

Advanced Visual Basic

Course Designer and Acquisition Editor

Centre for Information Technology and Engineering

Manonmaniam Sundaranar University

Tirunelveli

Lecture - 1

Client / Server

Objectives

In this lecture you will learn the following

- ❖ About Client
- ❖ About Server
- ❖ About Client / Server computing
- ❖ Client / Server Model

Lecture Unit - 1

- 1.1 Snap Shot
- 1.2 Client
- 1.3 Server
- 1.4 Client/ Server Computing
- 1.5 Client/Server Model
- 1.6 Short Summary
- 1.7 Brain Storm

Lab unit 1 - (2 Real Time Hrs)

1.1 Snap Shot

Any time two computers are involved in the mutual performance of executing an application, with each performing a different function, you are undoubtedly looking at a client/server application. Many definitions of client/server are used. A definition of client/server application is an application that has a client interface and that accesses data on a remote server. The work is distributed between the client system and the remote server system, based on the capabilities of the client and the system and the remote server system, based on the capabilities of the client and server software applications. Client/server systems usually are efficient because network traffic is minimized and each portion of the application is optimized for a particular function.

1.2 Client

A client may be either a device or a user on a network that takes advantages of the services offered by a server. Client is often used in a loose way to refer to a computer on the network. It also refers to a user that is running the client side of a client/server application. The Client area of a window is where end users enter data. Formally, the client is the region of a window that doesn't include the title bar, menus, toolbars, status bars or window borders. In Visual Basic, the width and height to the client area can be determined by examining the value of a form's Scale width and Scale height properties, respectively.

1.3 Server

A server is a network-connected computer system that provides services to network users. Servers may be a file server, application server, database server, e-mail gateways and communication server. These systems run network operating systems such as Novell Netware, Windows NT or a version of UNIX OS.

You can use VBScript to create Active Server Pages(ASP) in order to create logic that enables IIS to respond to various input from client computers. Also, you can use Visual Basic to create custom ActiveX components designed to work as extensions and enhancements to the IIS environment. These components are called **server -side components**.

1.4 Client / Server Computing

Client / Server computing defines an architecture for designing programs that distribute their processing load between a client computer and a server computer

and sharing the processing load between two. The client requests services, and the server provides the requested services.

In contrast to the file-server environment, the server in this environment is responsible for intelligent service to the client. The client does not request data at a file level, but sends a request to the server to execute a query and return specific records. This is a vast improvement over the file-server approach.

The following figure 1.1 shows typical Client-Server configuration.

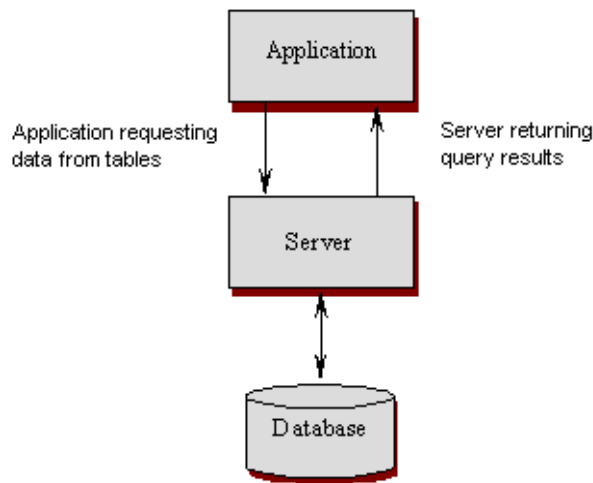


Figure 1.1 A typical Client-Server configuration.

Client side versus server side

If you write a program that resides and executes on a client computer, that program is called a client-side program. If you create a program that runs on a Internet server, that program is called a server-side program. Microsoft Word is considered a client-side program. Yahoo Search, a program that searches for articles on the Web, is a sever-side program.

1.5 Client / Server Model

Architecture that splits an application into a front-end client component and a back-end server component. The client component running on a workstation gathers data from the user, prepares it for the server, and issues a request to the server. On the back end, the server waits for requests from its clients. When it receives a request, the server processes it and returns the requested information to the client. The client then presents the data to the user through its own user interface.

1.5.1 Two - Tier Client/Server Model

The client communicates directly with the server with no intermediary. The limitation of this model is not easily scalable. If the server becomes over taxed by client traffic, the usual solution is to upgrade the server hardware to a fast processor with more memory. However, there is an upper limit to how fast the processor can be

1.5.2 Three - Tier Client /Server Model

Three - tier client/server design is frequently used in Internet applications. This increases the scalability of the application. An example of this three-tiered can be seen with a Web page that accesses a database on the server. Frequently, the database is on a different server than the Web server. This allows the distribution of the load over multiple systems. It also creates scalability. If the database activity becomes a bottleneck for the application, the database activity can be distributed over multiple servers. If the Web server becomes the bottleneck, the Web server can be distributed over multiple servers.

There are other uses of the three-tiered client/server model, including the following two very useful designs. The first design has a middle tier that provides an Online Analysis Processing data warehouse that is extracted from the base layer of operational databases. The second design makes use of a middle tier to enforce business rules. This separate layer for business rules makes the maintenance of the business rules much easier and less disruptive.

1.5.2.1 Three-Tiered Applications

To use Remote Data Service technology, you must understand the three-tiered client/server model. This model separates the various components of a client/server system into three "tiers":

- **Client tier**—a local computer on which either a Web browser displays a Web page that can display and manipulate data from a remote data source, or (in non-Web-based applications) a stand-alone compiled front-end application.
- **Middle tier**—a Microsoft Windows NT Server computer that hosts components that encapsulate an organization's business rules. Middle-tier components can be either Active Server Page scripts executed on Internet Information Server, or (in non-Web-based applications) compiled executables.
- **Data source tier**—a computer hosting a database management system (DBMS), such as a Microsoft SQL Server database. (In a two-tier application, the middle tier and data source tier are combined.)

These tiers don't necessarily correspond to physical locations on the network. For example, all three tiers may exist on only two computers. One computer could be a Microsoft Windows 95 computer running Microsoft Internet Explorer 4.0 as its browser. The second computer could be a Windows NT Server computer running both Internet Information Server and Microsoft SQL Server. Designing applications this way gives you greater flexibility when deploying processes and data on the network for maximum performance and ease of maintenance.

1.6 Short Summary

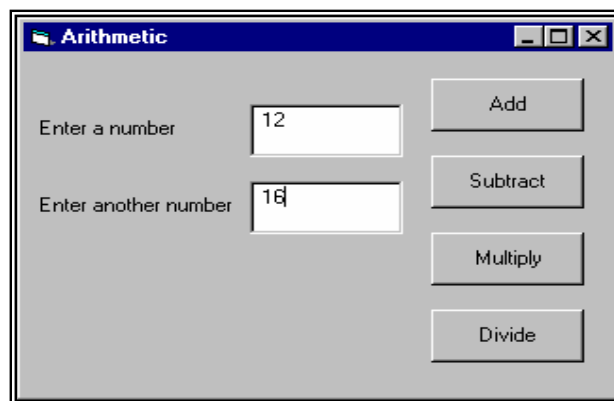
- ☞ Client request a server for its service
- ☞ Server waits for a client to be connected and serves them as soon as they are connect
- ☞ Distributes the processing load between the client and the server
- ☞ Network architecture can be two or three tier architecture

1.7 Brain Storm

1. With an example explain how a client / server application works ?
2. Differentiate Client Server Application over Stand alone application
3. With an example explain how a client / server works
4. How to process the Two tier & Three tier model?

Lab Unit - 1 (2 Real Time Hrs)

1. Open a new standard EXE project
2. Design your form as shown below
3. Result should be displayed depending upon the button pressed.



Lecture - 2

Get into Visual Basic 6.0

Objectives

In this lecture you will learn the following

- ❖ About Features of Visual Basic6
- ❖ New Object Oriented features
- ❖ Under About Integrated Development Environment
- ❖ How to develop a script in Visual Basic

Lecture unit - 2

- 2.1 Snap Shot
- 2.2 Features of Visual Basic 6
- 2.3 Integrated Development Environment
- 2.4 Basic Scripting in Visual Basic
- 2.5 Short Summary
- 2.6 Brain Storm

Lab unit 2 (2 Real Time Hrs)

2.1 Snap Shot

Visual Basic is the fastest and easiest way to create applications for Microsoft Windows. Visual Basic provides complete set of tools to simplify rapid application development both for the experienced professional and new Windows programmers.

In the name Visual Basic - the "Visual" part refers to the method used to create the graphical user interface (GUI). Unlike many languages which requires numerous lines of coding to describe the appearance and location of interface elements, Visual Basic provides pre-built objects that can be used to form the Graphical User Interface (GUI).

Note : The elements that can respond to the user actions such as mouse click, double click, drag, drop etc. are called interface elements.

The "Basic" part refers to the BASIC language as its basic syntax of statements is retained by Visual Basic. But Visual Basic now contains several hundred statements, functions, and keywords, many of which relate directly to the Windows GUI.

The Visual Basic programming language is not unique to Visual Basic. The Visual Basic programming system, Applications Edition included in Microsoft Excel, Microsoft Access, and many other Windows applications uses the same language. The Visual Basic Scripting Edition (VBScript) is a widely used scripting language and a subset of the Visual Basic language. So mastering Visual Basic also helps to master these other areas.

The Visual Basic comes in three flavors:

- ❖ The Visual Basic learning Edition
- ❖ The Visual Basic Professional Edition
- ❖ The Visual Basic Enterprise Edition

The Visual Basic Learning Edition is the introductory edition that lets to easily create windows applications. It comes with all the tools needed to build main stream windows applications.

The Visual Basic Professional Edition is for computer professionals and includes advanced features such as tools to develop ActiveX and Internet controls.

The Visual Basic Enterprise Edition is the most advanced edition and is aimed at programmers who build distributed applications in a team environment. It includes all the features of the Professional Edition, plus tools such Visual SourceSafe.

VB6 has a long list of minor changes, ranging from new language features to a re-vamping of the Setup Wizard (now called the Package and Deployment Wizard).

2.2 New Features of Visual Basic 6.0

Whenever a product's version number increases by one, it means that several enhancements have been made over the previous version. Before looking at the completely new additions to Visual Basic 6.0, this section presents general Visual Basic features briefly.

General Features

The compiler in Visual Basic gives many different options for optimizing the compiled code, such as Optimization for Fast Code, Optimization for Small Code and Favor of Pentium Pro etc.

The Visual Basic is a very open environment that supports the Client/Server architecture, ActiveX, Component Object Model (COM), Distributed Component Object Model (DCOM). It also supports Open Database Connectivity (ODBC).

IntelliSense

A new feature that Microsoft calls IntelliSense enables the system to react in real time during coding. There are five basic IntelliSense features:

- ❖ QuickInfo
- ❖ Complete Word
- ❖ Data Tips
- ❖ List members
- ❖ List Constants

QuickInfo is a feature that presents the syntax of the procedure, which is being typed, in a ToolTip like window. It even goes one step further in that it will bold the specific parameter which is being coded in real time.

Complete Word is a feature that automatically completes the word which is typed in real time, after enough characters that make the word distinct within the list of available words, has been typed.

Datatips is a very useful feature that simply shows the value of the variable in a yellow tooltip format at run time.

List Members is a feature that is used to simply list all of the properties or methods available to a given object.

List Constants is a feature that works the same as the List Members feature, except that with this feature available constant values for a given property are listed.

Other features

Multiple projects can be loaded into the IDE - Integrated Development Environment (explained shortly) and treated as one. For example, a standard project and an ActiveX project can be loaded at the same time. This saves tremendous amount of time for coding and debugging. Here's a list of some other enhancements:

- ❖ Break points can be toggled simply with a mouse click.
- ❖ Bookmarks can be placed for quick location.
- ❖ Properties in the property window can be manipulated alphabetically or even by category.
- ❖ A nice new feature is block comment and uncomment. With the click of a mouse button, all highlighted code will be commented or uncommented.
- ❖ Aligning and formatting can easily be applied to one or more controls.

Visual Basic 6.0 features

This section briefs what are newly added in Visual Basic 6.0 in various categories.

New in Data Access

ADO (ActiveX Data Objects)

It combines and supersedes the prior version's DAO and RDO technologies. ADO can be used to connect to a database on a local desktop or to a remote database server. Also, ADO allows to access more types of data - e-mail, for example.

Data Environment

The Data Environment designer provides an interactive, design-time environment for creating ADO objects. These can be used as a data source for data-aware objects on a form or report.

Data Report

It allows to use drag and drop to quickly create reports from any recordset, including hierarchical recordset.

Hierarchical FlexGrid Control

It is an updated version of the FlexGrid control that, in addition to supporting all the functionality of the FlexGrid control, can display a hierarchy of ADO Recordsets. Each Recordset returned is displayed as a separate band within the grid and can be formatted independently.

DataList Control, DataCombo Controls

These controls are OLE DB versions of the DBList and DBCombo controls. They also support the new ADO Data control.

New in Internet Features

IIS Applications

This feature enables to write server-side Internet applications that use Visual Basic code to respond to user requests from a browser.

DHTML Applications

DHTML Applications allows to write Visual Basic code to respond to actions on an HTML page, without transferring processing to the server.

New in Controls

ADO Data Control

A new OLEDB-aware data source control that functions much like the intrinsic Data and Remote Data controls, in that it allows to create a database application with minimum code.

Coolbar Control

A container control that can be used to create user-configurable toolbars similar to those in Microsoft Internet Explorer.

DataGrid Control

An OLEDB-aware version of DBGrid, the control allows to quickly build an application to view and edit recordsets. It also supports the new ADO Data control.

DataList Control, DataCombo Controls

These controls are OLE DB versions of the DBList and DBCombo controls. They also support the new ADO Data control.

DateTimePicker Control

Helps to quickly enter the dates and times.

Hierarchical FlexGrid Control

An updated version of the FlexGrid control that, in addition to supporting all the functionality of the FlexGrid control, can display a hierarchy of ADO Recordsets. Each Recordset returned is displayed as a separate band within the grid and can be formatted independently.

ImageCombo Control

Behaves like the standard ComboBox control with one addition: each items in the list can now have images.

MonthView Control

Allows the end user to pick dates and contiguous ranges of dates from a graphic representation of a calendar.

New in Language Features

Visual Basic 6.0 provides number of new functions for working with strings. They are:

Function	Description
Filter	Allows to filter a string function
Format currency	Allows to format a string to currency
Format DateTime	Allows to format a string to time or date
FormatPercent	Allows to format a string as a percent

	Takes a string array and combines its elements into one string
Replace	Replaces substrings within a string
Round	Return a rounded number as specified
StrReverse	Reverses the order of a string
WeekdayByname	Returns the day of the week

2.3 New Object Oriented Features

Visual Basic's object -based programming is a benefit to Windows developers. The built-in Visual Basic objects like forms and controls provide properties that are easily manipulated at design time and as the application runs on the user's computer. Traditional Windows development systems such as Visual C++ require extensive programming to modify even simple properties such as form's caption or back color.

The greatest advantages of using objects is encapsulation, which is the capability to wrap all aspects of functionality and user interface into a single entity. A Visual Basic object enables you abstract complex activities and tasks as a simple, compact object you can use in any Visual Basic or VBA project. An encapsulated object can be much easier to maintain than a traditional module or VBA procedure. Because an object contains all its functionality and appearance as a single entity, there is a single thing to modify or maintain as improvements are made to the program.

Most application include extensive data validation routines. Depending on the type of data being entered by the user, data validation ranges form a single VBA statement to extensive modules containing dozens or hundreds of lines of coed. By using Visual Basic's object oriented programming features, it's possible to wrap all data validation routines into a single object that can be used by setting its properties and invoking its methods.

Custom objects, therefore , provide a simplified interface to complex operations. When properly designed and implemented, custom objects can be used in virtually any compatible VBA programming system, exposing the same properties and methods you work with when incorporating the objects in your Visual Basic projects.

2.4 The Integrated Development Environment

Visual Basic is not just a language. It's an Integrated Development Environment (IDE), which enables to develop, run, test and debug applications. This IDE is

capable of creating different project type to develop applications. In VB 5 IDE was designed as a Single Document Interface. In single Document Interface each window is a free- floating window that is contained within a main window and can move anywhere on the screen as long as Visual Basic is the current application. But in VB 6.0, IDE is in a Multiple Document Interface (MDI) format. In this format, the windows associated with the project will stay within a single container known as parent. Code and form-based windows will stay within the main container form. When Visual Basic is started, it will prompt to select the project type as in Figure 2.1

To start Visual Basic

- ❖ Click the **Start** button in Windows Taskbar, choose Microsoft Visual Basic 6.0 group and click the Microsoft Visual Basic 6.0 item.

-- Or --

Simply double click the Microsoft Visual Basic 6.0 icon on the Desktop.



Figure 2.1 Types of projects available in Visual Basic.

Standard EXE It is a typical project type and used to create standard executable files.

ActiveX EXE, ActiveX DLL These types of projects are used to created ActiveX code components which are OLE automation servers. These two types are of projects are identical in functionality, but are packaged differently (as executable files or Dynamic Link Libraries).

ActiveX Control This type of project is used to create new ActiveX controls.

VB Application Wizard It generates a new, fully functional application from which more complex application can be built.

Data Project It creates a project that automatically includes Data Report and Data Environment.

ActiveX Document EXE, ActiveX Document DLL These project types are used to ActiveX documents, which are in essence Visual Basic applications that can run in the Web browser such as Internet Explorer.

IIS Application, DHTML application These projects are used to create Internet Applications for both Server-side and Client-side.

The New Project dialog box as in Figure 2.1 has three tabs:

- New
- Existing
- Recent

The New tab allows to select the type of a new project, as explained already. Switching to the Existing tab will enable to select an existing project and open it. Switching to the Recent will enable to select and open the most recently worked projects during the last few days.

