interface as a database programming tool
- ⇒ Learn database structure, terminology, and proper database design
- ⇒ Learn how to connect to a database using the Visual Basic DAO (data access object) control
- ⇒ Use the ADO (ActiveX data object) data control and data environment to connect to a database (Visual Basic 6 only)
- ⇒ Learn the use of Visual Basic data bound controls
- ⇒ Learn to make database queries using SQL (structured query language)
- ⇒ Understand proper database search techniques
- ⇒ Learn how to use the Visual Basic Data Manager to create a database
- ⇒ Learn database management techniques
- ⇒ Learn to create and produce database reports
- ⇒ Learn how to distribute a Visual Basic database application
- ⇒ Understand connection to remote databases
- ⇒ Introduce multiple-user and database security concepts

---

## Course Requirements

- An obvious requirement is a Windows-based computer with Windows 95, Windows 98, or Windows NT installed, as well as Visual Basic. The student should be familiar with the basics of using the Windows operating system.
- **Visual Basic and Databases** requires some edition of **Visual Basic 5** or **Visual Basic 6**. There are two controls used by Visual Basic to interact with databases: the **DAO** (data access object) control and the **ADO** (ActiveX data object) control. Both controls will be discussed in this course. You should be aware, however, that the ADO control is available only with Visual Basic 6
- Most examples presented in the course notes are done using the Professional Edition of Visual Basic 6. Hence, if you are using Visual Basic 5 or another edition of Visual Basic 6, some of your screens may differ from the ones seen in the notes.
- No knowledge of databases or how to work with databases is presumed. Adequate introductory material is presented. Even if you've worked with databases before, it is suggested you read through this introductory information to become acquainted with the nomenclature used by the author for databases and their component parts.
- This course does **not** teach you how to build a Visual Basic application. It is assumed that the student has a basic understanding of the Visual Basic development environment and knows the steps involved in building a Visual Basic application. You should feel quite comfortable with building the example application at the end of this first chapter. If not, our company, KIDware, offers several tutorials that teach this information. Please visit our web site or contact us for more information.

## What is a Database?

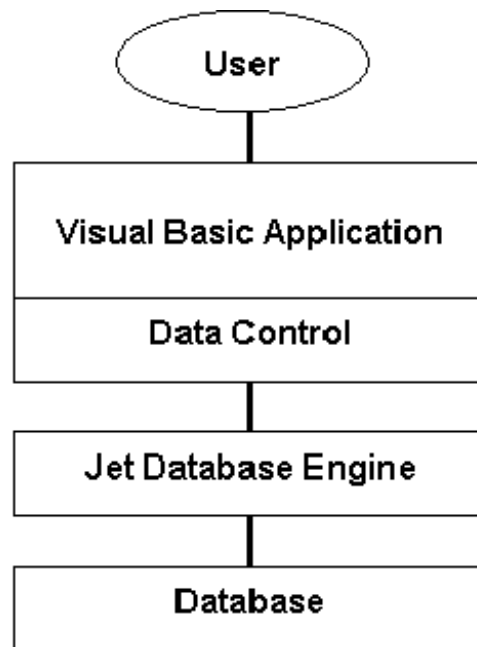
- A **database** is a collection of **information**. This information is stored in a very structured manner. By exploiting this known structure, we can access and modify the information quickly and correctly.
- In this information age, databases are everywhere:
  - ⇒ When you go to the library and look up a book on their computer, you are accessing the library's book **database**.
  - ⇒ When you go on-line and purchase some product, you are accessing the web merchant's product **database**.
  - ⇒ Your friendly bank keeps all your financial records on their **database**. When you receive your monthly statement, the bank generates a **database report**.
  - ⇒ When you call to make a doctor appointment, the receptionist looks into their **database** for available times.
  - ⇒ When you go to your car dealer for repairs, the technician calls up your past work record on the garage **database**.
  - ⇒ At the grocery store, when the checker scans each product, the price is found in the store's **database**, where inventory control is also performed.
  - ⇒ When you are watching a baseball game on television and the announcer tells you that "the batter is hitting .328 against left-handed pitchers whose mother was born in Kentucky on a Tuesday morning," that useless information is pulled from the team's **database**.
- You can surely think of many more places that databases enter your life. The idea is that they are everywhere. And, each database requires some way for a user to interact with the information within. Such interaction is performed by a **database management system (DBMS)**.
- The tasks of a **DBMS** are really quite simple. In concept, there are only a few things you can do with a database:
  1. View the data
  2. Find some data of interest
  3. Modify the data
  4. Add some data
  5. Delete some data

There are many commercial database management systems that perform these tasks. Programs like Access (a Microsoft product) and Oracle are used world-wide. In this course, we look at using **Visual Basic** as a **DBMS**.