Select the Standard EXE icon in the New Project window, and click the OK button. This will open the Integrated Development Environment (IDE) as in Figure 2.2

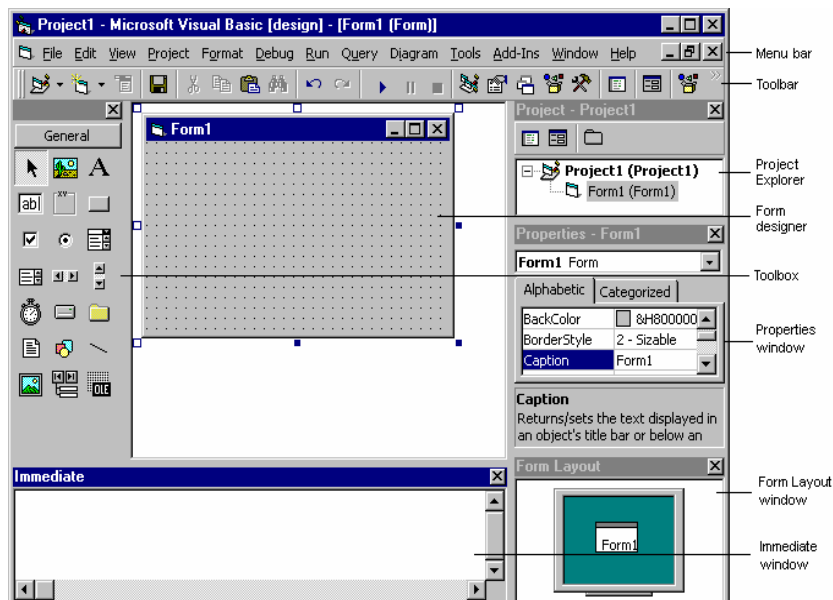


Figure 2.2 Integrated Development Environment.

The following sections briefly explain the parts of the IDE.

The Menu Bar

The menu bar contains the commands needed to work with the Visual Basic. The basic menus are:

- ❖ **File** Contains the commands for opening and saving projects and creating executable files and list of recent projects.
- ❖ **Edit** Contains editing commands (Undo, Copy, Paste, and so on) plus a number of commands for formatting and editing code (Find, Replace).
- ❖ **View** Contains commands for showing or hiding components of the IDE.
- ❖ **Project** Contains commands that add components to the current projects references to Windows objects, and new tools to the Toolbox.
- ❖ **Format** Contains commands for aligning the controls on the Form.
- ❖ **Debug** Contains usual debugging commands.
- ❖ **Run** Contains the commands that start, break, and end execution of the current application.
- ❖ **Tools** Contains tools needed in building ActiveX components and ActiveX controls; contains the command to start the Menu Editor and the options command, which lets to customize the environment.
- ❖ **Add-Ins** Contains add-ins that can be added and removed as needed. By default, only the Visual Data Manager Add-In is installed in this menu. Add-In manager command can be used to add and remove add-ins.
- ❖ **Window** Contains the commands to arrange windows on the screen: the standard window menu of a windows application.
- ❖ **Help** Contains information to help user as he works.

The Toolbars

Provide quick access to commonly used commands in the programming environment. By default, the Standard toolbar is displayed when Visual Basic starts. Additional toolbars for editing, form design, and debugging can be toggled on or off

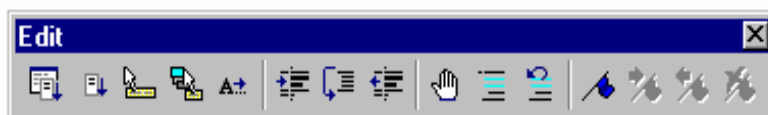
from the Toolbars command on the View menu. For example, to view the Debug Toolbar, choose View | Toolbar and check the Debug option. Likewise to hide the Debug follow the same procedure, but this time uncheck the Debug option.

Toolbars can be docked beneath the menu bar or can "float" if the vertical bar on the left edge is selected and dragged away from the menu bar. Each Toolbar and its brief description are given below.

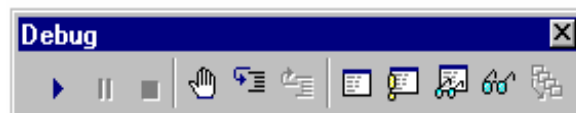
- ⊙ Standard toolbar is just below the menu bar and is displayed by default.



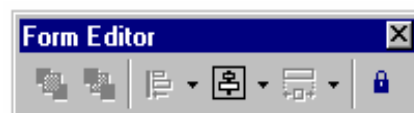
- ⊙ The Edit toolbar contains the commands of the Edit menu.



- ⊙ The Debug toolbar contains the commands of the Debug menu.



- ⊙ The Form Editor menu contains the commands of the Format menu.



The Project Explorer

Lists the forms and modules in current project. A *project* is the collection of files that is used to build an application.

The project components are organized in folders, and the Project Window is called Project Explorer because it resembles the look of the Windows Explorer.

The Toolbox

The Toolbox contains the icons of the controls that can be placed on a form to create the application's user interface. By default, the Toolbox contains the pointer icon and the icons of 20 intrinsic controls.

To place a control, say command button, on the Form

- ❖ Select the command button with the mouse.
- ❖ Move the mouse over the form. When the mouse pointer is over the form it will turn to cross.
- ❖ Draw the control on the form, just as drawing a rectangle.

The Properties Window

The Properties Window contains the property settings for the selected control. Properties are attributes of an object, such as its size, font, background color and so on. By setting these properties the appearance of the selected control can be adjusted easily. For example to change the caption and appearance of a command button follow the steps given below:

- Select the command button icon on the Toolbar and draw the button on the Form. The button will appear with its default name as its caption, as in

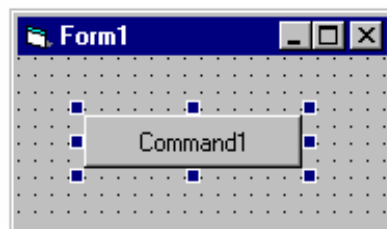


Figure 2.3 Command button with its default name as its caption.

- Make sure that the command button is selected and press F4 or click the Properties window to make it as active.
- In the Properties Window (Figure 2.4) locate the **Caption** property and change its value as **Click &Me**.

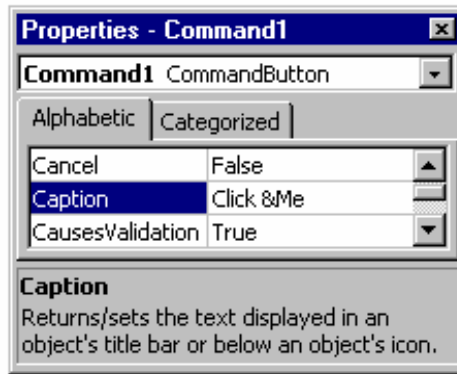


Figure 2.4 New Caption value for the Command button.

- Placing ampersand before the letter **M** will make the caption of the command button to appear as Click Me. It meaning is that this button can be pressed by using keyboard characters Alt and M (Alt+M).

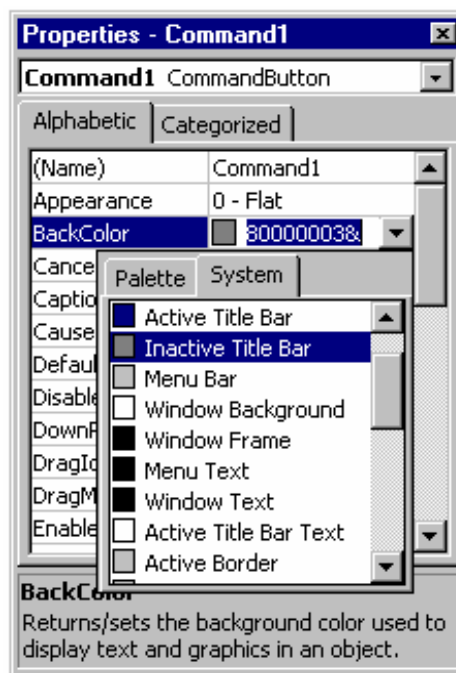


Figure 2.5 Color selection box for the BackColor property.

- To change the background color of the command button, locate the Back color property and click the down arrow button next to the current value of the color. This will display the color selection box as in Figure 2.5
- Select the **Inactive Title Bar** color. Now the command button will appear as in Figure 2.6

Note: When the color selection box is displayed, its Palette tab might be active by default. This tab will show the colors alone. To view each color's description switch to the System tab.



Figure 2.6 New appearance of the Command Button.

The Form Designer

The Form Designer enables to add controls, graphics, and pictures to a Form to define the interfaces in the desired look. Each Form in an application has its own Form designer. The Form Designer displays two windows for each Form.

- ❖ The Form itself (the elements of the user interface)
- ❖ A code window (the code behind the elements of the Form)

Double clicking on a Form will bring its code window front. Double clicking on a Form name in the Project Explorer window will bring the Form to the front. Or to switch between these window easily, the View Code and View Form at the top of the Project Explorer window (Figure 2.7) can be used.

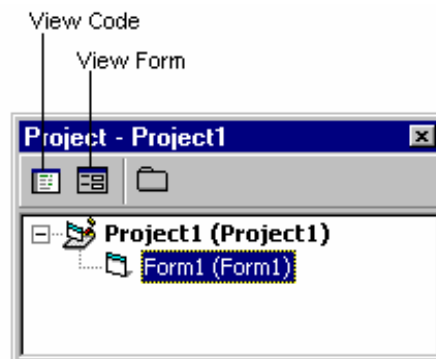
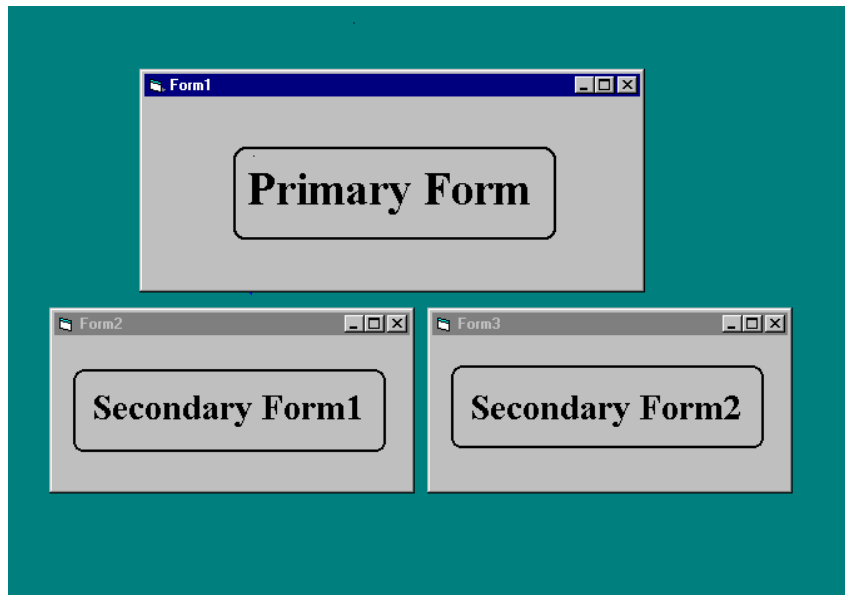


Figure 2.7 Icons at the top of the Project Explorer to switch between Code and Form window.

The Form Layout

The Form Layout window helps to position the forms of an application using a small graphical representation of the screen. This window is very useful in the application that uses multiple forms, because each form's position can easily be specified with respect to the main form. For example, assume that an application contains three forms. The Form Layout window will show the layouts of the all the three forms. Mouse can be used to drag these layouts to the desired location as in Figure 2.8



When the application shows all the Forms, they will appear in the screen as in Figure 2.9

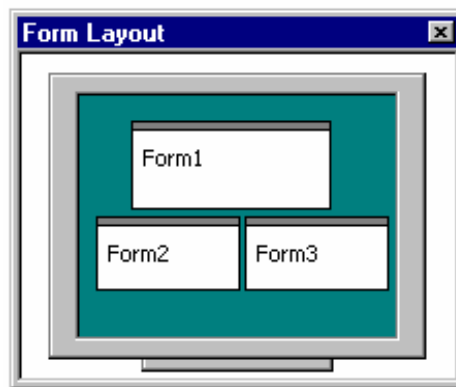


Figure 2.9 Forms – appearing on the Desktop as positioned in the Form Layout window.

The Immediate Window

The Immediate Window is a debugging aid. While an application is running it can be paused and the immediate window can be used to examine or change the values of the application's variable. It can be used to execute simple basic commands even when the application is not running. For example, type **Print 30 + 40** in the Immediate window and hit enter. The result 70 will appear in the next line.

Context Menus

It contains shortcuts to frequently performed actions. When right mouse button is clicked, context menus pops up with list of shortcuts specific to part of the

environment where the mouse is right clicked. For example context menu displays items namely Components..., Add Tab..., Dockable, and Hide, when the mouse is right clicked on the Toolbox (Figure 2.10).



Figure 2.10 Context menu that appears when the right mouse Button is clicked on the Toolbox.

2.4 Basic Scripting In Visual Basic

Microsoft Visual Basic Scripting Edition, the newest member of the Visual Basic family of programming languages, brings active scripting to a wide variety of environments, including Web client scripting in Microsoft Internet Explorer and Web server scripting in Microsoft Internet Information Server. No scripting language including VBScript has disc-accessing features and so it is accepted to be a safe language to deploy in the web.

Easy to Use and Learn

If you already know Visual Basic or Visual Basic for Applications, VBScript will be very familiar. Even if you don't know Visual Basic, once you learn VBScript, you're on your way to programming with the whole family of Visual Basic languages.

It is a free language and so it can be used anywhere. We can write our VBSCRIPT program in notepad and execute it using Internet Explorer. You can use the SCRIPT element to add VBScript code to an HTML page.

The <SCRIPT> Tag

Any scripting language can be enclosed within a script tag in an ordinary HTML file. The script tag can be placed anywhere in the HTML page and it is better to place it in the head tag. We should mention the language as an attribute. It is a container tag.


```
<script language = vbscript>
```

```
</script>
```

2.5 Short Summary

- ☞ Visual Basic is an ideal programming language for developing sophisticated professional application
- ☞ Visual Basic make use of Graphical User Interface for creating robust and powerful application
- ☞ Encapsulation is the combination of data and characteristics in a single package
- ☞ IDE describes the Interface and environment that we use to create our application
- ☞ VBScript has disc-accessing features and so it is accepted to be a safe language

2.6 Brain Storm

1. Is Visual Basic programming language unique to Visual Basic? If yes, in which other application it is used?
2. Differentiate between the previous version and the new version of Visual Basic
3. What is Visual programming?
4. Give some new features of object Oriented Programming?
5. What is IDE? Discuss about the new features of IDE?
6. Why Visual Basic claim as a Free language?

Lab Unit - 2 (2 Real Time Hrs)

1. Create a new Standard EXE project
2. Enter the Code in the Form Load event procedure
3. During run time the caption, mouse pointer, Window State, Height and Width properties should be changed.

Lecture - 3

Program Constructs

Objectives

In this chapter you will learn the following

- ❖ About Variable
- ❖ About constant
- ❖ Know about statements
- ❖ Understand predefined dialog box's

Lecture unit - 3

- 3.1 Snap Shot
- 3.2 Variables
- 3.3 Constants
- 3.4 Operators
- 3.5 User defined data types
- 3.6 Arrays
- 3.7 Rem statement
- 3.8 End statement
- 3.9 Functions
- 3.10 Input box
- 3.11 Msg box
- 3.12 Control statements
- 3.13 Short summary
- 3.14 Brain storm
- 3.15 Lab unit

Lab unit 3 (2 Real Time Hrs)

3.1 Snap Shot

Visual Basic is only a basic element of the application. The code for this interface must be written to define the application's behavior. As with any modern programming language, Visual Basic supports a number of common programming constructs and language elements. This chapter introduces the programming fundamental of Visual Basic.

3.2 Variables

In Visual Basic, as in any other programming language, variables store values during a program's execution. A variable is a named storage location that can contain a value, which can be modified during the program execution or if the value of a specified memory location changes at run time then it is defined to be a variable. The name with which we assign or manipulate the memory location at runtime is called a variable name. Each variable has a name (which uniquely identifies it), value and type. For example, the variable **Name** can have the value "Ram", and the variable **Age** can have the value 24. Here **Name** and **Age** are variable names and "Ram" and 24 are their values, and String and Integer are types of variables **Name** and **Age** respectively.

Declaring Variables

In most programming languages, variables must be declared. In other words, compiler must be told in advance, about the variables to be used. If the compiler knows the variables and their types, it can produce optimized code. When the compiler is informed that the variable Age will hold a number, then it will set required bytes in memory to make ready to use it.

Note: The Compiler is a translator that translates the programming code into the machine's understandable form in order to make it ready for execution.

A variable name:

- ❖ Must begin with a letter.
- ❖ Can't contain an embedded period or embedded type-declaration character.
- ❖ Must not exceed 255 characters.
- ❖ Must be unique within the same *scope*.

Explicit Declarations

To declare variables and to allocate storage space the Dim statement is used as in the following:

```
Dim meters as Integer
Dim greetings as String
```

The first variable, **meters** will store an integer value such as 10 or 388, and the second variable, **greetings**, will store a text such as "Happy BirthDay". When the compiler finds a Dim statement, it creates placeholder by reserving some space in memory and assigning a name to it. Each time this name is used in subsequent statements, Visual basic uses this area in memory to read or set its value. For instance, when the following statement is used:

```
meters = 143
```

Visual Basic places the value 143 in the placeholder reserved for the variable **meters**. When the program asks for the value of this variable, Visual Basic reads it from the same area of memory. The following statement:

```
Debug.Print meters
```

Causes Visual Basic to retrieve the value from the area of memory named **meters** and prints it the immediate window Form.

Note: The Print method with Debug prefix sends output to the Immediate window. Using the Print method in this way is helpful to examine the values of the variables in much easier way.

A single statement can both read and set the value of a variable. The following statement increases the value of the **meters** variable:

```
meters = meters + 1
```

Visual Basic reads the value (say 143), adds 1 to it, and then stores the new value (144) in the same memory location.

Attempting to assign a value of the wrong type to a declared variable generates a "TypeMismatch " run time error. For example, if an attempt is made to assign the value "Welcome" to the variable **meters** which is of integer type, Visual Basic would not execute the statement, as it violates the variable declaration.

Implicit Declarations

Variables can also be used without declaring them. When Visual Basic meets an undeclared variable name, it creates a new variable on the spot and uses it. The new variable's type is **Variant**, the generic data type that can accommodate all the data types. Visual Basic adjusts the undeclared variable's type according to the value assigned to it. For example when an undeclared variable is used as follows:

```
var1 = 1000
```

Visual Basic creates a variable named var1 of Variant type and adjusts its type to Integer and an integer value (in this case 1000) is assigned to it.

Types of Variables

The type of a variable determines how to allocate storage space in the computer's memory to their values. All variables have a type that determines what kind of data they can store.

Visual Basic recognizes the following types of variables:

- ❖ Numeric
- ❖ String
- ❖ Variant
- ❖ Date
- ❖ Object

The Numeric Data Types

Visual Basic supplies several numeric data types – Integer, Long (long integer), Single (single-precision floating point), Double (double-precision floating point), and Currency. Using a numeric data type generally uses less storage space than a variant. If it is known that a variable will store whole numbers (such as 12,388,62, etc.) rather than numbers with a fractional amount (such as 3.57, 0.7876, etc.) then it is better to declare it as an Integer or Long type variable. Operations are faster with integers, and these types consume less memory than other data types.

Different types of variables can be declared in a single Dim statement. For example, the following declaration statements

```
Dim var1 as Single  
Dim var2 as Double
```

```
Dim var3 as Currency
```

Can be minimized as in the following:

```
Dim var1 as Single, var2 as Double, var3 as Currency.
```

Different types of numbers are represented internally in different formats. All numeric values are truncated to a certain extent. The result of the operation $1/3$ is 0.333333... (an infinite number of digits 3). It will fill the total capacity of RAM with the digit 3, and the result will still be truncated. Here's a simple but illuminating example.

```
Dim a as Single, b as Double
```

Single and **Double** are two basic data types for storing *floating-point numbers* (numbers that have a fractional part), and the Double type can represent these numbers more accurately than the Single type. The following statements

```
a = 1 / 3  
Debug.Print a
```

Will produce the result in the immediate window:

```
0.3333333
```

And if the following statements are executed subsequently:

```
a = a * 100000
```

```
The output will be  
33333.34
```

The above result is not accurate, as it is not rounded properly. If this result is divided by 100000, its result will be 0.3333334, which is different from the initial result (0.3333333). This is an important point in numeric calculations, and it is called error propagation. In long sequences of numeric calculations, errors propagate. If the above same operation is performed with the variable b, which is declared as Double as in the following:

```
b = 1 / 3  
Debug.Print b  
B = b * 100000  
Debug.Print b
```

The result will be

```
0.3333333333333333
```

33333.3333333333

which is more accurate.

The choice of data types for variables can make a difference in the results of the calculations. The proper variable types are determined by the nature of the values they present. The choice of data types is frequently a tradeoff between precision and speed of execution (less precise data types are manipulated faster). The following table shows range and storage space allocated for each type of numeric variable.

Data type	Storage size	Range
Integer	2 bytes	-32,768 to 32,767
Long (long integer)	4 bytes	-2,147,483,648 to 2,147,483,647
Single (single-precision floating-point)	4 bytes	-3.402823E38 to -1.401298E-45 for negative values; 1.401298E-45 to 3.402823E38 for positive values
Double (double-precision floating-point)	8 bytes	-1.79769313486232E308 to -4.94065645841247E-324 for negative values; 4.94065645841247E-324 to 1.79769313486232E308 for positive values
Currency (scaled integer)	8 bytes	-922,337,203,685,477.5808 to 922,337,203,685,477.5807

Table 3.1 Numeric data types.

The Byte Data Type

The Byte type holds basically an integer in the range 0 to 225 and they are declared with the following statement.

```
Dim varname as Byte
```

As the name implies bytes allocated in the memory for Byte data type is one. Bytes variables are frequently used to access binary files, image and sound files, and so on. All operators that work on integers work with the Byte data type except unary minus. Since Byte is an unsigned type with the range 0-255, it cannot represent a negative number.

String Variables

The String data type stores only text, and string variables are declared with the following statement:

```
Dim varname as string
```

By default, a string variable is a *variable-length string*; the string grows or shrinks as new data is assigned to it. String variables can also be declared with fixed length as in below:

Syntax

```
Dim String * size
```

For example, to declare a string that is always 50 characters long, the following statement is used:

```
Dim EmpName As String * 50
```

If a string of fewer than 50 characters is assigned, the EmpName is padded with enough trailing spaces to total 50 characters. If a string that is too long for the fixed-length string is assigned, Visual Basic simply truncates the characters.

If a string variable's size will change drastically during the course of an application, it is better to declare it as fixed-length to prevent Visual Basic from having to resize it constantly, which slows down the execution speed.

The Boolean Data Type

The Boolean data type stores True/False values. Although a single bit would be adequate, for efficiency reasons Visual Basic allocates two bytes to this data type. Boolean variables are in essence integers that take the value -1(for True) and 0 (for False). Boolean variables are declared as

```
Dim Running as Boolean
```

These variables are combined with the logical operators AND, OR and NOT. The NOT operator toggles the value of a Boolean variable. The following statement:

```
Running = NOT Running
```

is a toggle. If the variable Running is True, it's reset to False, and vice versa. This statement is a shorter way of coding the following:

```
If Running = True Then
```

```

        running = False
Else
        Running = True
End If

```

The Date Variable

Date and time values are stored internally as in a special way. A variable declared as Date:

```
Dim expiration as Date
```

Can store both date and time values. The following are all valid assignments:

```

expiration = "01/01/1999"
expiration = "01/23/1999"
expiration = "13:03:05 AM"
expiration = "02/23/1999 13:03:05 AM"

```

Two date variables can be subtracted know the difference as in the following:

```

Dim Date1 as date, Date2 as date;
Dim days as integer;
Date1 = "01/02/1999"
Date2 = "01/08/1999"
Days = Date2 - Date1

```

In the above example the integer variable Days will hold the value 6 which is the difference between the dates Date1 and Date2. The value are stored in the date variables in mm/dd/yyyy (mm - Month, dd - Day, yyyy - Year) format.

Integer values can be added to a Date variable to add days. For example, if Date1 holds the date "01/01/1999" the following line

```
Debug.Print Date1 + 10
```

Will print the value "01/11/1999"

Note: To add an hour to a Date variable, add 1/24 of a day; to add a minute add 1/(24*60) of a day.

Object Variables

Object variables are stored as 32-bit (4-byte) addresses that refer to objects within an application or within some other application. A variable declared as Object is one that can subsequently be assigned (using the Set statement) to refer to any actual object recognized by the application.

For example, it has been assumed that a Form in a Visual Basic application has two command buttons namely Command1 and Command2. And two object variables are declared as follows:

```
Dim a as CommandButton, b as CommandButton;
```

Each of these two Object variables can be set to one of the two command buttons with the following Statements:

```
Set a = Command1
```

```
Set b = Command2
```

From now on the command buttons' properties can be manipulated through the variables a and b. To change the Caption property of the first command button, the following statement is used:

```
a.Caption = "Click Me"
```

To turn on the bold attribute of the second command button (so that its caption appears in bold), the following statement is used:

```
b.FontBold = True
```

This section just briefed about object variables. Objects and Object variables are explained in detail in the chapter "Object Programming in Visual Basic"

The Variant Variable

A Variant variable is capable of storing all system-defined types of data. Variant variables are declared without specifying a type as follows:

```
Dim var
```

Or for cleaner code it can be declared as Dim var as Variant. Variant variable can be used in both numeric and string calculations. There is no need to perform any type conversions. Visual Basic automatically performs any necessary conversion. For

Example

```
Dim SomeValue                ' Variant by default.  
  
eValue = "17"                ' SomeValue contains "17" (a  
  
                              'two-character string)
```

SomeValue = SomeValue-15 ' SomeValue now contains the
' numeric value 2.

SomeValue = "U" & SomeValue ' SomeValue now contains
' "U2" (a two- character
' string).

Converting Variable Types

In some situation variables has to be converted from one type to another. For example, an automatically generated number may have to be concatenated with a prefix to produce an another string say Employee Id (something like AAA1001). Concatenation can take place only on strings. So, first the number has to be converted into a string, and then it can be concatenated with the prefix. The CStr function converts an expression into a sting. The following example explains this process.

```
Dim EmployeeId as String
Dim AutoNum as Integer
```

AutoNum = *(some expression that automatically generates a number)*

EmployeeId = "AAA" + CStr(AutoNum)

The following are the list of type conversion functions.

<i>Conversion Function</i>	<i>Converts an expression to</i>
CBool	Boolean
CByte	Byte
CCur	Currency
CDate	Date
CDbl	Double
CInt	Integer
CLng	Long
CSng	Single
CStr	String
CVar	Variant

Special Values

Generally variables have value. Before any value is assigned to them, numeric variables are zero, and string variables are zero length (""). There are, however, two special values: Empty and Null.

The Empty Value

If a variant variable is declared but not initialized then its value is Empty. The Empty value is different from a zero-length string. The IsEmpty() function can be used to find out whether a variant variable, say someVariable, is initialized or not as in the following:

```
If IsEmpty(someVariable) Then Debug.Print "Variable is not initialised"
```

The IsEmpty function will return a Boolean value. That is, it will return True, if the variable is not initialized or False, if the variable is initialized.

The Null Value

Null is commonly used in database applications to indicate that a field doesn't contain data or Object variable hasn't been assigned a value. Uninitialized variables that refer to objects (databases, custom objects) are Null, not Empty as in the following.

```
If Not IsNull(someVariable) then  
    (Process the variable someVariable)  
End if
```

Examining the Type of a Variable

Besides setting the types of the various variables and converting between types, the type of a variable can be examined by using the VarType and TypeName functions. For example, a variable might be declared of variant type. But it will be converted to a specific type automatically by Visual Basic according to its current value. In this sense to know the new type either VarType or TypeName functions can be used. Both functions accept as argument the name of a variable, and they return a number (the VarType() function) or a string (the TypeName() function) indicating the type of the variable.

The VarType Function

The VarType() function returns one of the numbers shown in Table3.2, depending on the type of the argument.

Constant	Value	Description
vbEmpty	0	Empty (uninitialized)
vbNull	1	Null (no valid data)
vbInteger	2	Integer
vbLong	3	Long integer
vbSingle	4	Single-precision floating-point number
vbDouble	5	Double-precision floating-point number
vbCurrency	6	Currency value
vbDate	7	Date value
vbString	8	String
vbObject	9	Object
vbError	10	Error value
vbBoolean	11	Boolean value
vbVariant	12	Variant (used only with arrays of variants)
vbDataObject	13	A data access object
vbDecimal	14	Decimal value
VbByte	17	Byte value
VbUserDefined Type	36	Variants that contain user-defined types
VbArray	8192	Array

Table 3.2 The Numbers that the VarType() returns.

Example

This example uses the **VarType** function to determine the type of a variable.

```
Dim IntVar, StrVar, DateVar, MyCheck
```

```
' Initialize variables.
```

```
IntVar = 459: StrVar = "Hello World": DateVar = #2/12/69#
```

```
MyCheck = VarType(IntVar)           ' Returns 2.
```

```
MyCheck = VarType(DateVar)         ' Returns 7.
```

```
MyCheck = VarType(StrVar)          ' Returns 8.
```

Note: The single quote (') in the above statements indicates the beginning of comment, which will be ignored by Visual Basic.

While checking the type of the variables, it is hard to remember the return value that corresponds to a specific type. So instead of numeric values constants (listed in Table 3.2) can be used as in the following:

```
If VarType(IntVar) = vbInteger then
    Debug.Print "Variable is of Integer Type"
End If
```

The TypeName function

This function returns the type of a variable as a string as the following example shows:

```
' Declare variables.
Dim MyType, StrVar As String, IntVar As Integer
Dim CurVar As Currency
Dim ArrayVar (1 To 5) As Integer

MyType = TypeName(StrVar) ' Returns "String".
MyType = TypeName(IntVar) ' Returns "Integer".
MyType = TypeName(CurVar) ' Returns "Currency".
MyType = TypeName(ArrayVar) ' Returns "Integer()".
```

Is it a Number or an Array or a Date?

There is another set of functions that returns variables' data types, but not the exact type. They are:

- ❖ **IsNumeric()** Returns True if its argument is a number (Integer, Long, Currency, Single, or Double).
- ❖ **IsDate()** Returns True if its argument is a valid date or time.
- ❖ **IsArray()** Returns True if its argument is an array.

Forcing Variable Declarations

Visual Basic doesn't enforce variable declaration. However, declaring all variables will help to write clean code and will simplify debugging. So, to enforce Visual Basic for variable declarations, the following statement has to be placed in the declaration section of a Form or Module.

```
Option Explicit
```

This statement tells the compiler to check each variable and issue an error message if it is used without declaration. If this statement is omitted, then Visual Basic creates variables as needed.

Another reason for declaring variables in advance is to simplify debugging, especially in large applications. Suppose the variable DM2USD appear in many places in the application. If in one of these places the name is typed DM2UDS instead of DM2USD and the program doesn't enforce variable declaration, the compiler creates a new variable, assigns it the value zero, and then uses it. If the application enforces variable declaration, the compiler will complain (the DM2UDS is not declared) and this enable to catch the error.

If the **Require Variable Declaration** check box is checked in the Options dialog box's Editor tab (Figure 3.1), which appears when Tools | Option is chosen, Visual Basic automatically inserts the Option Explicit statement in any new modules, but not in existing modules. To enforce the variable declaration in the existing module Option Explicit statement must manually be typed.

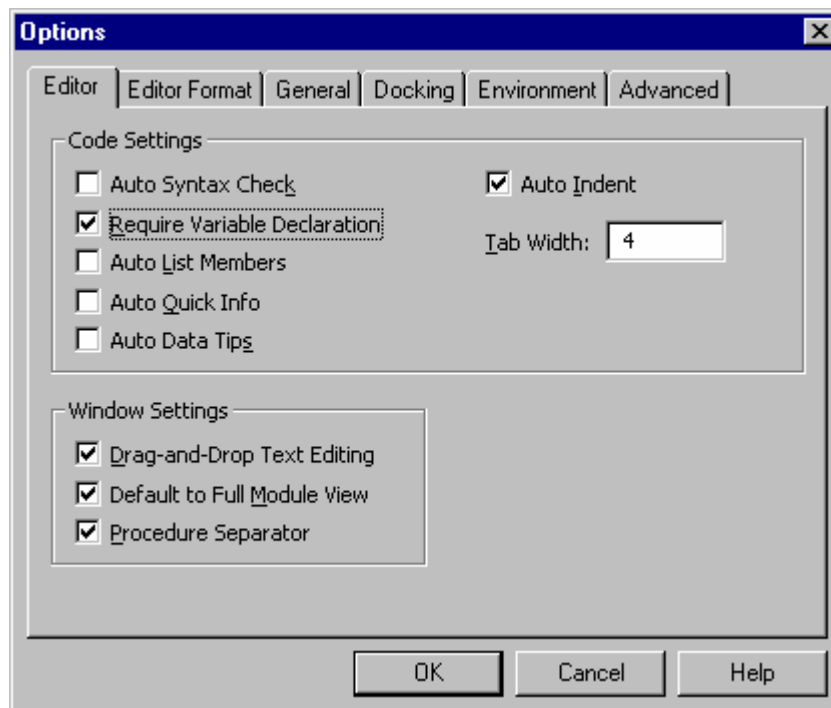


Figure 3.1 Editor tab of Options dialog box in which Require Variable Declaration checkbox is checked.

A variable's scope

If a variable is declared within a procedure, only the code in the specific procedure

has access to that variable. This variable doesn't exist for the rest of the application. This variable's scope is limited to a procedure, and it's called local.

Example

```
Sub Command1_Click()  
    Dim i as Integer, j as Integer  
  
    -----  
    -----  
  
End Sub
```

The variables *i* and *j* are declared inside the `Command1_Click()` procedure and so, they are local variables and accessible only within this procedure. If a variable has to be accessed by all procedures then it has to be declared in the declaration section.

Finally, in some situations the entire application must access a certain variable. In this case, the variable must be declared as `Public`. Public variables have a global scope (they are visible to any part of variable). To declare a Public variable, `Public` statement is used instead of `Dim` statement as follows:

```
Public a as Integer
```

Moreover, Public variables may not appear inside procedures. They must be declared in a standard module.

The Lifetime of a Variable

In addition to type and scope, variables have a lifetime, the period during which they retain their value. Variables declared as `Public` exist for the lifetime of the application. Local variables, declared within procedures with the `Dim` or `Private` statement, live for as long as the procedure in which they are declared. When the procedure finishes, the local variables cease to exist, and the allocated memory is returned to the system. The same procedure can be called again, of course. In this case, the local variables are recreated and initialized again.

However, a local variable can be forced to preserve its value, with the `Static` keyword. `Static` keyword is used to declare variables inside a procedure, exactly in the same way as `Dim` statement as in the following:

```
Static Depth
```

For example, the following function calculates a running total by adding a new value

to the total of previous values stored in the static variable ApplesSold:

```
Function RunningTotal(num)
Static ApplesSold
ApplesSold = ApplesSold + num
RunningTotal = ApplesSold
End Function
```

If ApplesSold was declared with Dim instead of Static, the previous accumulated values would not be preserved across calls to the function, and the function would simply return the same value with which it was called.

The same result can be produced by declaring ApplesSold in the Declarations section of the module, making it a module-level variable. However, the procedure no longer has exclusive access to it. Because other procedures can access and change the value of the variable, the running totals might be unreliable and the code would be more difficult to maintain.

3.3 Constants

Some variables don't change value during the execution of a program. These are constants that appear many times in the program code. For example if a program does math calculations, the value of pi (3.14159....) may appear many times in the program code.

These values are best represented by constants. Instead of typing the value 3.14159 over and over again, a constant can be defined with a constant name, say *pi*, and it can be used in the program code as follows:

$$\text{Area} = \text{pi} * \text{Radius} ^ 2$$

which is much easier to understand than the equivalent:

$$\text{Area} = 3.1415 \text{ Radiius} ^ 2$$

Pi could be declared as a variable, but constants are preferred for two reasons:

1. **Constants can't change value:** This is a safety feature. Once a constant has been declared, its value can not be changed in subsequent statements, it can be assured that the value specified in the constant's declaration will take effect in the rest of

the program.

2. **Constants are processed faster than variables:** When the program is running, the values of constants don't have to be looked up. The compiler substitutes constant names with their values, and the program executes faster.