- Examples where you might use Visual Basic as a DBMS:
  - ⇒ Implementing a new application that requires management of a database
  - ⇒ Connecting to an existing database
  - ⇒ Interacting with a database via the internet
- In a DBMS, the database may be available **locally** on your (or the user's) computer, available on a **LAN** (local area network) shared by multiple users, or only available on a **web server** via the Internet. In this course, we spend most of our time looking at local databases, but access with remote databases is addressed.
- We will look at databases in more depth in the next chapter. You will see that databases have their own vocabulary. Now, let's take a look at how Visual Basic fits into the database management system.

## Where Does Visual Basic Fit In?

- For database management, we say our Visual Basic application acts as a **front-end** to the database. This means the Visual Basic application provides the **interface** between the user and the database. This interface allows the user to tell the database what he or she needs and allows the database to respond to the request displaying the requested information in some manner.
- A Visual Basic application cannot directly interact with a database. There are two intermediate components between the application and the database: the **data control** and the **database engine**:



- The **data control** is a Visual Basic object that connects the application to the database via the database engine. It is the conduit between the application and the engine, passing information back and forth between the two.
- The **database engine** is the heart of a Visual Basic database management system. It is the actual software that does the management. Having this engine saves programmers a lot of work. The database engine native to Visual Basic is known as the **Jet** engine. It is the same engine used by Microsoft Access for database management. Hence, it is primarily used to work with Access databases, but it can also work with others.

- As mentioned, the Jet engine will save us lots of work. An observation that illustrates the power of using Visual Basic as a front-end for database management systems:

Using Visual Basic, it requires less code to connect to an existing database, view all information within that database, and modify any and all information within that database, than it does to add two numbers together.

That's right - all the database tasks mentioned above can be done without writing one line of code! That's the power of the Jet database engine!

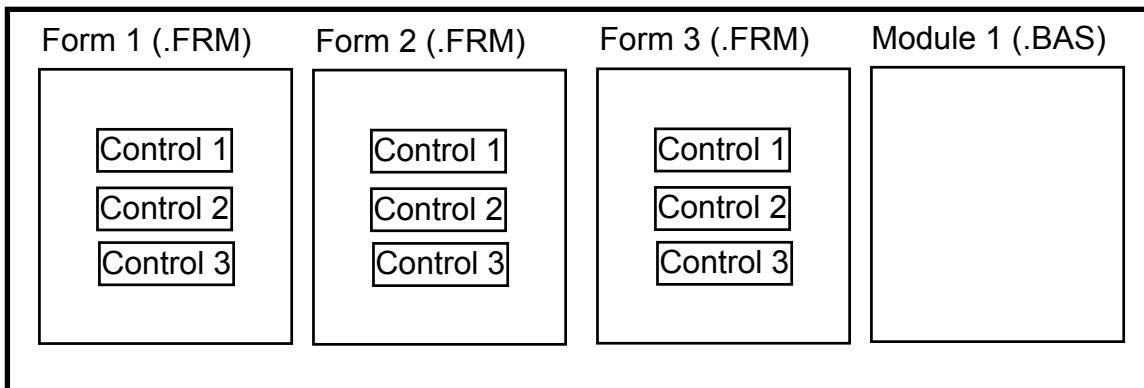
- So, if the Jet engine is so powerful and is the same engine used by Microsoft Access, why not just use Access as a DBMS instead of writing a custom Visual Basic application? There are two primary advantages to using Visual Basic as a DBMS instead of Access:
  1. Your users don't need to have Access installed on their computers or know how to use Access.
  2. By building a custom front-end, you limit what your user can do with the information within the database. Under normal operation, Access provides no such limits.
- So, in this course, we will look at how to build Visual Basic applications that operate as front-ends to databases. Research has shown that over half of all Visual Basic applications involve working with databases. We will look at how to make our applications into complete database management systems, being able to view, search, modify, add, and/or delete database information.
- Before going any further, let's review the steps in building a Visual Basic application and then build a simple application for practice.

## Building a Visual Basic Application

- In the remainder of this chapter, we will provide an overview of a Visual Basic application and how the Visual Basic development environment is used to develop an application. This should provide you with some idea of what knowledge you need to possess to proceed in this course and introduce the terminology used by the author to describe a Visual Basic application.

## Structure of a Visual Basic Application

Project (.VBP)



**Application** (Project - saved as a file with a .VBP extension) is made up of:

- ⇒ **Forms** - Windows that you create for user interface (saved as a file with a .FRM extension).
- ⇒ **Controls** - Graphical features drawn on forms to allow user interaction (text boxes, labels, scroll bars, command buttons, etc.) (Forms and Controls are also called **objects**.)
- ⇒ **Properties** - Every characteristic of a form or control is specified by a property. Example properties include names, captions, size, color, position, and contents. Visual Basic applies default properties. You can change properties at design time or run time.
- ⇒ **Methods** - Built-in procedure that can be invoked to impart some action to a particular object.
- ⇒ **Event Procedures** - Code related to some object. This is the code that is executed when a certain event occurs.
- ⇒ **General Procedures** - Code not related to objects. This code must be invoked by the application.
- ⇒ **Modules** - Collection of general procedures, variable declarations, and constant definitions used by application (saved as a file with a .BAS extension).