Often you'll find that your code contains constant values that reappear over and over. Or you may find that the code depends on certain numbers that are difficult to remember – numbers that, in and of themselves, have no obvious meaning.

In these cases, you can greatly improve the readability of your code – and make it easier to maintain – by using constants. A constant is a meaningful name that takes the place of a number or string that does not change. Although a constant somewhat resembles a variable, you can't modify a constant or assign a new value to it as you can to a variable. There are two sources for constants:

- Intrinsic or system-defined constants are provided by applications and controls. Visual Basic constants are listed in the Visual Basic (VB) and Visual Basic for applications (VBA) object libraries in the Object Browser. Other applications that provide object libraries, such as Microsoft Excel and Microsoft Project, also provide a list of constants you can use with their objects, methods, and properties. Constants are also defined in the object library for each ActiveX control.
- Symbolic or user-defined constants are declared using the Const statement.

In Visual Basic, constant names are in a mixed-case format, with a prefix indicating the object library that defines the constant. Constants from the Visual Basic and Visual Basic for applications object libraries are prefaced with "vb" – for instance, vbTileHorizontal.

The prefixes are intended to prevent accidental collisions in cases where constants have identical names and represent different values. Even with prefixes, it's still possible that two object libraries may contain identical constants representing different values. Which constant is referenced in this case depends on which object library has the higher priority.

To be absolutely sure you avoid constant name collisions, you can qualify references to constants with the following syntax:

```
[libname.][modulename.]constname
```

Libname is usually the class name of the control or library. Modulename is the name of the module that defines the constant. Constname is the name of the constant. Each of these elements is defined in the object library, and can be viewed in the Object Browser.

Creating Your Own Constants

The syntax for declaring a constant is:

```
[Public | Private] Const constantname[As type] = expression
```

The argument constantname is a valid symbolic name (the rules are the same as those for creating variable names), and expression is composed of numeric or string constants and operators; however, you can't use function calls in expression.

A Const statement can represent a mathematical or date/time quantity:

```
Const conPi = 3.14159265358979
Public Const conMaxPlanets As Integer = 9
Const conReleaseDate = #1/1/95#
```

The Const statement can also be used to define string constants:

```
Public Const conVersion = "07.10.A"
Const conCodeName = "Enigma"
```

You can place more than one constant declaration on a single line if you separate them with commas:

```
Public Const conPi = 3.14, conMaxPlanets = 9, _
conWorldPop = 6E+09
```

The expression on the right side of the equal sign (=) is often a number or literal string, but it can also be an expression that results in a number or string (although that expression can't contain calls to functions). You can even define constants in terms of previously defined constants:

```
Const conPi2 = conPi * 2
```

Once you define constants, you can place them in your code to make it more readable. For example:

```
Static SolarSystem(1 To conMaxPlanets)
If numPeople > conWorldPop Then Exit Sub
```

Scoping User-Defined Constants

A Const statement has scope like a variable declaration, and the same rules apply:

- To create a constant that exists only within a procedure, declare it within that procedure.
- To create a constant available to all procedures within a module, but not to any code outside that module, declare it in the Declarations section of the module.
- To create a constant available throughout the application, declare the constant in the Declarations section of a standard module, and place the Public keyword before Const. Public constants cannot be declared in a form or class module.

Avoiding Circular References

Because constants can be defined in terms of other constants, you must be careful not to set up a cycle, or circular reference between two or more constants. A cycle occurs when you have two or more public constants, each of which is defined in terms of the other.

For example:

```
' In Module 1:
Public Const conA = conB * 2 ' Available throughout
                             ' application.
```

```
' In Module 2:
Public Const conB = conA / 2 ' Available throughout
                             ' application.
```

If a cycle occurs, Visual Basic generates an error when you attempt to run your application. You cannot run your code until you resolve the circular reference. To avoid creating a cycle, restrict all your public constants to a single module or, at most, a small number of modules.

3.4 Operators

Operators can be classified into three main categories.

1. Arithmetic Operators

2. Relational or Comparison Operators
3. Logical Operators

The table lists the operators available in vbscript and most of them are self-explanatory.

Arithmetic		Comparison		Logical	
Description	Symbol	Description	Symbol	Description	Symbol
Exponentiation	^	Equality	=	Logical Negation	Not
Unary Negation	-	Inequality	<>	Logical Conjunction	And
Multiplication	*	Less than	<	Logical Disjunction	Or
Division	/	Greater than	>	Logical Exclusion	Xor
Integer	\	Less than or equal to	<=	Logical Equivalence	Eqv
Modulus Arithmetic	Mod	Greater than or equal to	>=	Logical Implication	Imp
Addition	+	Object Equivalence	Is		
Subtraction	-				
String Concatenation	&				

Arithmetic operators

<Script language = vbscript>

Dim x, y

X=10

Y=3

E = x ^ y ' find x to the power of y

I = x \ y ' performs integer division

M = x mod y ' returns remainder of the division performed by x/y

</script>

Output of the script

E=1000; I=3; M=1

The usage of relational operators and logical operators will be discussed in the next section.

Logical Operators

Logical Operators follows a set of rules, which can be used with comparison statements.

The rule followed by the operator is defined through truth table.

Truth Table for And Operator

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Truth Table for Or Operator

		Y
		0
		1
		1
		1

Truth Table for Not Operator

Where A, B refers input

Y refers output

0 considered False

1 considered True

3.5 The User Defined Data Types

Most programs store sets of data of different types. For example, a program for maintaining a company's system information must store several pieces of information for each system; the system brand, its cost, and its purchase date etc. All these information should be stored together. To achieve this user defined data types are used. They are useful to create a single variable that records several related

pieces of information.

User-defined data types are created with the Type statement, which must be placed in the Declarations section of a module.

Syntax

```
Type varType
    variable1 as varType
    variable2 as varType
    -----
    -----
End Type
```

After this declaration, variables can be created of this type and manipulated as all other variable (with a little extra typing).

Example

This example shows how to create a user defined data type SystemInfo that holds the System Information.

```
Type SystemInfo
    Brand   as String
    Cost           as Currency
    PurchaseDate as Date
End Type
```

From now onwards variables can be declared of the SystemInfo type in the same way as regular variables are declared. To define variables of this type, the following statement is used.

```
Dim system1 as SystemInfo, system2 as SystemInfo
```

To assign value to these variable, values to each one of its components (called as fields) must separately be assigned, which can be accessed by combining the name of the variable and the name of a field separated by a period, as in the following:

```
system1.Brand           = "HCL system"
system1.Cost            = 30000
system1.PurchaseDate    = "01/01/1999"
```

3.6 Arrays

A standard structure for storing data in any programming language is an array. Whereas individual variables can hold single entities, such as a number, a date, or a string, arrays can hold sets of related data. An array has a name, as does a variable, and the values stored in it can be accessed by an index.

Suppose if salary for 10 employees is to be stored, then variables Salary1, Salary2, ...Salary10 could be created to hold all 10 employees' salary. But if number of employee's goes to very large extent then it is impossible to create a variable for each employee. This way is not elegant too. Arrays are useful here, as they allow to refer to a series of variables by the same name and to use a number (an index) to tell them apart. Arrays are declared as in the following:

```
Dim Salary(99) as Integer
```

This declaration creates an array named Salary with 100 elements, with index numbers running from 0 to 99 and holds 100 values; the salaries of 100 employees. Salary (0) is the first person's salary, Salary (1) the second person's salary, and so on. Values are assigned to the elements of arrays as follows:

```
Salary(0) = 1000
```

All elements in an array have the same data type. Of course, when the data type is Variant, the individual elements can contain different kinds of data. By default, the first element of an array has index 0 (lower - bound) and the number that appears in parentheses in the Dim statement is the array's upper limit (upper - bound). However, the array's first element need not be zero. Lower limit (lower - bound) can explicitly be specified as in the following:

```
Dim Salary(1 to 100) as Integer
```

The lower bound can have any other value, provided it is smaller than the upper bound. The following declarations are valid:

```
Dim Greetings(10 to 20) as String
```

```
Dim Amounts (100 to 101) as Double
```

Suppose if the array Salary contains salaries of all the employees, one thing that is to be remembered is which person corresponds to each salary. This can be avoided by declaring an another array Names as in the following:

```
Dim Names(1 to 100) as String
```

Now values to the elements of both arrays can be assigned as in the following:

```
Names(0) = "Ekambaram"
```

```
Salary(0) = 20000
```

```
Names(1) = "Prema"
```

```
Salary(1) = 15000
```

```
Names(2) = "Gayathri"
```

```
Salary(2) = 50000
```

```
-----  
-----
```

Note: If the index value that is used to access a element of an array, is not between the array's lower limit and upper limit, Visual Basic will generate an error "Subscript out of range" at run time.

Multi-Dimentional Arrays

Sometimes when storing values in an array, its related information has also to be stored. For example, to keep track of each pixel on a computer screen, both X and Y coordinates are to be referred. This can be done using a multidimensional array to store the values.

Arrays of multiple dimentions can be declared as follows:

```
Dim Matrix(9,9) as Double
```

The above statement declares a two-dimensional 10-by-10 array. But arrays can be extended for more dimensions as in the following:

```
Dim MultiD(3, 1 To 10, 1 To 15)
```

This declaration creates an array that has three dimensions with sizes 4 by 10 by 15. The total number of elements is the product of these three dimensions, or 600.

The benefit of using multidimensional arrays is that they are conceptually easier to manage. The following example demonstrates this.

Example

In a game board, positions of certain pieces are to be tracked. Each square on the board is identified by two numbers, its horizontal and vertical coordinates. The obvious structure for tracking the board's squares is a two-dimensional array, in which first index corresponds to the row number and the second index corresponds to the column number. The array could be declared as follows:

Dim Board(1 to 10,1 to 10) as Integer

When a piece is moved from the square on the first row and first column to the square on the third row and fifth column, 0 is assigned to the element that corresponds to the initial position:

Board(1,1) = 0

And 1 is assigned to the square to which is moved:

Board(3,5) = 1

To find out whether a piece on the bottom right square the following statements are used:

If Board(10,10) = 1 then

 { Piece found }

Else

 { Empty Square }

End if

Dynamic Arrays

Sometimes when declaring arrays its required size may not be known. Instead of making it large enough to hold the maximum number of data (which means the most of the array may be empty), arrays can be declared dynamically.

A dynamic array can be resized at any time. Dynamic arrays are among the most flexible and convenient features in Visual Basic, and they help to manage memory efficiently. For example, a large array can be used for a short time and then can be -- freed up memory to the system when that array is no longer used.

Dynamic arrays are created in the same way as normal arrays, but without specifying it dimensions as in the following:

Dim DynArray()

Later in the program, when the number of elements to be stored in the array is known, the Redim statement is used to redimension the array with the required size as follows:

ReDim DynArray(*count*) ' count is probably a user-

' entered value.

The ReDim statement can appear only in a procedure. Unlike the Dim statement, Redim is executable and so, can appear only inside a procedure. An array can be redimensioned to multiple dimensions as follows:

```
Dim Matrix() as Double
```

```
ReDim Matrix(9,9,9)
```

Note: The ReDim statement can't change the type of the array. Moreover, subsequent ReDim statements can change the bounds of a array and number of its dimensions.

The Preserve Keyword

Whenever the ReDim statement is executed, all the values currently stored in the array are lost. Visual Basic resets the values to the Empty value (for Variant arrays), to zero (for numeric arrays), to a zero-length string (for string arrays), or to Nothing (for arrays of objects). The Preserve Keyword is used to resize an array without losing its contents. For example if an array is having 15 elements and have to be resized to hold 25 elements, without losing its contents, then the preserve keyword is used at redimension as in the following:

```
ReDim Preserve DynArray(14+10)
```

3.7 REM Statement

Rem comments

Used to include explanatory remarks in a program. Allows comments to be added to a program. Everything on the line after the Rem statement is ignored by Visual Basic. A apostrophe(`) can also be used in lieu of the Rem statement

Syntax

```
Rem comment
```

or

```
' comment
```

The comment argument is the text of any comment you want to include. After the Rem keyword, a space is required before comment.

Note : If you use line numbers or line labels, you can branch from a GoTo or GoSub statement to a line containing a Rem statement. Execution continues with the first executable statement following the Rem statement. If the Rem keyword follows other statements on a line, it must be separated from the statements by a colon (:).

You can use an apostrophe (') instead of the Rem keyword. When you use an apostrophe, the colon is not required after other statements.

The following example illustrates the use of the Rem statement:

```
Dim MyStr1, MyStr2
MyStr1 = "Hello" : Rem Comment after a statement separated by a colon.
MyStr2 = "Goodbye" ' This is also a comment; no colon is needed.
Rem Comment on a line with no code; no colon is needed.
```

3.8 End Statement

Ends a procedure or block.

Syntax

End

End Function

End If

End Property

End Select

End Sub

End Type

End With

The End statement syntax has these forms:

<i>Statement</i>	<i>Description</i>
End	Terminates execution immediately. Never required by itself but may be placed anywhere in a procedure to end code execution, close files opened with the Open statement and to clear variables.
End Function	Required to end a Function statement.
End If	Required to end a block If...Then...Else statement.

End Property	Required to end a Property Let, Property Get, or Property Set procedure.
End Select	Required to end a Select Case statement.
End Sub	Required to end a Sub statement.
End Type	Required to end a user-defined type definition (Type statement).
End With	Required to end a With statement.

When executed, the End statement resets all module-level variables and all static local variables in all modules. To preserve the value of these variables, use the Stop statement instead. You can then resume execution while preserving the value of those variables.

Note The End statement stops code execution abruptly, without invoking the Unload, QueryUnload, or Terminate event, or any other Visual Basic code. Code you have placed in the Unload, QueryUnload, and Terminate events of forms and class modules is not executed. Objects created from class modules are destroyed, files opened using the Open statement are closed, and memory used by your program is freed. Object references held by other programs are invalidated.

The End statement provides a way to force your program to halt. For normal termination of a Visual Basic program, you should unload all forms. Your program closes as soon as there are no other programs holding references to objects created from your public class modules and no code executing.

3.9 Functions

A function is similar to a subroutine, but a function returns a result. Subroutines perform a task and don't report anything to the calling program; functions commonly carry out calculations and report the result. The statements making up a function are placed in a pair of Function / End Function statements. Moreover, because a function reports a result, it must have a type, as in the following:

```
Function NextDay() As Date
    NextDay = Date() + 1
End Function
```

The NextDay() function returns tomorrow's date by adding 1 day to the current date. Because it must report the result to the calling program, the NextDay() function has a type, as do variables, and the result is assigned to its name (Which can't be done with subroutines).

The Exit Function statement causes an immediate exit from a Function procedure, as

the Exit Sub statement does in the subroutine. Program execution continues with the statement following the statement that called the function.

3.10 Input Box Function

Displays a prompt in a dialog box, waits for the user to input text or click a button, and returns the content entered by the user.

Syntax

InputBox (*prompt* [, *title*] [, *default*] [, *xpos*] [, *ypos*])

The arguments are described in the following table 1.3.

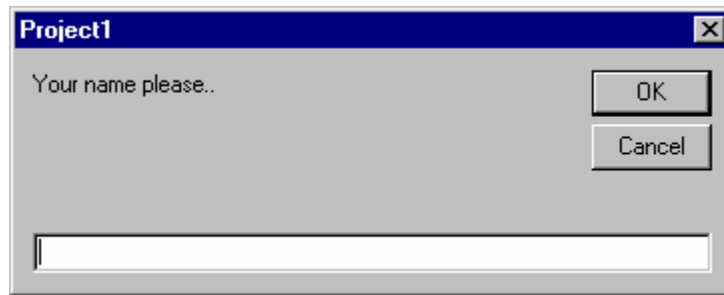
Argument	Description
Prompt	Required. String expression displayed as the message in the dialog box.
Title	Optional. String expression displayed in the title bar of the dialog box. If omitted, the application name is placed in the title bar.
Default	Optional. String expression displayed in the text box as the default response if no other input is provided. If omitted, the text box is displayed empty.
Xpos	Optional. Numeric expression that specifies, in twips, the horizontal distance of the left edge of the dialog box from the left edge of the screen. If omitted, the dialog box is horizontally centered.
Ypos	Optional. Numeric expression that specifies, in twips, the vertical distance of the upper edge of the dialog box from the top of the screen. If is omitted, the dialog box is vertically positioned approximately one-third of the way down the screen.

Table 3.3 Arguments of InputBox function.

Example 1

Dim Name as String

```
' Specifying prompt alone
Name = InputBox("Your name please..")
```



The above statement will display the Input box as in Figure 1.13

Figure 3.2 Inputbox with Project's title and without default value.

When the user enters his name and clicks OK, the InputBox returns his name to the variable **Name**.

Example 2

Dim Age as Integer

Name = InputBox("Your age please..", "CyberPro", "25")

The above statement will display the Input box as in Figure 1.14

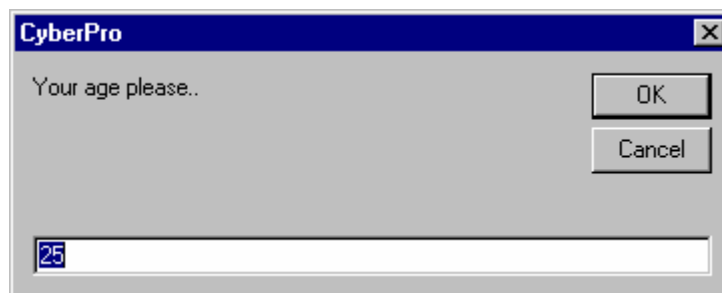


Figure 3.4 Inputbox with specified title and default value.

If the user clicks OK without entering anything the default value 25 will be returned by the InputBox to the variable Age.

Note: Though InputBox returns only strings, when they are assigned variables of other types, Visual Basic makes necessary type conversions automatically.

3.11 MsgBox Funtion

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

Syntax

`MsgBox(prompt[, buttons] [, title])`

The *prompt* and *title* arguments are same as in `InputBox` function. The *buttons* argument is a numeric expression that is the sum of values specifying the number and type of buttons to display, the icon style to use, the identity of the default button, and the modality of the message box. If omitted, the default value for *buttons* is 0. Its settings are given in the following table 1.4

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.
VbCritical	16	Display Critical Message icon.
VbQuestion	32	Display Warning Query icon.
VbExclamation	48	Display Warning Message icon.
VbInformation	64	Display Information Message icon.
VbDefaultButton1	0	First button is default.
VbDefaultButton2	256	Second button is default.
VbDefaultButton3	512	Third button is default.
VbDefaultButton4	768	Fourth button is default.

Table 3.4 Values for the buttons argument.

Example 1

Msgbox "You mistyped everything!!!. So Try Again"

This statement display a simple message box as in Figure 1.15



Figure 3.5 A simple message box.

Example 2

```
Dim Choice As Integer
```

```
Choice = MsgBox("Terrific error encountered. Proceed_
                anyway?", vbYesNo, "CyberPro")
```

```
If Choice = vbYes Then
```

```
    {some code}
```

```
Else
```

```
    {some code}
```

End If

The MsgBox function in this example, displays a message box with Yes and No button as in Figure 1.16

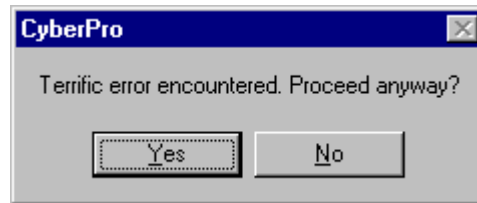


Figure 3.6 - MsgBox with specific title and two buttons.

When the user clicks one the buttons, the MsgBox function returns either `vbYes` or `vbNo`, which is subsequently checked in the code for further execution.

Example 3

More than one option can be specified in the *buttons* argument by using the plus (+) operator as in the following:

```
MsgBox("Terrific error encountered. Proceed anyway?","_vbYesNo + vbCritical + vbDefaultButton2, "CyberPro")
```

This statement displays the message box with critical icon and **No** button (2nd button) as the default button, as in the Figure1.17.

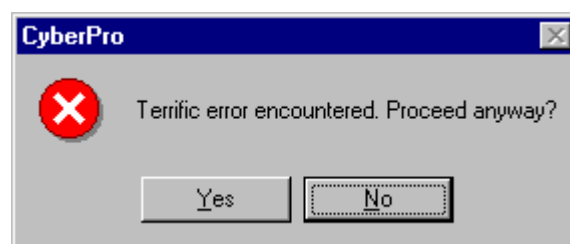


Figure 3.7 MsgBox with critical icon and 2nd button as default.

3.12 Control Flow Statements

Programs are not monolithic sets of commands that carry out the same calculations every time they are executed. Instead, they adjust their behavior depending on the data supplied or based on the result of a test condition. Visual Basic provides three-control flow, or decision, structures to take a course of action depending on the

outcome of the test. They are:

- ❖ If..Then
- ❖ If...Then...Else
- ❖ Select Case

If...Then...End If

The If structure test the condition specified and, if it it's True, executes the statements(s) that follow. The If structure can have a single-line or multiple-line syntax.

If condition Then Statement

In the above statement, Visual Basic evaluates the condition and, if it's True, executes the statement that follows. If the condition is not True, it continues with the statement following the If structure. Multiple statements can also be supplied, provided they must be separated by colon as in the following:

If condition Then Statment1:Statement2:Statement3

Here is an example of single-line If statement:

```
If Salary > 3000 Then DAPercent = 100
```

This statement can be broken into Multiple lines, as it will be easier to read as follows:

```
If Salary > 3000 Then
    DAPercent = 100
End If
```

When a statements that is to be executed, are placed next to the If statement's line, to mark the end of If block **End If** statement is used as in the above example.

If...Then...Else...End If

A variation of the If...Then is the If...Then...Else statement, which executes one block of statements if the condition is true and another if the condition is false.

Syntax

```
If condition Then
    statementblock - 1
Else
    statementblock - 2
End If
```

Visual Basic evaluates the *condition*. If it's True, it executes the first block of statements and then jumps to the statement following the End If statement. If the condition is False, Visual Basic ignores the first block of statements and executes the block following the Else keyword.

Example

```
If Salary > 5000 Then
    DaPercent = 100
    HraPercent = 80
Else
    DaPercent = 80
    HraPercent = 60
End If
```

Another variation of the If...Then...Else statement uses several conditions, with the **Elseif** keyword:

Syntax

```
If condition1 Then
    statementblock - 1
Elseif condition2 Then
    statementblock - 2
Elseif condition3 Then
    statementblock - 3
Else
    statementblock - 4
End If
```

A If statement can have any number of Elseif clauses. The conditions are evaluated from the top, and if one of them is True, the corresponding block of statements is executed. The Else clause will be executed if none of the previous expressions is True.

Example

```
If score > 90 Then
    Result = "Excellent"
Elseif score > 75 Then
    Result = "Very Good"
Elseif score > 50 Then
    Result = "Good"
Else
    Result = "Failed"
```

End If

This statement is easier to read, but not as efficient in terms of execution time because Visual Basic evaluates all conditions, even if the first one is True. This inefficiency is overcome with Select Case statement.

Select Case

The select case structure compares the same expression to a different value. The advantage of the select case statement over multiple if.then.else statements is that it makes the code easier to read and maintain.

Syntax

```
Select Case expression
  Case Value1
    statementbolck - 1
  Case Value2
    statementbolck - 2
  -----
  -----
  Case Else
    statementbolck - 3
End Select
```

Some Case statements can be followed by multiple values, separated by commas or can have a range or a single condition as given in the example below:

```
Dim Number as Integer

Number = {some integer value}

Select Case Number          ' Evaluate Number.
Case 1
  Print "Number -- 1"
Case 2,3,4,5,6              ' Number between 2 and 6
  Print "Number -- Between 2 and 6"
Case 7 To 10                ' Number between 7 and 10
  Print "Number -- Between 7 and 10"
Case Is <= 20
  Print "Number -- Between 11 and 20"
Case Else                   ' Other values.
```

```
Print "Number Not between 1 and 20"
```

```
End Select
```

3.12 Loop Statements

Loop statements allow to execute one or more lines of code repetitively. Many tasks consist of trivial operations that must be repeated over and over again, can be done using loop statements. Visual Basic supports the following loop statements:

- ❖ Do...Loop
- ❖ For...Next
- ❖ For Each...Next

Do...Loop

The Do...Loop statement is used to execute a block of statements for an indefinite number of times. There are several variations of the Do...Loop statement, but each evaluates a Boolean condition to determine whether to continue execution. As with If...Then, the *condition* must be a value or expression that evaluates to False (zero) or to True (nonzero).

The following Do...Loop executes the *statements* as long as the *condition* is True:

```
Do While condition
    statements
Loop
```

When Visual Basic executes this Do loop, it first tests *condition*. If *condition* is False (zero), it skips past all the statements. If it's True (nonzero), Visual Basic executes the statements and then goes back to the Do While statement and tests the condition again.

Consequently, the loop can execute any number of times, as long as *condition* is nonzero or True. The statements never execute if *condition* is initially False.

Example

In this example the loop block is executed while the variable I is equal to 10.

```
Dim I as Integer
I = 10
Do While I=10
```

```
    -----
    I = {someExpression} ' SomeExpression may return 10 or
```

'other value

Loop

When the condition $I=10$ is tested, the result will be True as I holds the value 10. So the execution of the loop block will begin. If *someExpression* returns 10, the loop block will be executed again. Otherwise the execution will jump to the next statement of the loop block.

Another variation of the Do...Loop statement executes the statements first and then tests *condition* after each execution. This variation guarantees at least one execution of *statements*:

```

Do
  Statements
Loop While condition
    
```

There are two more variations of Do..Loop statements, which uses **Until** instead of **While**. They are analogous to the previous two except that they loop as long as *condition* is False rather than True.

Loop zero or more times	Loop at least once
Do Until <i>condition</i> statements Loop	Do statements Loop Until <i>condition</i>

Example

This example prints the value of the variable I until it becomes equal 10.

```
Dim I as Integer
```

```
I = 0
```

```
Do Until I = 10
```

```
    Debug.Print I
```

```
    I = I + 1
```

```
End If
```

In this example when the condition $I = 10$ is tested, the result will be False and the loop block will be executed. When the I becomes equal to 10 the condition will return True, the execution will jump to the next statement of the loop block. So the output of the above code will be 0 to 9.

For...Next

Do...Loop is useful when a block of statements are to be executed for unknown number of times. But if a block of statements are to be executed for specific number of times then a For...Next loop is a better choice. Unlike a Do loop, a For loop uses a variable called a counter that increases or decreases in value during each repetition of the loop.

Syntax

```
For counter = start To end [Step increment]  
    statements  
Next [counter]
```

The arguments *counter*, *start*, *end*, and *increment* are all numeric.

The *increment* argument can be either positive or negative. If *increment* is positive, *start* must be less than or equal to *end* or the statements in the loop will not execute. If *increment* is negative, *start* must be greater than or equal to *end* for the body of the loop to execute. If Step isn't set, then increment defaults to 1.

In executing the For loop, Visual Basic:

1. Sets *counter* equal to *start*.
2. Tests to see if *counter* is greater than *end*. If so, Visual Basic exits the loop. (If *increment* is negative, Visual Basic tests to see if *counter* is less than *end*.)
3. Executes the *statements*.
4. Increments *counter* by 1 or by *increment*, if it's specified.
5. Repeats steps 2 through 4.

The following code prints the names of all the available Screen fonts:

```
Dim I As Integer  
For i = 0 To Screen.FontCount  
    Print Screen.Fonts(i)  
Next
```

For Each...Next

A For Each...Next loop is similar to a For...Next loop, but it repeats a group of statements for each element in a collection of objects or in an array instead of repeating the statements a specified number of times. This is especially helpful when the number of elements of a collection is not known.

Syntax

For Each *element* **In** *group*

statements

Next *element*

Example

This example prints the contents of array **Names**.

```
Dim Age(10) as integer
Dim Item    ' Must be variant when used with arrays

-----
-----
For Each Item In Age()
    Print Item
Next
```

With ... Endwith

WITH ... ENDWITH provides a convenient way to specify a number of properties for a single object. Note that you can also execute methods from within a WITH ... ENDWITH structure. Specifies multiple properties for an object.

Syntax

```
WITH ObjectName
    Statements
ENDWITH
```

ObjectName Specifies the name of the object. ObjectName can be the name of the object or a reference to the object.

Statements Statements can consist of any number of Microsoft Visual Basic commands used to specify properties for ObjectName. Place a period before Statement to denote that it is a property of ObjectName.

Examples

The following example creates a custom class name Employee. After the Employee class has been created with CREATEOBJECT(), WITH ... ENDWITH is used to set multiple properties for the class. The properties values are then displayed.








```
moemployee = CREATEOBJECT('employee')
```

```
WITH moemployee
    .First_Name = 'John'
    .Last_Name = 'Smith'
    .Address = '16 Maple Lane'
    .HireDate = {^1998-02-16}
ENDWITH
```

```
CLEAR
    ? moemployee.First_Name + ' '
    ?? moemployee.Last_Name
    ? moemployee.Address
    ? moemployee.HireDate
```

```
DEFINE CLASS employee AS CUSTOM
    First_Name = SPACE(20)
    Last_Name = SPACE(20)
    Address = SPACE(30)
    HireDate = { - - }
ENDDEFINE
```

3.13 Short Summary

-  Variables is an entity whose values will change during the program execution.
-  Constant is an entity whose value does not change during the program execution.
-  Arrays are nothing but user defined datatypes used to keep a set of values in it
-  Control statements are used to control the program execution .
-  Loops are used to execute a single (or) set of statements at 'n' time.
-  Input box is used to get the data thru the keyboard
-  Msg Box is used to display the messages.

3.14 Brain Storm

1. Explain variable, constant.
2. What is control statement? Explain.
3. What is a loop? Explain.
4. Define Array. What are the advantages of Array?
5. Differentiate loops and control statement?

Lab units 3 (2 Real Time Hrs)

Exercise 1

1. (Declaring Variable)
2. Open a new standard EXE project
3. Add four text boxes and a command button to the form
4. Open the code window of the form
5. Place the code Option Explicit as the first line of the declarations section of the form. This forces the variables to be declared before they are used.
6. Open the event procedure for the Click event of the command button. You can do this by selecting the name of the command button from the object list on the left of the code window.
7. Use the following statements to create the variables for calculating area and perimeter. These statements should be place at the top of the event procedure
8. Dim sngRoomLength As Single, sngRoomWidth As Single
9. Dim sngRoomArea As Single, sngRoomPerimeter As Single
10. Place the following code in the event procedure to perform the calculation. (Note, on of the variable names is intentionally misspelled to illustrate how forced declaration helps you.)

```
sngRoomLength = Val(Text1.Text)
sngRoomWidth = Val(Text2.Text)
sngRoomArea = sngRoomLength * sngRoomWidth
sngRoomPerimeter = 2 * (sngRoomLength + sng RoomWidth)
Text1.text = val(sngRoomArea)
Text2.Text = val(sngRoomPerimeter)
```

Run the program

Exercise 2

1. (Use If..Then..Else statement)
2. Open a new standard EXE project

3. Place a text box and a command button in the form
4. Enter a number in the text box
5. On clicking the command button, a message box is to be displayed
6. The message box should display whether the number in the text box is a Single, Two, Three digit numbers

Exercise 3

1. (Use Select ...Case statement)
2. Open a new standard EXE project
3. Place a text box , label and a command button in the form
4. Enter a number (0 or 1 or 2)in the text box
5. On clicking the command button, the messages
6. God's delays are not denials (for 0)
7. A Single drop makes an Ocean (for 1)
8. Hardwork and success go hand-in-hand (for 2)

Lecture - 4

Controls

Objectives

In this unit you will learn the following

- ❖ Know how to handle the Controls
- ❖ About Form object
- ❖ Understand the Properties of the controls

Lecture unit - 4

- 4.1 Snap Shot
- 4.2 Controls
- 4.3 Intrinsic Controls - introduction
- 4.4 Common properties & some importance
- 4.5 Form object
- 4.6 Intrinsic controls
- 4.7 Short summary
- 4.8 Brain storm

4.1 Snap Shot

In the previous chapter we have discussed about Variables, constants, loops and control statements. This chapter will give the ways of creating and implementing the Controls. It also helps us to understand the concept of Control Arrays. Controls are used to receive user input and display ourput and has its own set of properties, methods and events. Let us discuss few of these controls in this chapter.

4.2 Controls

Custom controls are the building blocks of a Visual Basic application. Controls are used to add significant functionality to your programs, and they are easy to work. in Visual Basic, forms are the foundations and are generally used to build programs. A form is where you put all the things that people interact with as they use your program. Those things you put on the form are controls, which enable the people who use your program to do things, such as enter text and click buttons. That is, controls are also interfaces between the user and the application. Every control has got its own properties, events and methods. You can set the properties and write the code to make the control active.

For example if you build a house, you start with a foundation – think of this as the form. On top of the foundation, you add all the things that allow you use the house: a floor, walls and doors. These things are the controls.

The Toolbox is the containing window that holds the custom controls for your applications (see Figure 4.1). There are also several advanced controls that come with Visual Basic. Some controls work with multimedia, and others utilize the Internet. Best of all, you can now create your own ActiveX custom controls and add them to your Toolbox.

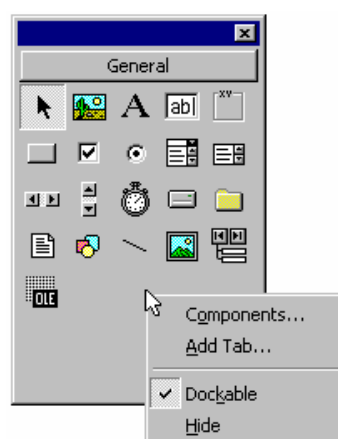


Figure 4.1 Visual Basic Toolbox.

Adding and removing Controls

You can add controls to a form in two ways: by double-clicking and by drawing. Whenever you double-click an icon on the toolbar, the associated control appears on

your form. When you do this, though, you can't control where the control goes; you're at the mercy of Visual Basic. When you draw a control on your form, you can put it wherever you want it.

Draw a control on a form

1. Click the control's Toolbox icon.
2. Move the mouse pointer over your form. Notice that your pointer is now shaped as a crosshair instead of an arrow
3. Click(and hold) the mouse button where you want the control to go.
4. Drag the mouse down slightly and to the left. As you move the mouse, notice that a box starts to appear.
5. When the box is the proper size, let go of the mouse button. The control you selected now appears on the form.

Remove a control from a form

1. Select the control you want to delete by clicking it. The control you selected will appear with a box at each corner and side
2. Press the Delete key.

You can also remove a control by right-clicking it. from the context menu that appears and selecting Delete.

How to Size and Position a Control

When you're drawing controls on a form, you don't have to be exact. It's very easy to make them bigger or smaller, and to put them in a different spot on the form.

Size controls with the mouse

In the Toolbox, select the Pointer tool (if it isn't already selected). On your form, select the control you want to resize. Grab a sizing handle with the mouse by moving the pointer over it and then holding down the left mouse button. You know when you're over the sizing handle because the mouse pointer turns into a double-sided arrow. While holding down the mouse button, notice that a box appears. The box shows you what the size of the control will be. When it's the right size, release the mouse button. Changing the position of a control is also easy. Just click it to select it, and drag it to its new position.

4.3 Intrinsic Controls

Intrinsic controls are automatically displayed in the tool box when a form is loaded. When you program in Visual Program you'll use a relatively small set of controls. However, these controls are very powerful. With them, you can add buttons, check boxes, labels and text boxes to your programs. You can use them to see files on your hard drive right from your program. You can even read a database! These basic controls are intrinsic controls.

4.4 Some Important And Common Properties

Visual Basic Forms and controls are the basic elements in the user interface of any Windows application. They are called as objects and they have their own properties, methods and they react to external events. Properties can be thought of as an object's attributes methods as its actions, and events as its responses.

An everyday objects like a child's helium balloon also has properties, methods and events. A balloon's properties include visible attributes such as its height, diameter and color. Other properties describe its state (inflated or not inflated), or attributes that aren't visible such as its age. By definition, all balloons have these properties; the settings of these properties may differ from one balloon to another.