## Steps in Developing Application

- There are three primary steps involved in building a Visual Basic application:
  1. **Draw** the user **interface**
  2. **Assign properties** to controls
  3. **Write code** for event procedures. Develop any needed general procedures.

We'll look at each step.

## Drawing the User Interface and Setting Properties

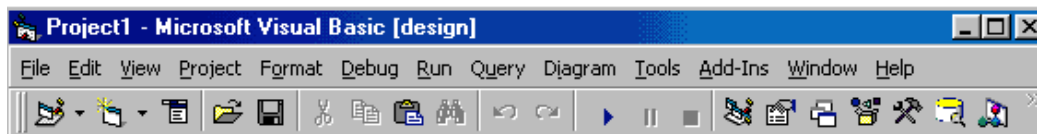
- Visual Basic operates in three modes.
  - ⇒ **Design** mode - used to build application
  - ⇒ **Run** mode - used to run the application
  - ⇒ **Break** mode - application halted and debugger is available

We focus here on the **design** mode.

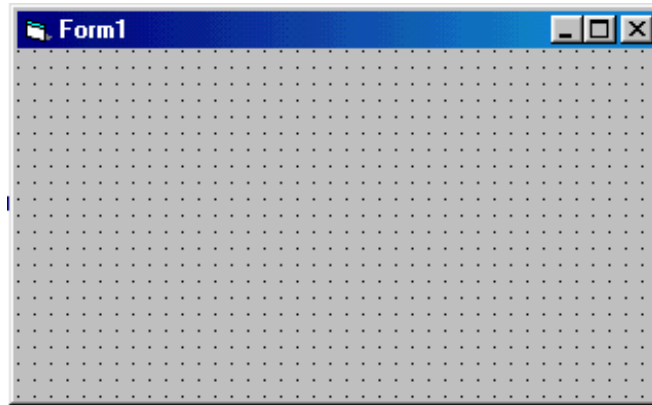
- Six windows should appear when you start Visual Basic. If any of these windows do not appear, they may be accessed using the main window menu **View** item.

- ⇒ The **Main Window** consists of the title bar, menu bar, and toolbar.

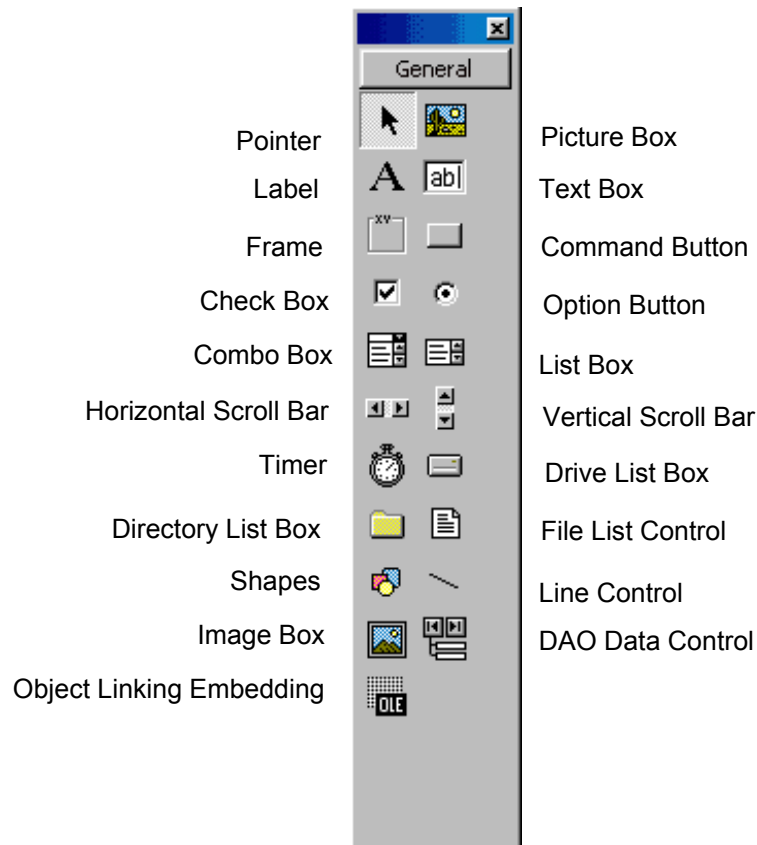
The title bar indicates the project name, the current Visual Basic operating mode, and the current form. The menu bar has drop-down menus from which you control the operation of the Visual Basic environment. The toolbar has buttons that provide shortcuts to some of the menu options (ToolTips indicate their function). The main window also shows the location of the current form relative to the upper left corner of the screen (measured in twips) and the width and length of the current form.