A balloon also has inherent methods or actions that it might perform. It has an inflate method (the action of filling it with helium), a deflate method (expelling its contents) and a rise method (if you were to let go of it). Again, all balloons are capable of these methods.

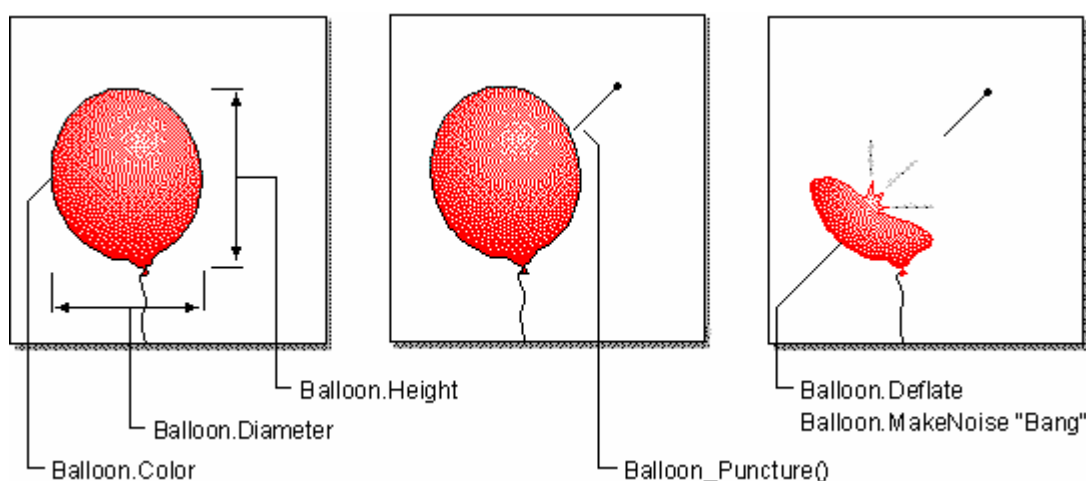


Figure 4.1 Objects have properties, respond to events, and perform methods.

Balloons also have predefined responses to certain external events. For instance, a balloon would respond to the event of being punctured by deflating itself, or to the event of being released by rising into the air.

If a balloon can be programmed, the Visual Basic code might look like the following. To set the balloon's properties:

```
Balloon.Color = Red
Balloon.Diameter = 10
Balloon.Inflated = True
```

In this code the object (Balloon) is followed by the property (.Color) followed by the assignment statement and substituting a different value. Properties can also be set in the Properties window while the application is being designed.

A balloon's methods are invoked like this:

```
Balloon.Inflate
Balloon.Deflate
Balloon.Rise 5
```

The syntax is similar to the property – the object (a noun) is followed by the method (a verb). In the third example, there is an additional item, called an *argument*, which denotes the distance to rise. Some methods will have one or more arguments to further describe the action to be performed.

The balloon might respond to an event as follows:

```
Sub Balloon_Puncture()
    Balloon.Deflate
    Balloon.MakeNoise "Bang"
    Balloon.Inflated = False
    Balloon.Diameter = 1
End Sub
```

In this case, the code describes the balloon's behavior when a puncture event occurs: invoke the Deflate method, then invoke the MakeNoise method with an argument of "Bang" (the type of noise to make). Since the balloon is no longer inflated, the Inflated property is set to False and the Diameter property is set to a new value.

While developing a Visual Basic application the programmer has to decide which properties should be changed, methods invoked or events responded to, in order to achieve his goal.

Common Properties

• Name	It sets the object's name through which the object can be manipulated.
• Appearance	Its value can be either 0 for a flat look or 1 for a 3-D look.
• BackColor	It sets the background color of an object.
• BorderStyle	It sets the Border style of an object.
• ForeColor	Its sets the foreground color of an object.
• Font	It is used to set the appearance of the text in an object.
• Width, Height	They set object's dimensions.
• Left, Top	They set the coordinates of the object's upper left corner. Using these properties, the location of an object can be changed both at design time and run time.
• Enable	It contains a Boolean (True/False) value that determines whether user can manipulate the object or not.
• Visible	It contains a Boolean (True/False) value that determines whether user can see the object or not.
• MousePointer	It sets the type of the mouse pointer displayed when over part of an object.

4.5 Form Object

The most basic object you will be working with in Visual Basic is the form object, which is the visual foundation for building an application. It is basically a window that you can add different elements to in order to create a complete application.

Every application you can see on the screen is based on some type of form. Before going into the details of application development, let's take a look at the most basic form object that you will probably use the most often in your projects: the single form.

As you learn Visual Basic, most of your applications will only have one interface, which is the single form. When you become more experienced and start writing letter, editor-styled applications, you may want to use multiple forms, called documents.

The Appearance of Forms

The main characteristic of a form is the title bar, on which the form's caption is displayed. On the left end of the title bar is the control menu icon. Clicking on this icon opens the control menu. On the right side of the title bar are three buttons—Minimize, Maximize, and Close. Clicking on these buttons minimizes, maximizes and closes the Form. When a form is maximized, the Maximize button is replaced by the Normal button, which restores the form to its size and position before it was maximized.

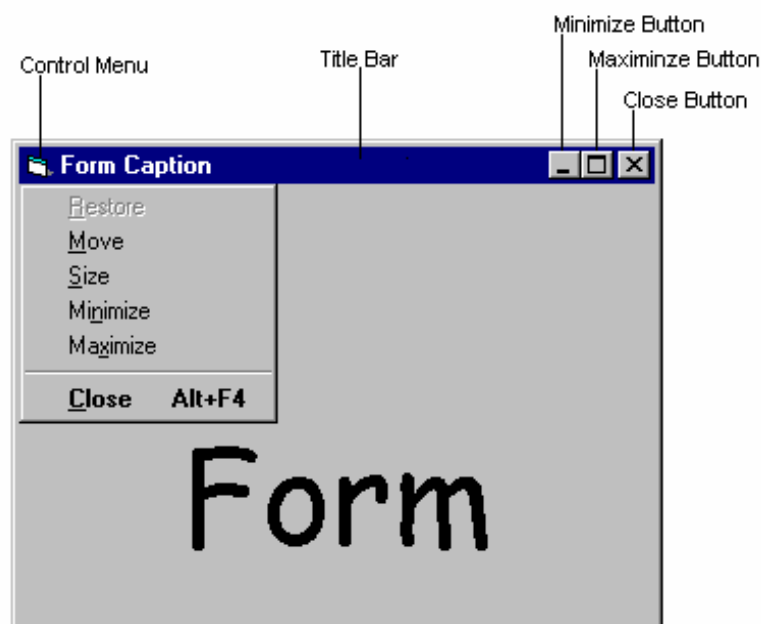


Figure 4.2 The elements of the Form.

Control Menu

Control Menu is a simple menu that allows you to restore, move, resize, minimize, maximize and close the form. To enable this button on your form, set the form's *ControlBox* property to True in the form's properties window or determines whether

the control-menu box is displayed on the form at run time. That is, if it is set to False the control box won't appear at left corner of the title bar of the Form.

The control menu contains the following commands:

- ❖ Restore Restore a maximized Form to its size before it was maximized. Enabled only if the Form is maximized.
- ❖ Move Lets the user move the form around with the keyboard.
- ❖ Size Lets the user resize the control with the keyboard.
- ❖ Minimize Minimizes the form.
- ❖ Maximize Maximizes the form.
- ❖ Close Closes the form.

The Border

The form's border is what gives the form its elasticity. Depending on the type of form you want to display, you can program the border to be fixed, sizable or even nonexistent. These features can be set with the *BorderStyle* property.

The Caption

The form's caption is the text you see in the form's title bar. It can be used to identify the name of the application, the current function of the form, or as a status bar. What you put in the caption depends on what your program is trying to achieve. If you set a form's *BorderStyle* property to None, then caption (along with the whole title bar) is hidden. You can modify the caption by setting the form's caption property in the Properties window to the text your want to display.

The Title Bar

The title bar is the colored bar on the top of most forms. If your desktop color scheme is set to the Windows default scheme, this bar will be blue. You can use the title bar to drag the window around the screen. In addition, double-clicking on it will alternately maximize and restore the form.

Maximize/Restore Button (Boolean)

The Maximize button has two purposes. If the form is in its normal state, that is its normal size, you can click the Maximize button to stretch the current form to the size of the screen or container of the form. A form's container is also known as multiple document interface (MDI) form. If the form is maximized, you can click this button

on your form, set the form's *MaxButton* property to True in the Properties window.

MinButton (Boolean)

The Minimize button is used to minimize the current form, that is, move it out of the way. To enable this button on your form, set the form's *Minbutton* property to True in the form's Properties window.

The Close Button

The Close button's sole purpose is to close the current window. In Visual Basic, you can control whether the Close button is visible to the user with the *ControlBox* property. The Close button will not be visible if the Control box, is not visible. If you decide not to enable the Close button or the Control box, they you must use a menu or a button to close the form.

Moveable (Boolean)

Determines whether the Form can be moved. That is, if it is set to False, the Form will remain in it initial startup position. User can't move it around screen. But it can be resized.

StartPosition

Specifies the position of the Form when it first appears with one the values given in the following table.

Value	Description
0-Manual	No initial is setting specified.
1-CenterOwner	Center on the item to which the UserForm belongs.
2-CenterScreen	Center on the whole screen.
3-WindowsDefault	Position in upper-left corner of screen.

Table 4.1 Values for the StartUpPosition property.

WindowState

Set the visual state such as Maximized, Minimized or Normal of the Form at run time with one the value given in the following table.

Value	Description
0-Normal	(Default) Normal.
1-Minimized	Minimized (minimized to an icon)
2-Maximized	Maximized (enlarged to maximum size)

Table 4.2 Values for WindowsState property.

Form Properties

The following shows the properties for a form object.

ActiveControl	DrawWidth	HelpContextID	NegotiateMenus
ActiveForm	Enabled	hWnd	Picture
Appearance	FillColor	Icon	ScaleHeight
AutoRedraw	FillStyle	Image	Scaelleft
BackColor	Font	KeyPreview	Scalemode
BordeStyle	FontBold	Left	ScaleTop
Caption	FontItalic	LindMode	ScaleWidth
ClipControls	FontName	LinkTopic	ShowInTaskbar
ControlBox	FontSize	MaxButton	Tag
Controls	FontStrikethru	MDIChild	Top
Count	FontTransparent	MinButton	Visible
CurrentX	FontUnderline	MouseIcon	WhatThisButton
CurrentY	ForeColor	MousePointer	WhatsThisHelp
DrawMode	hDC	Moveable	Width
DrawStyle	Height	Name	WindowState

Only few of the proprieties are used frequently, these are boldface in the above list- the rest you'll probably only use occasionally. You won't see all these proprieties in the Properties window. If you can't see a certain property, it means it's a run-time-only property and can't be set at design time.

Methods

Before listing the two important methods of the Form - Show, Hide, this section briefs the possible states of a Form and two important statements - Load and Unload.

A Form can be in one of the following states:

- ❖ Not loaded - The Form lives on a disk file and doesn't take up any resources.
- ❖ Loaded but not shown - The Form is loaded into memory, takes up the required resources, and is ready to be displayed.
- ❖ Loaded and shown - The Form is shown, and the user can interact with it.

The Load and Unload to load and unload the Forms. The Load statement has the following syntax:

Load *formName*

And the Unload statement has this syntax:

Unload *formName*

The *formName* is the name of the Form to be loaded or unloaded. When Visual Basic starts, it loads the default Form, say Form1, and displays it. To load other Forms, say Form2, the load statement is to be issued as in below:

Load Form2

This statement will load the Form2 into the memory. But the Form2 will not be visible immediately. To make it visible its Show method (explained shortly) must be called. Once a Form is loaded, it takes over the required resources, so when the Form is not needed it must be unloaded as in below:

Unload Form2

When a Form is unloaded, the resources it occupies are returned to the system and can be used by other Forms and/or applications.

Show

This method is used to show a Form. If the Form is loaded but invisible, the Show method brings the specified Form to the top. If the Form isn't loaded, the Show method loads it first, and then displays it.

Syntax

FormName.Show mode

The *FormName* is the Form's name, and the optional argument *mode* determines whether the Form will be modal or modeless. It can have one of the following values:

- 0 - Modeless (Default)
- 1 - Modal

A modal Form takes total control of the application and won't let the application proceed unless the Form is closed. A modal Form, therefore, must have a Close button or some means for the user to close it so that he can return to the Form from which the Form was loaded.

Modeless Forms are the norm. They interact with the user, and they allow the user to switch to any other Form of the application.

For example, in a text editor, say MS-Word, dialog box that appears when File | Open is chosen is a Modal dialog. It won't let user to go to the editing area unless the dialog box is closed. On the other hand, the dialog box that appears for the Find and Replace operation is a modeless dialog. It will let the user to go to the editing

area, while it is still visible.

Hide

This method removes a Form from the screen, but doesn't unload it.

FormName.Hide

When a Form is hidden, the user can't interact with it. But application's code can manipulate the Form like the one in the following:

```
Form2.Caption = "New Caption"
```

This change in the Caption property will be visible when the Form is shown again by using the Show method.

Note : Instead of calling the Show or Hide methods to show or hide a Form its Visible property can be set to True or False like this - `Form1.Visible = True`

Events

Activate

This event occurs when the Form becomes the active window either by user's action such as Clicking it, or by using the Show or SetFocus methods in code.

Deactivate

This event occurs when the Form is no longer the active window. A Form will be become inactive, when the user switches his focus to some other Form, or by the Hide method in code.

Load

This event occurs when the Form is loaded. Typically, this event procedure includes initialization code for a Form - for example, code that specifies default settings for controls, indicates contents to be loaded into ComboBox or Listbox etc.

Unload

This event occurs when the Form is unloaded. Generally, code for resetting properties to their original values or for clean up operation is placed here, if required.

Modifying a Form's Size

Every new form start using in Visual Basic will be the same size. However, this size probably won't be right for your program, so you'll have to resize it ti better fit the controls you're

using. You resize a form just as you resize a control: Grab one of its sizing handles with the mouse and drag it to the proper size.

Resize your forms to better fit the controls you're using

- ✓ Use the side sizing handles to make a control wider or narrower
- ✓ Use the top and bottom sizing handles to make a control shorter or taller
- ✓ Use a corner sizing handle to change height and width simultaneously.

Developing a Form application

This section shows how to develop a simple Form application that contains three Forms. The first Form (Form1) has command buttons to show and hide the other Forms. It also accepts Captions to change the caption of all the Forms.

To develop the Form application

- ❖ Start a new Standard EXE project.
- ❖ Choose Project | Add Form. This will display a dialog box to select the type of the Form. Accept the default selection Form and click Open to add the second Form to the Project.
- ❖ Repeat the above step to add the third Form to the Project.
- ❖ In the Project Explorer window double click the Form1 to bring its design window front.
- ❖ Create objects and set their properties as given below.

OBJECT	PROPERTY	VALUE
Label	Caption	"FORMS DEMO"
Command Button	Name	ShowFrm2
	Caption	"Show Form 2"
Command Button	Name	ShowFrm3
	Caption	"Show Form 3"
Command Button	Name	HideFrm2
	Caption	"Hide Form 2"

Command Button	Name	HideFrm3
	Caption	"Hide Form 3"
Label	Caption	"New Caption for Me"
Label	Caption	"New Caption for Form2"
Label	Caption	"New Caption for Form3"
Text box	Text	""
Text box	Text	""
Text box	Text	""
Command Button	Name	ChangeCaps
	Caption	"Change Forms' Caption"

❖ Arrange the controls so that the Form1 resemble the Figure 1.21

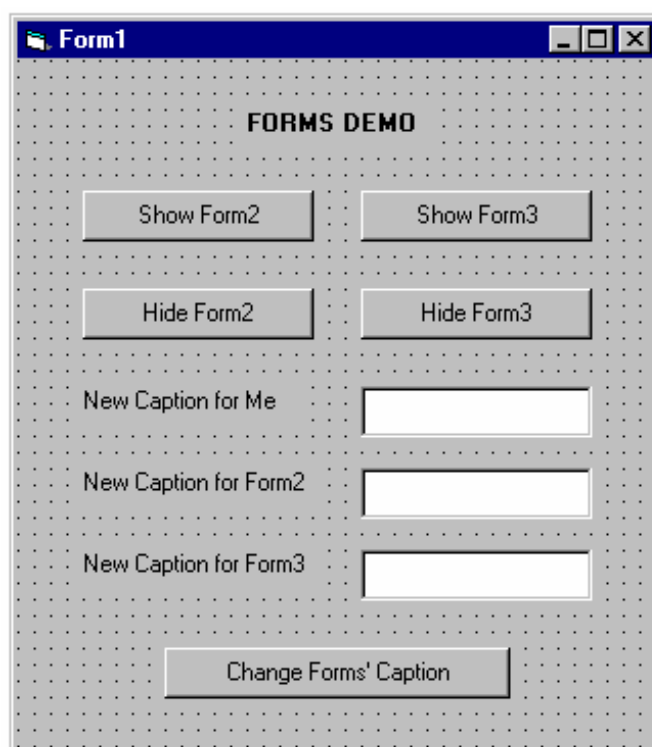


Figure 4.2 Designed Form1.

- ❖ Add the following code to the Form1.

Option Explicit

```
Private Sub ChangeCaps_Click()  
    ' Changing the Captions of all Forms  
    Form1.Caption = Text1.Text  
    Form2.Caption = Text2.Text  
    Form3.Caption = Text3.Text  
End Sub
```

```
Private Sub Form_Load()  
    ' Loading the other Forms  
    Load Form2  
    Load Form3  
End Sub
```

```
Private Sub Form_Unload(Cancel As Integer)  
    ' Unloading the other Forms  
    Unload Form2  
    Unload Form3  
End Sub
```

```
Private Sub HideFrm2_Click()  
    ' Hiding the Form2  
    Form2.Hide  
End Sub
```

```
Private Sub HideFrm3_Click()  
    ' Hiding the Form3 using its Visible property  
    Form3.Visible = False  
End Sub
```

```
Private Sub ShowFrm2_Click()  
    ' Showing the Form2  
    Form2.Show  
End Sub
```

```
Private Sub ShowFrm3_Click()  
    ' Showing the Form3 using its Visible property  
    Form3.Visible = True  
End Sub
```

- ❖ Save and run the Project.