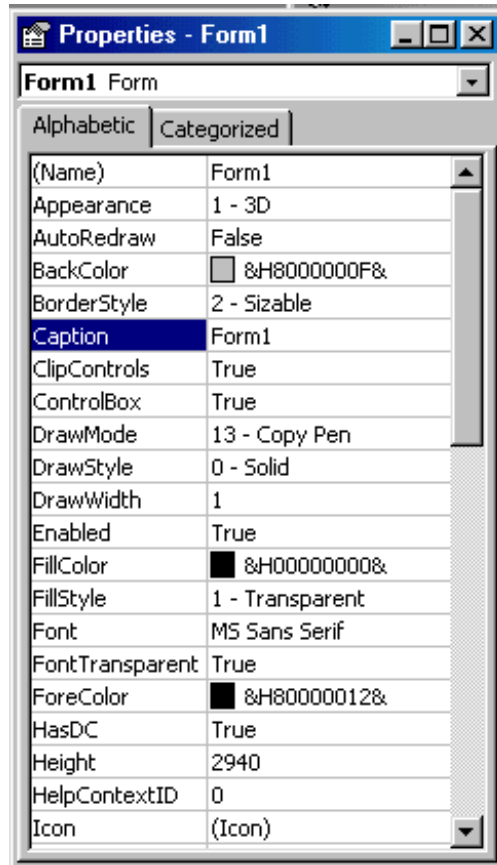
- ⇒ The **Form Window** is central to developing Visual Basic applications. It is where you draw your application.



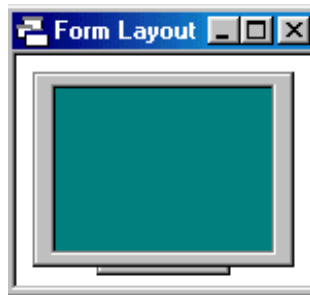
- ⇒ The **Toolbox** is the selection menu for controls (objects) used in your application.



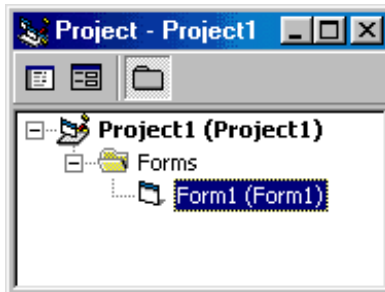
- ⇒ The **Properties Window** is used to establish initial property values for objects. The drop-down box at the top of the window lists all objects in the current form. Two views are available: **Alphabetic** and **Categorized**. Under this box are the available properties for the currently selected object.



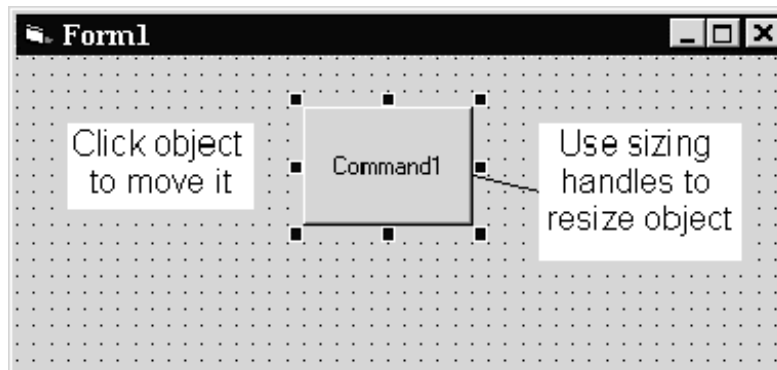
- ⇒ The **Form Layout Window** shows where (upon program execution) your form will be displayed relative to your monitor's screen:



- ⇒ The **Project Explorer Window** displays a list of all forms and modules making up your application. You can also obtain a view of the **Form** or **Code** windows (window containing the actual Basic coding) from the Project Explorer window.

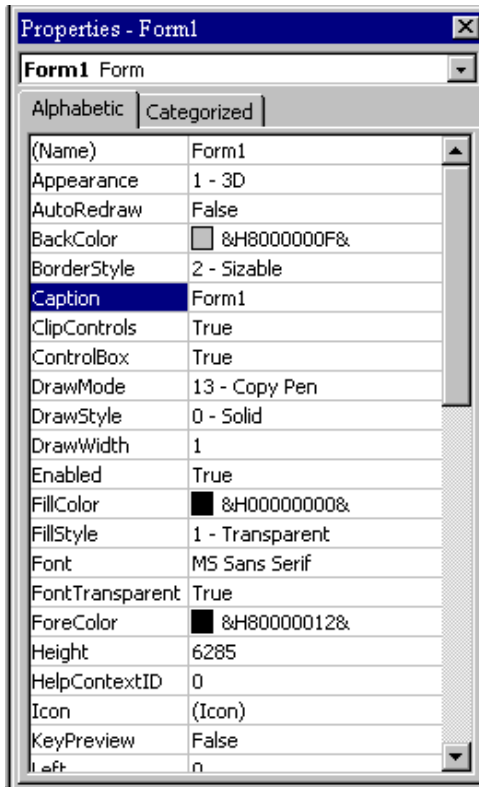


- As mentioned, the user interface is 'drawn' in the form window. There are two ways to place controls on a form:
  1. Double-click the tool in the toolbox and it is created with a default size on the form. You can then move it or resize it.
  2. Click the tool in the toolbox, and then move the mouse pointer to the form window. The cursor changes to a crosshair. Place the crosshair at the upper left corner of where you want the control to be, press the left mouse button and hold it down while dragging the cursor toward the lower right corner. When you release the mouse button, the control is drawn. This approach must be used to place controls in a frame or picture box control.
- To **move** a control you have drawn, click the object in the form window and drag it to the new location. Release the mouse button.
- To **resize** a control, click the object so that it is select and sizing handles appear. Use these handles to resize the object.



## Setting Properties of Controls at Design Time

- Each form and control has **properties** assigned to it by default when you start a new project. There are two ways to display the properties of an object. The first way is to click on the object (form or control) in the form window. Then, click on the Properties Window or the Properties Window button in the tool bar. The second way is to first click on the Properties Window. Then, select the object from the **Object** box in the Properties Window. Shown is the Properties Window for a new application:



The drop-down box at the top of the Properties Window is the **Object** box. It displays the name of each object in the application as well as its type. This display shows the **Form** object. The **Properties** list is directly below this box. In this list, you can scroll through the list of properties for the selected object. You may select a property by clicking on it. Properties can be changed by typing a new value or choosing from a list of predefined settings (available as a drop down list). Properties can be viewed in two ways: **Alphabetic** and **Categorized**.

A very important property for each object is its **name**. The name is used by Visual Basic to refer to a particular object in code.

- A convention has been established for naming Visual Basic objects. This convention is to use a three-letter prefix (depending on the object) followed by a name you assign. A few of the prefixes are (we'll see more as we progress in the course):

Object	Prefix	Example
Form	frm	frmWatch
Command Button	cmd, btn	cmdExit, btnStart
Label	lbl	lblStart, lblEnd
Text Box	txt	txtTime, txtName
Menu	mnu	mnuExit, mnuSave
Check box	chk	chkChoice
Data control	dat	datExample

- Object names can be up to 40 characters long, must start with a letter, must contain only letters, numbers, and the underscore (\_) character. Names are used in setting properties at run time and also in establishing procedure names for object events.

## Setting Properties at Run Time

- In addition to setting control properties in design mode, you can set or modify properties while your application is running (run mode). To do this, you must write some code. The code format is:

**ObjectName.Property = NewValue**

Such a format is referred to as dot notation. For example, to change the **BackColor** property of a form name **frmStart**, we'd type:

```
frmStart.BackColor = vbBlue
```

- Using the three-letter prefix when naming an object and using an appropriate name makes reading such code easier and more meaningful.

## How Names are Used in Object Events

- The names you assign to objects are used by Visual Basic to set up a framework of event-driven procedures for you to add code to. The format for each of these subroutines (all object event procedures in Visual Basic are subroutines) is:

**Sub ObjectName\_Event (Optional Arguments)**

.  
.

**End Sub**

Visual Basic provides the **Sub** line with its arguments (if any) and the **End Sub** statement. You provide any needed code.

- Using the three-letter prefix when naming an object and using a meaningful name makes finding appropriate event procedures a simpler task.

## Writing Code

- The last step in building a Visual Basic application is to write code using the **BASIC** language. This is the most time consuming task in any Visual Basic application, not just ones involving databases. As objects are added to the form, Visual Basic automatically builds a framework of all event procedures. We simply add code to the event procedures we want our application to respond to. And, if needed, we write general procedures.
- Code is placed in the **code window**. At the top of the code window are two boxes, the **object** (or control) **list** and the **procedure list**. Select an object and the corresponding event procedure. A blank procedure will appear in the window where you write BASIC code.