- ❖ Click the buttons to show and hide the Forms 2 and 3.
- ❖ Enter new captions and click the “Change Forms’ caption” button to change the captions of all the Forms.

4.6 Common Intrinsic Controls

Controls are elements that provide the User Interface. The User Interface is what appears in the application's window when it runs. It enables the user to work with the application such as entering text and clicking buttons. The elements of the User Interface are common to all Windows applications and they are called as Intrinsic Controls. They are all shown as icons in the Visual Basic ToolBox.

The Text box control



It provides an area to enter or display text. It behaves like a mini text editor.

The Label Control



This control displays text on a form that the user can't edit. Labels commonly identify other controls.

The Command button control



Carries out a command or action when a user chooses it.

The Check box control



Displays a True/False or Yes/No option. Any number of check boxes can be checked on a form at one time.

The Option button control



Option buttons, or radio buttons, appear in a group, and the user can choose only one of them

The List box control



It contains a list of options from which the user can chose one or more. It can contain many lines, and the user can scroll the list to locate an item.

The Combo box control



The combo box control is similar to the List box control, but it contains a text edit field. The user can either choose an item from the list or enter a new string in the edit filed.

The Picture box control



The picture box control is used to display bitmaps, icons or Windows metafiles.



The Image Control

The image control is similar to the PictureBox control and it can also display images, but it supports only a few features of the Picture box control and requires fewer resources.

The Shape control



The shape control is used to draw graphical elements, such as boxes and circles on the surface of a Form.

The Line control



The line control is used to draw lines on the surface of a form.

The Frame control



This control is used to draw boxes on the form and to group other elements.

The Drive list box control



It displays the drives on the system in a drop-down list from which the user can select a valid drive.

The Directory list box control



It Displays a list of all folders in the current drive and lets the user move up or down in the hierarchy of the folders.

The File list box control



It displays a list of all files in the current folder.

The Timer Control



It is used to perform tasks at regular intervals.

The Horizontal and Vertical scroll bars



These controls can be used as inputs devices or indicators of speed or quantity - for example, to control the volume of a computer game or to view the time elapsed in a timed process.

Developing an application with Intrinsic Controls

This section shows how to create an simple application with intrinsic controls. Functioning of this application is explained shortly.

To develop the application

- ❖ Start a new Standard EXE project.
- ❖ Name the Project as CtrlDemo.
- ❖ Create the objects and set their properties as given below:

Object	Property	Value
Form1	Caption	"Controls' Demo"
Label	Caption	"Intrinsic Controls"
Shape	Shape	4- Rounded Rectangle
Frame	Name	fraView
	Caption	View
CheckBox	Name	chkExplorer
	Caption	Explorer
CheckBox	Name	chkAnimation
	Caption	Animation
Frame	Name	fraAnimation
	Caption	Animation
OptionButton	Name	optCombo
	Caption	Combo box
OptionButton	Name	optList
	Caption	List Box
Listbox	Name	lstVehicle
	Visible	False
ComboBox	Name	cmbVehicle
	List	Taxi
		Tempo
		Auto
Label	Caption	Speed
Line	BorderStyle	3-Dot
Label	Name	lblSpeed
	Caption	10
HScrollBar	BorderStyle	1-Fixed Single
	Name	hsbSpeed
	Max	500
	Min	10
	LargeChange	30
	SmallChange	10

Timer	Name	Timer
	Interval	500
	Tag	Forward
Image	Name	imgVehicle
	Picture	"Taxi.bmp"
Frame	Name	fraExplorer
	Caption	Explorer
DriveListBox	Name	Drive
DirListBox	Name	Directory
FileListBox	Name	File
Label	Caption	"Full Path"
Label	Name	lblPath
	Caption	""
	BorderStyle	1-Fixed Single

Arrange the controls so that it resemble the Figure 1.22

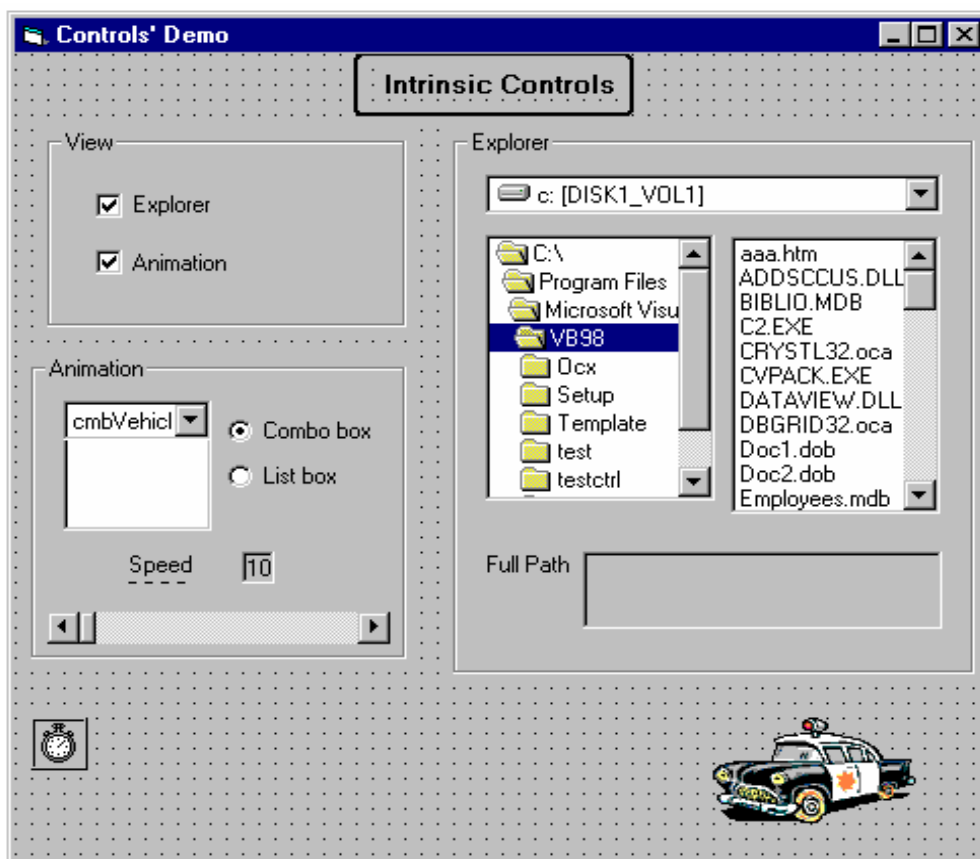


Figure 4.3 Arranged Form with Intrinsic controls.

The above Form contains three Frames namely View, Explorer and Animation. The View Frame has two check boxes whose checking and unchecking makes the respective Frames Visible and invisible at run time. The second Frame contains Drive, Directory and File list boxes, which behave like a mini Windows Explorer. The label (lblPath) displays the full path of the selected file in the File list box. The third frame contains a combo box, a list box and two corresponding option buttons. Either the combo box or the list box will be visible at the run time. Selecting one of the option buttons will make its corresponding control (combo box or list box) visible and another invisible (Note that both List box and combo box are placed at a same location). Both List box and combo box contains three items namely Taxi, Tempo, and Auto whose selection will replace its corresponding picture in the image control (imgVehicle). This control will move in forward and reverse direction by adjusting its Left property according to the value set by the Horizontal scroll bar in the Animation frame.

❖ Add the following code to the Form.

Option Explicit

```
Private Sub chkAnimation_Click()
    ' Displaying and Hiding Animation Frame and Image
    ' control
    If chkAnimation.Value = 1 Then
        fraAnimation.Visible = True
        imgVehicle.Visible = True
    Else
        fraAnimation.Visible = False
        imgVehicle.Visible = False
    End If
End Sub
```

```
Private Sub chkExplorer_Click()
    ' Displaying and Hiding Explorer Frame
    If chkExplorer.Value = 1 Then
        fraExplorer.Visible = True
    Else
        fraExplorer.Visible = False
    End If
End Sub
```

```
Private Sub cmbVehicle_Click()
    ' Loading a new image corresponding to the selected
    ' item
    imgVehicle.Picture = _
```

```
        LoadPicture(cmbVehicle.Text + ".gif")
    End Sub

    Private Sub Directory_Change()
        ' Assigning the newly selected directory to the file
        ' list box
        lblPath.Caption = ""
        File.Path = Directory.Path
    End Sub

    Private Sub Drive_Change()
        ' Assigning the newly selected directory to the
        ' directory list box
        lblPath.Caption = ""
        Directory.Path = Drive.Drive
    End Sub

    Private Sub File_Click()
        ' Displaying the full path of the selected file
        lblPath = Drive.Drive + "\" + Directory.Path + "\" + _
            File.FileName
    End Sub

    Private Sub Form_Load()
        ' Adding items to list box at run time using AddItem
        ' method
        lstVehicle.AddItem "Taxi"
        lstVehicle.AddItem "Tempo"
        lstVehicle.AddItem "Auto"
    End Sub

    Private Sub hsbSpeed_Change()
        ' Setting HScroll bar's value to the label lblSpeed
        lblSpeed.Caption = hsbSpeed.Value

        ' Changing the interval for the occurrence of the
        ' timer event.
        ' Note: Though higher value of the HScroll bar deems
        ' as increase the speed, only low value can make the
        ' timer event to occur more ' quickly. So subtracting
        ' from 510...
        Timer.Interval = 510 - hsbSpeed.Value
    End Sub

    Private Sub lstVehicle_Click()
```

```
' Loading a new image corresponding to the selected
  ' item
  ImgVehicle.Picture = _
      LoadPicture(lstVehicle.Text + ".gif")
End Sub
Private Sub optCombo_Click()
  ' Making combo box visible and list box invisible
  cmbVehicle.Visible = True
  lstVehicle.Visible = False
End Sub

Private Sub optList_Click()
  ' Making combo box invisible and list box visible
  cmbVehicle.Visible = False
  lstVehicle.Visible = True
End Sub

Private Sub Timer_Timer()
If Timer.Tag = "Forward" Then
  ' Moving in forward direction (left)
  ' (Note: Image is facing the left side).
  ImgVehicle.Left = imgVehicle.Left - 100

  ' If image has reached Form's left...
  ' Taking reverse
  If imgVehicle.Left <= Form1.Left Then
    Timer.Tag = "Reverse"
  End If
Else
  ' Moving in reverse direction
  imgVehicle.Left = imgVehicle.Left + 100

  ' No right property. So..
  ' Finding right position by adding Left and Width
  ' properties
  If imgVehicle.Left + imgVehicle.Width >= Form1.Width Then
    Timer.Tag = "Forward"
  End If
End If
End Sub
```

- ❖ Save, Run and Test the application.

The Naming Convention of Intrinsic Controls

It is an art to choose suitable names to controls. The name of a control should reveal its purpose and also its type. The name of a control should be chosen, in such a way that it should give appropriate meaning about the usage of that control and one of the following abbreviations should be prefixed with its name according to its type.

Control	Abbreviation	Control	Abbreviation
Label	lbl	PictureBox	pic
Frame	fra	TextBox	txt
Checkbox	chk	Command Button	cmd
ComboBox	cbo	Option Button	opt
HscrollBox	hsb	ListBox	lst
Timer	tmr	VscrollBox	vsb
DirListBox	dir	DriveListBox	drv
Shape	shp	FileListBox	fil
Image	img	Line	lin
OLE Container	ole	Data	dat

Common methods

❖	Move	Changes an object's position in response to a code request.
❖	Drag	Handles the execution of a drag-and-drop operation by the user.
❖	SetFocus	Gives focus to the object specified in the method call.
❖	ZOrder	Determines the order in which multiple objects appear on screen.
❖	Refresh	Forces a complete repaint of a form or object.
❖	TabIndex	Determines the tab order of most objects within their parent form.

Common Events

Events determine the control's reactions to external conditions. Events are recognized by the various controls, but are handled by the application. A command button will recognize that it is clicked upon, but it won't react to the event unless some code is provided in the application.

If a subroutine is specified for the control's click event, this subroutine executes each time the control is clicked. The subroutine that determines how a control reacts to an event is called an *event handler*.

To write an event handler for a control, follow these steps:

- ❖ Switch to the Code window, or double click on the control, which requires the event handler. The top of the Code window contains two drop down lists (Figure 1.23). The first list contains the names of all the controls on the Form.
- ❖ Select the control, which requires the event handler. The second list contains all the events the selected control can recognize.
- ❖ Select the event, which requires the event handler.

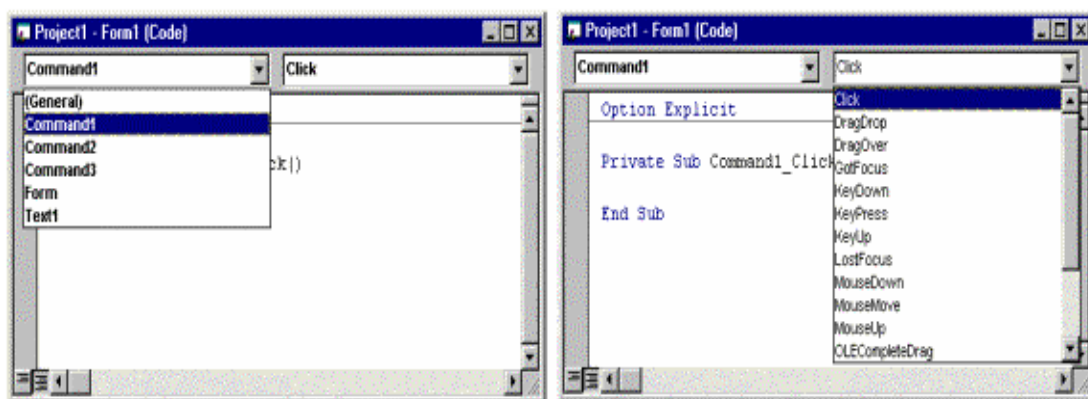


Figure 4.4 Drop lists – listing controls and event in the Code Window

The combination of the control's name and the event's name is unique and is the name of the event handler. For example, the event handler name for the Command1's click event will be Command1_Click. Each time an event takes place, Visual Basic looks for the subroutine made up of the name of the control and on which the event took place and the name of the event. If such a handler exists, it's executed. If not, the application won't react to the event.

The following is the list of common events:

-
- ❖ Click The user clicks the primary (left) mouse button on an object.

❖	DbClick	The user double clicks the primary (left) mouse button on an object.
❖	Dragdrop	The user drags an object to another location.
❖	DragOver	The user drags an object over another control.
❖	GotFocus	An object receives focus.
❖	KeyDown	The user presses and releases a keyboard key while an object has focus.
❖	KeyUp	The user releases a keyboard key while an object has focus.
❖	LostFocus	An object loses focus.
❖	MouseDown	The user presses any mouse button while the mouse pointer is over an object.
❖	MouseMove	The user moves the mouse pointer over an object.
❖	MouseUp	The user releases any mouse button while the mouse pointer is over an object.
❖	Change	Indicates that the contents of the control have changed.

4.6.1 The Command Buttons

A command button control is one of the most common controls found in Windows applications. Visual Basic is no exception. You can use a command button to elicit sample response from the user or to invoke special functions on forms. Surely you have encountered command buttons before. Every time you click the OK button on a dialog box you are clicking a command button.

Command Button Properties

Below is the list of the properties of a command button. The most commonly used properties appear in boldface.

Appearance	Enabled	HelpContextID	TabIndex
BackColor	Font	hWnd	TabStop
Cancel	FontBold	Index	Tag

Caption	FontItalic	Left	ToolTipText
Container	FontName	MaskColor	Top
Default	FontSize	MouseIcon	UseMaskColor
DisabledPicture	FontStrikethru	Name	Value
DownPicture	FontUnderline	Parent	Visible
DragIcon	ForeColor	Picture	WhatsThisHelp
Dragmode	Height	Style	Width

Caption Determines the text that appears on the command button. Placing an ampersand character (&) in the caption gives a keyboard access key alternative to a mouse-click. You access these controls by holding the Alt key down while you press the underlined letter of the control you wish to access. The user could also tab to the command button and press the spacebar to simulate a mouse-click on the button.

Default (*Boolean*) Determines whether the command button is the default button or not. That is, if a command button's Default property setting is True and its parent Form is active, the user can choose the command button (invoking its Click event) by pressing ENTER. Any other control with the focus doesn't receive a keyboard event (KeyDown, KeyPress, or KeyUp) for the ENTER key unless the user has moved the focus to another command button on the same form. In this case, pressing ENTER chooses the command button that has the focus instead of the default command button.

Cancel (*Boolean*) Determines whether the command button is the cancel button or not. That is, if a command button's Cancel property setting is True and its parent Form is active, the user can choose the CommandButton by clicking it, pressing the ESC key, or pressing ENTER when the button has the focus.

Note: Only one Command Button control on a Form can be the Cancel button. When the Cancel property is set to True for one Command Button, it's automatically set to False for all other Command Button controls on the form. Likewise only one Command button can be the Default button.

Name The Name property is used to give the control its own identity. This name is used by your code and Visual Basic to distinguish it from the rest of the controls.

Style Indicates the display type of the control. It can have any one of the two values as in the following table.

Value	Description
0-Standard	Default. Displays the command button as a standard button without any associated graphic.
1-Graphical	Displays the standard button that can also have an associated graphic.

Table 4.6 Values for Style property.

BackColor This property's value of a command button will take effect, only when the Style property is set to Graphical. More over, the command button can display a picture by using the Picture property (explained next).

Enable This property can stop the user from accessing a command button. If either is set to false, the command button will be unusable. Disabling a command button is a handy technique if you want to force the end user to complete certain actions(such as filling in text boxes) before clicking the next button in the process.

TabIndex A control with a *TabIndex* of 0(zero) is the first to receive the focus on a form, provided it's not disabled or invisible.If you alter the *TabIndex* of the one control, the other controls' orders adjust to accommodate the new order.

TabStop Determines the user to prevent from tabbing to a control, set it to False, though this does not prevent a mouse-click on the control - to stop that, use the Enable or Visible properties described previously.

Picture Sets the picture to be displayed in a command button, provided the command button's Style property must be set to Graphical. The value part of the Picture property in the Properties window has a button (...), which enables to choose a bitmap, icon, metafile, GIF, or JPEG file. For example, if a command button has a caption "Play Music" and Picture "..\Key.bmp", it will appear as in Figure 1.24



Figure 4.5 Command Button with Caption and Picture.