## Review of 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)

## Visual Basic Data Types

- ⇒ Boolean (True or False)
- ⇒ Integer (Whole numbers)
- ⇒ Long (Large whole numbers)
- ⇒ Single (Floating point numbers)
- ⇒ Double (Large floating point numbers)
- ⇒ Currency
- ⇒ Date
- ⇒ Object (yes, objects can be variables!)
- ⇒ String (Used for many control properties)
- ⇒ Variant (A chameleon, becomes what it needs to be)

## Variable Declaration

- There are three ways for a variable to be typed (declared):
  1. Default (Variant type)
  2. Implicit (old technology)
  3. Explicit
- There are many advantages to **explicitly** typing variables. Primarily, we insure all computations are properly done, mistyped variable names are easily spotted, and Visual Basic will take care of insuring consistency in upper and lower case letters used in variable names. Because of these advantages, and because it is good programming practice, we will explicitly type all variables.
- To **explicitly** type a variable, you must first determine its **scope**. There are four levels of scope:
  - ⇒ Procedure level
  - ⇒ Procedure level, static
  - ⇒ Form and module level
  - ⇒ Global level

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

```
Dim MyInt As Integer
Dim MyDouble As Double
Dim MyString As String, YourString As String
```

Procedure level variables declared in this manner do not retain their value once a procedure terminates.

- To make a procedure level variable retain its value upon exiting the procedure, replace the Dim keyword with **Static**:

```
Static MyInt As Integer
Static MyDouble As Double
```

- Form (module) level variables retain their value and are available to all procedures within that form (module). Form (module) level variables are declared in the **declarations** part of the **general** object in the form's (module's) code window. The **Dim** keyword is used:

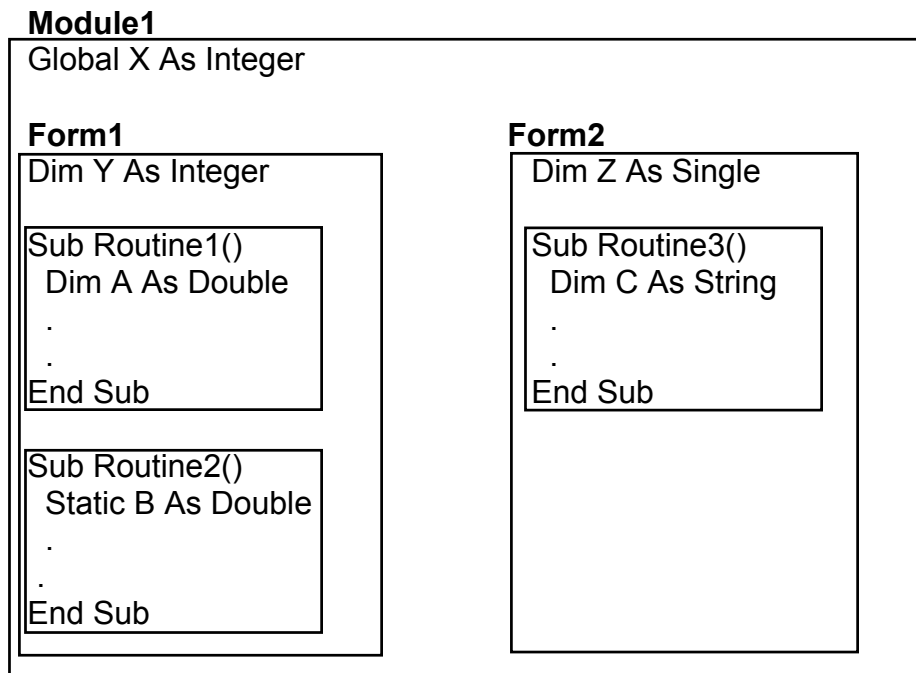
```
Dim MyInt As Integer
Dim MyDate As Date
```



- Global level variables retain their value and are available to all procedures within an application. Module level variables are declared in the **declarations** part of the **general** object of a module's code window. (It is advisable to keep all global variables in one module.) Use the **Global** keyword:

```
Global MyInt As Integer
Global MyDate As Date
```

- What happens if you declare a variable with the same name in two or more places? More local variables **shadow** (are accessed in preference to) less local variables. For example, if a variable MyInt is defined as Global in a module and declared local in a routine MyRoutine, while in MyRoutine, the local value of MyInt is accessed. Outside MyRoutine, the global value of MyInt is accessed.
- Example of Variable Scope:



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

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

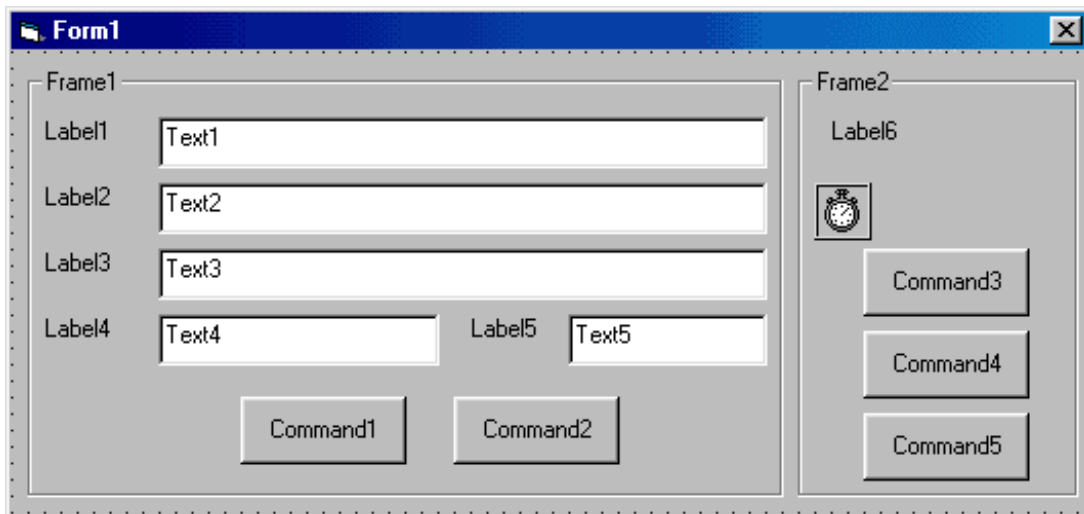
Procedure Routine3 has access to X, Z, and C (loses value)

### Example 1

#### Mailing List Application

In this example, we will build a Visual Basic application that could function as a database interface. The application allows the entry of information (names and addresses) to build a mailing list. An added feature is a timer that keeps track of the time spent entering addresses. After each entry, rather than write the information to a database (as we would normally do), the input information is simply displayed in a Visual Basic message box. We present this example to illustrate the steps in building an application. If you feel comfortable building this application and understanding the corresponding code, you probably possess the Visual Basic skills needed to proceed with this course.

1. Start a new project. Place two frames on the form (one for entry of address information and one for the timing function). In the first frame, place five labels, five text boxes, and two command buttons. In the second frame, place a label control, a timer control and three command buttons. Remember you need to 'draw' controls into frames. Resize and position controls so your form resembles this:



2. Set properties for the form and controls (these are just suggestions – make any changes you might like):

**Form1:**

Name	frmMailingList
BorderStyle	1-Fixed Single
Caption	Mailing List Application

**Frame1:**

Name	fraMail
Caption	Address Information
Enabled	False

**Label1:**

Caption	Name
---------	------

**Label2:**

Caption	Address
---------	---------

**Label3:**

Caption	City
---------	------

**Label4:**

Caption	State
---------	-------

**Label5:**

Caption	Zip
---------	-----

**Text1:**

Name	txtInput
Index	0 (a control array)
TabIndex	1
Text	[blank it out]

**Text2:**

Name	txtInput
Index	1
TabIndex	2
Text	[blank it out]

**Text3:**

Name	txtInput
Index	2
TabIndex	3
Text	[blank it out]

**Text4:**

Name	txtInput
Index	3
TabIndex	4
Text	[blank it out]

**Text5:**

Name	txtInput
Index	4
TabIndex	5
Text	[blank it out]

**Command1:**

Name	cmdAccept
Caption	&Accept
TabIndex	6

**Command2:**

Name	cmdClear
Caption	&Clear

**Frame1:**

Name	fraTime
Caption	Elapsed Time

**Label6:**

Name	lblElapsedTime
Alignment	2-Center
BackColor	White
BorderStyle	1-Fixed Single
Caption	00:00:00
FontBold	True
FontSize	14

**Timer1:**

Name	timSeconds
Enabled	False
Interval	1000

**Command3:**

Name	cmdStart
Caption	&Start

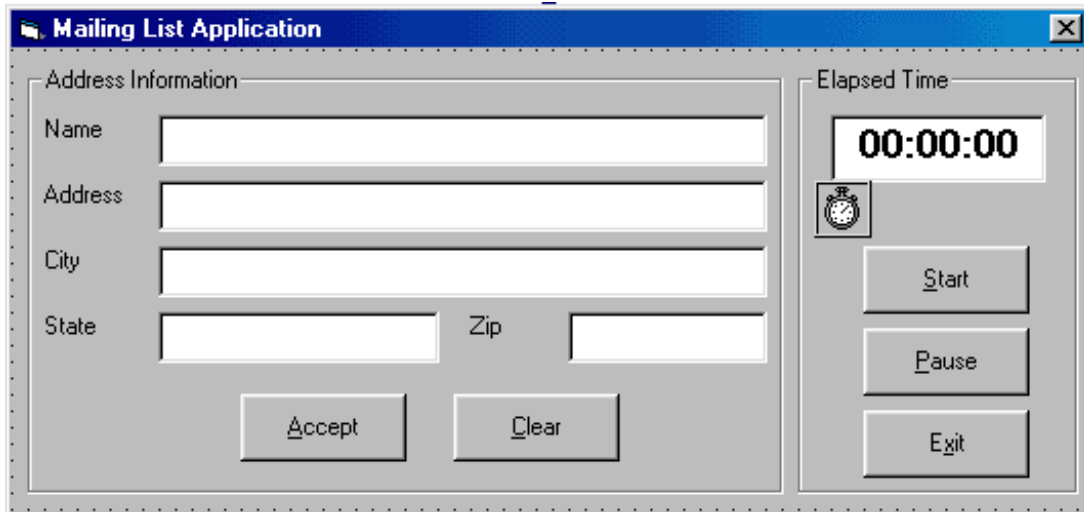
**Command4:**

Name	cmdPause
Caption	&Pause
Enabled	False

**Command5:**

Name	cmdExit
Caption	E&xit

When done, the form should appear something like this:



3. Put these lines in the **General Declarations** area of the code window:

```
Option Explicit  
Dim ElapsedTime As Variant  
Dim LastNow As Variant
```

4. Put these lines in the **Form\_Load** event procedure:

```
Private Sub Form_Load()  
    ElapsedTime = 0  
End Sub
```

5. Put this code in the **txtInput\_KeyPress** event procedure:

```
Private Sub txtInput_KeyPress(Index As Integer, KeyAscii  
As Integer)  
    'Check for return key  
    If KeyAscii = vbKeyReturn Then  
        If Index = 4 Then  
            cmdAccept.SetFocus  
        Else  
            txtInput(Index + 1).SetFocus  
        End If  
    End If  
End Sub
```

```

'In Zip text box, make sure only numbers or backspace
pressed
If Index = 4 Then
    If (KeyAscii >= Asc("0") And KeyAscii <= Asc("9")) Or
KeyAscii = vbKeyBack Then
        Exit Sub
    Else
        KeyAscii = 0
    End If
End If
End Sub

```

Note the line beginning with 'If (KeyAscii >= Asc("0") And ...' is so long that the word processor wraps the line around at the margin. Type this as one long line, not two separate lines or review the use of the Visual Basic line continuation character (\_). Be aware this happens quite often in these notes when actual code is being presented.

6. Put this code in the **cmdAccept\_Click** event procedure:

```

Private Sub cmdAccept_Click()
Dim S As String, I As Integer
'Accept button clicked - form label and output in message
box
'Make sure each text box has entry
For I = 0 To 4
    If txtInput(I).Text = "" Then
        MsgBox "Each box must have an entry!", vbInformation +
vbOKOnly, "Error"
        Exit Sub
    End If
Next I
S = txtInput(0).Text + vbCrLf + txtInput(1).Text + vbCrLf
S = S + txtInput(2).Text + ", " + txtInput(3).Text + " " +
txtInput(4).Text
MsgBox S, vbOKOnly, "Mailing Label"
Call cmdClear_Click
End Sub

```

7. Put this code in the **cmdClear\_Click** event procedure:

```
Private Sub cmdClear_Click()  
Dim I As Integer  
'Clear all text boxes  
For I = 0 To 4  
    txtInput(I).Text = ""  
Next I  
txtInput(0).SetFocus  
End Sub
```

8. Put this code in the **cmdStart\_Click** event procedure:

```
Private Sub cmdStart_Click()  
'Start button clicked  
'Disable start and exit buttons  
'Enabled pause button  
cmdStart.Enabled = False  
cmdExit.Enabled = False  
cmdPause.Enabled = True  
'Establish start time and start timer control  
LastNow = Now  
timSeconds.Enabled = True  
'Enable mailing list frame  
fraMail.Enabled = True  
txtInput(0).SetFocus  
End Sub
```

9. Put this code in the **cmdPause\_Click** event procedure:

```
Private Sub cmdPause_Click()  
'Pause button clicked  
'Disable pause button  
'Enabled start and exit buttons  
cmdPause.Enabled = False  
cmdStart.Enabled = True  
cmdExit.Enabled = True  
'Stop timer  
timSeconds.Enabled = False  
'Disable editing frame  
fraMail.Enabled = False  
End Sub
```

10. Put this code in the **cmdExit\_Click** event procedure:

```
Private Sub cmdExit_Click()  
    'Exit button clicked  
End  
End Sub
```

11. Put this code in the **timSeconds\_Timer** event procedure:

```
Private Sub timSeconds_Timer()  
    'Increase elapsed time and display  
    ElapsedTime = ElapsedTime + Now - LastNow  
    lblElapsedTime.Caption = Format(ElapsedTime, "hh:mm:ss")  
    LastNow = Now  
End Sub
```

12. Save the application. Run the application. Make sure it functions as designed.  
Note that you cannot enter mailing list information unless the timer is running.



## **Summary**

- In this chapter, we introduced databases in general terms and how Visual Basic can be used to develop a front-end application to interact with the database. And, we reviewed the steps involved in building a Visual Basic application.
- In the second chapter, we take a closer look at databases. We look at their structure, their terminology, and how they are constructed. You may be asking - when do we get to do some programming? The answer - in a couple more chapters. We want to make sure we have a firm foundation in place before diving into actual coding.

This page intentionally not left blank.