Note: The Picture also available to Form, Check box, Option button, Picture Box and Image control.

Command Button Events

Without a doubt, the most frequently coded event procedure for a command button corresponds to the Click event; the other events for a command button are listed here:

Click	GotFocus	KeyUp	MouseMove
DragDrop	KeyDown	LostFocus	MouseUp
DragOver	KeyPress	MouseDown	

In your first programs, the Click event is probably the only event in which you'd be interested. It's the most commonly used event on a command button. You won't use many of the other events until you become more proficient in Visual Basic. You can also use the MouseUp event in place of the Click event. Many Windows applications use this event so it gives the user a chance to back out without causing a Click event.

Command Button Methods

Listed below are the methods for the command button. The most commonly used method is SetFocus.

Drag	Refresh	ZOrder	Move	SetFocus
------	---------	--------	------	----------

The SetFocus method is sometimes used to place the focus on a particular button. This comes in handy if you want the user to return to a default button after editing a text box on a form. If that were so, the code for the focus button looks like this:

```
cmdMyButton.SetFocus
```

and it might be placed in the Change event procedure for a text box.

4.6.2 Frame Control

The Frame Control is not particularly useful. The controls normally placed in a frame are option buttons and check boxes. This has the effect of grouping them together so when the frame is moved, the other controls move, too. For this to work you can't double-click a control (say, an option button) to add it to the form, and then drag it into position within the frame. Instead, you must single-click the control in the Toolbox and drag a location for it inside the frame. Then all the controls move together.

In addition, the option buttons function as a group—that is, if you select one at run time, the others become deselected. If you simply scatter option buttons randomly, on a form, then they all function as one large group. To create separate groupings of option buttons, you place them in frames. The button items within each frame act as

a self-contained group and have no effect on the option buttons in other frame groups.

Although a frame is often used as a container for check box groups too, each check box is completely independent. Thus the setting for one check box has no effect on the setting for the others in the same group. This is the behavior you would expect of check boxes. Checkboxes are not mutually exclusive. This contrasts with option buttons, where the buttons within a single group should be mutually exclusive. The reason then for placing check boxes in a frame is to enable you to move the group as a whole, when you reposition the frame at the design time. The frame also serves as a visual grouping for the check boxes. For example, the check boxes relating to a particular feature can be in one frame and those pertinent to another feature in another frame.

A frame is usually given an *fra* prefix. You place the frame on the form before you place the controls it's going to contain.

Frame Properties

The frame control has several properties, listed below:

Appearance	Font	Height	Parent
BackColor	FontBold	HelpContextID	TabIndex
Caption	FontItalic	hWnd	Tag
Container	FontName	Index	ToolTipText
ClipControls	FontSize	Left	Top
DragIcon	FontStrikethru	MouseIcon	Visible
Dragmode	FontUnderline	MousePointer	WhatsThisHelp
Enabled	ForeColor	Name	Width

After the Name property, perhaps the single most important property is the Caption. You use this to give a meaningful title to the frame on the form. Then it's clear to the end user which feature the option buttons (or Check boxes) in the frame refer to. The provide a clue as to how each option button affects the feature, you use the Caption property of the buttons. For example, in order dispatch system you might have a frame with the caption Delivery. And within that frame you might have two option buttons, with the captions Normal and Express.

Frame Events

The frame control only supports a few events:

Click	MouseDown
DbClick	MouseMove
DragDrop	MouseUp
DragOver	

The frame control events are only rarely used. In an application the uses drag-and-drop, however, the DragDrop event is sometimes used to initiate actions when the user drops an object into a frame area.

Frame Methods

A frame object supports only a few methods. None are very helpful and they're hardly ever seen in Visual Basic projects:

Drag	Refresh	Zorder	Move	ShowWhatsThis
------	---------	--------	------	---------------

4.6.3 Option Button



Option button controls (shown here) are used to allow the user to select one, and only one, option from a group of options. Usually option buttons are grouped together within a frame control, but they can also be grouped on a plain form, if there is to be only one group of option buttons. Thus, if you had a frame specifying a delivery method, you might have one button for UPS (United Parcel Service) and another for Courier delivery. Products can only be shipped by one of these methods (not both and not none). In contrast, option buttons representing, say bold and italic settings for text would not make sense. Text can be both bold and italic, or neither (none).

Option Button Properties

The option button supports many properties which are shown in the table below:

Alignment	Enabled	HelpContextID	TabIndex
Appearance	Font	hWnd	TabStop
BackColor	FontBold	Index	Tag
Caption	FontItalic	Left	ToolTipText
Container	FontName	MaskColor	Top
DisabledPicture	FontSize	MouseIcon	UseMaskColor
DownPicture	FontStrikethru	Name	Value
DragIcon	FontUnderline	Parent	Visible
Dragmode	ForeColor	Picture	WhatsThisHelp
	Height	Style	Width

Once again the Name property is the one to set first; option buttons have an opt prefix by convention. The Caption property helps the user determine the purpose of an option button. The other popular property is Value. This is invaluable at both design time and run time. At run time you test the Value property to see if the user has turned on (or off) the option button. The property has two settings, True or False. At design time you can set the Value property to True for one of the buttons if you

wish – the default setting is False. This means that the option button (and only that option button in a group) is pre-selected when the form opens. If you try to make Value for another button in the group True, then the previous one reverts to a False setting.

Option Button Events

The option button control has a few events, but only the click event is really used:

Click	DbClick	DragDrop	DragOver	GotFocus
KeyDown	KeyPress	KeyUp	LostFocus	MouseDown
MouseMove	MouseUp			

The typical way of dealing with option buttons is to test the Value property at run time to see if they're selected. Your code then initiates actions accordingly. It's common to test for the Value property in the Click event procedure for a command button that's clicked after the user has select the option button of interest. This allows you to check for a condition before the next procedure is called. You test the Value property in an If...End If or Select Case.....End Select construct. But there may be occasions when you want to initiate an action immediately after the user makes a choice. Then you may want to trap the option button's Click event.

Example

1. Run the Controls project by selecting Run ► Start
2. Click The Option Button on the Control Examples form
3. Click on any of the option buttons and watch the label at the top of the form. The click event of each option button is used to change the Caption property of the label.
4. When you are done watching the results, click the Close button to close the dialog box
5. End the application by clicking the Exit button on the Control Examples form.

Option Button Methods

The methods for the option button are of little use in the Visual Basic environment.

Drag	Move	RefreshSetFocus	ShowWhatsThis
Zorder			

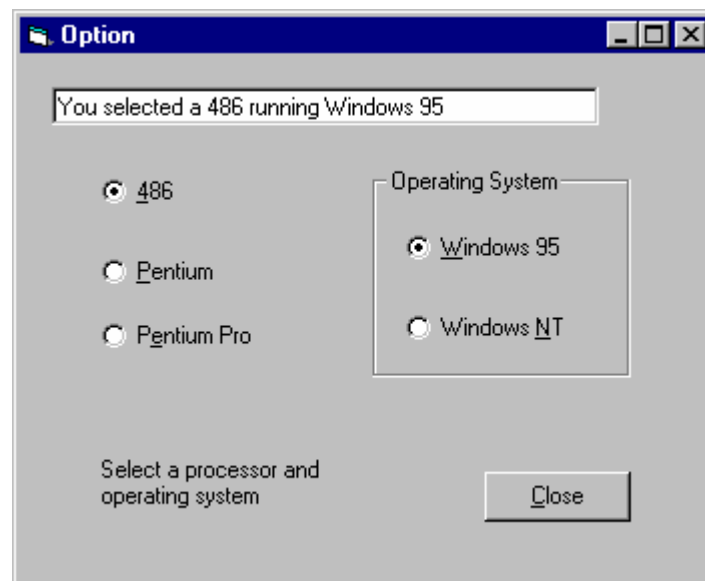


Figure 4.6 The Option dialog box

4.6.4 Check Box Control

A Check Box control is rather similar to an option button. Both often partake in groups, and the Value property is tested to see if a check box is on or off. But there are two fundamental differences between check boxes and option buttons: Check boxes are valid as single controls – a single option button is probably counter-intuitive. Check boxes (even when a group) are not mutually exclusive. Finally , check boxes have three possible settings for the Value property.

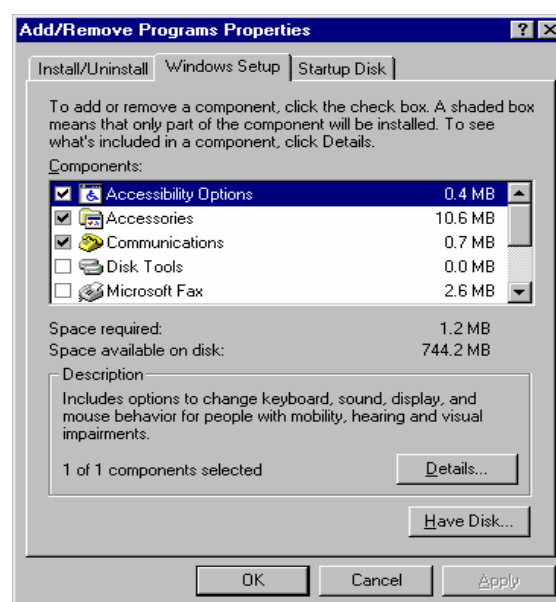


Figure 4.6 Grayed or “dimmed” check boxes

An option button is either on or it's off. Therefore, Value can be either True or False. Check boxes can be in one of three states - on, off, or grayed. Grayed(dimmed) does not mean the same as disabled in this context- a grayed check box is neither on nor off, though the user can change its setting. If the check box were disabled, the user wouldn't be able to turn it on or off. A grayed check box is used to signify that some, but not all, option on another dialog box are selected. If you look at the dialog box in Figure 1.26, you will notice two of the check boxes are grayed. The Accessories check box is gray because it was installed.

Check Box Properties

The following table lists the properties for the check box control:

Alignment	Enabled	HelpContextID	TabIndex
Appearance	Font	hWnd	TabStop
BackColor	FontBold	Index	Tag
Caption	FontItalic	Left	ToolTipText
Container	FontName	MousePointer	Top
DisabledPicture	FontSize	Name	UseMaskColor
DownPicture	FontStrikethru	Parent	Value
DragIcon	FontUnderline	Picture	WhatsThisHelp
Dragmode	ForeColor	Style	Width
DataChanged	Height		
DataField			

Value Indicates the state of the control. It can have any one of the values - 0-Unchecked, 1-Checked, and 2-Grayed.

Style Indicates the appearance of the control, whether standard or graphical. If it has the value Graphical then the check box will appear as a command button as in Figure 1.27

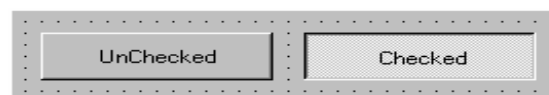


Figure 4.7 Check boxes whose Style property has the Graphical value.

DisabledPicture Sets the graphic to be displayed when the button is in the disabled, if the Style is set to Graphical.

DownPicture Sets the graphic to be displayed when the button is in down position, if the Style is set to Graphical.

Note: The above four properties are same for the Option button also.

Check Box Events

The following table shows you that the check box has similar events to the option button control:

Click	DragDrop	DragOver	GotFocus
KeyDown	KeyPress	KeyUp	LostFocus
MouseDown	MouseMove	MouseUp	

To carry out further processing as soon as the user has clicked the check box, use the click event. Normally, though, if you do not put code directly within the click event, you will have another procedure that will ascertain the value of a check box by retrieving its Value property.

Check Box Methods

Drag	Refresh	ShowWhatsThis
Move	SetFocus	Zorder

Like option buttons, the methods for the check box control are non-vital to the operation of the control.

4.6.5. The Picture Box Control

As you might expect, picture boxes often graphics (for example, bitmaps, icons, JPEGs and GIFs). In this role, picture boxes are similar to image controls. However, picture boxes and images have slightly different properties and therefore behave differently. If you just want to show a picture, then a image control is usually a better choice than a picture box. Images take up less memory and are a lightweight version of picture boxes. However, if you want to move the graphic around the form, a picture box produces a smoother display. In addition, you can create text and use graphics methods in a picture box at run time. The graphics methods enable you to draw lines, circles and rectangles at run time. But most importantly for this application, picture boxes can act as containers for other controls. Thus you can place a command button within a picture box. In this respect, picture boxes function as “forms within forms”

Picture Box Properties

Picture Sets the graphic to be displayed in the control. At design time a graphic file name can easily be specified through the Properties Window. But to assign a picture at run time the LoadPicture statement is used as in the following:

```
Picture1.Picture = LoadPicture("C:\Windows\Clouds.bmp")
```

The LoadPicture statement loads the Clouds.bmp file from the disk to memory and assigns it to the picture property of the Picture1 control.

AutoSize (*Boolean*) Determines whether a control is automatically resized to display its entire contents.

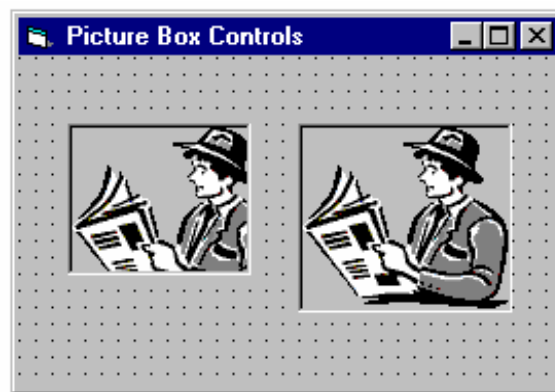


Figure 4.8 Different appearance of a same picture due to different values of the AutoSize property of the Picture box Control.

In the above Figure 4.8 the first Picture box control has the value False in its AutoSize property. So only portion of a picture that can fit into the current coordinates of the first Picture box is displayed. The second Picture box has the value True in its AutoSize property. So, it is automatically resized to accommodate the entire picture.

Picture Box Method

The Picture box control supports more methods than its counterpart, the image box. The most important ones are listed in boldface in the following table:

Circle	LinkPoke	Point	SetFocus
Cls	LinkRequest	Print	ShowWhatsThis
Drag	LinkSend	PSet	TextHeight
Line	Move	Refresh	TextWidth

LinkExecute

PaintPicture

Scale

ZOrder

The Circle, Cls, Line, Print and Pset methods are all concerned with drawing graphics or text in the picture box at run time - Cls (like the old DOS command for "clear screen") is actually used to erase entries. The ZOrder method is the run-time equivalent of Format Order Bring to Front or Format Order Send to Back. You can use ZOrder to determine which controls overlap other controls. However, you should be aware that there are three layers on a form- ZOrder only works within the layer that contains the control. All the nongraphical controls except labels (for example, command buttons) belong to the top layer. Picture boxes and other graphical controls (labels) belong to the middle layer. The bottom layer contains the results of the graphics methods- for instance, a circle drawn with the circle method is on the bottom layer. This contrasts with a circle drawn with the shape control, which is in the middle layer. What all this means is that you can't position a picture box over a command button with ZOrder- the picture box is permanently relegated to the layer behind. The ZOrder method is for rearranging objects within one layer.

PaintPicture Draws the contents of a Picture box control in the another picture box control.

Syntax

Picturebox.PaintPicture *picture, x1, y1, width1, height1, x2, y2, width2, height2* Each argument is described in the following table.

Part	Description
Picture	Required. The source of the graphic to be drawn onto <i>object</i> . Must be the Picture property of a PictureBox.
X1, y1	Required. Single-precision values indicating the destination coordinates (x-axis and y-axis) on <i>object</i> for <i>picture</i> to be drawn.
Width1	Optional. Single-precision value indicating the destination width of <i>picture</i> . If omitted, the source width is used.
Height1	Optional. Single-precision value indicating the destination height of <i>picture</i> . If omitted, the source height is used.
x2, y2	Optional. Single-precision values indicating the coordinates (x-axis and y-axis) of a clipping region within <i>picture</i> . If omitted, 0 is assumed.
Width2	Optional. Single-precision value indicating the source width of a clipping region within <i>picture</i> . If omitted, the entire source width is used.

Height2	Optional. Single-precision value indicating the source height of a clipping region within <i>picture</i> . If omitted, the entire source height is used.
----------------	--

Table 4.7 Arguments of the PaintPicture method.

Examples

The following line shows how to copy the contents of the Picture1 control to the Picture2 control.

```
Picture2.PaintPicture Picture1.Picture, 0, 0
```

As optional arguments are omitted, their default values will be used. The following line shows how to display mirror images of the Picture1 control in the Picture2 control.

```
Picture2.PaintPicture Picture1.Picture, 0, 0, _  
Picture1.Width, Picture1.Height, Picture2.Width, 0, _  
-Picture2.Width, Picture2.Height
```

Here, destination X's origin is not 0, but the picture's width and the destination width are the negative of the actual width. So the contents of the Picture1 will be displayed in the reverse order to produce the mirror image.

4.6.6 The Image Box Control

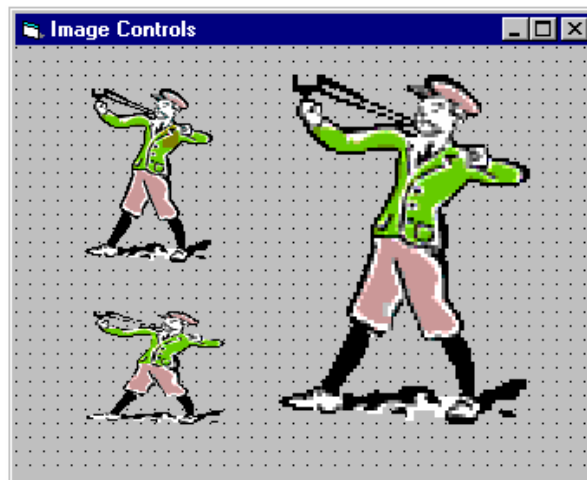
The image control (Prefix -img) is a lightweight equivalent of the picture box control. But unlike the picture control, the image control can't act as a container for other objects. In some of its other properties it's not as versatile, but it's a good choice if you simply want to display a picture on a form. Image controls consume far less memory than picture controls.

The image control that comes with Visual Basic can now display bitmap(.BMP), icon(.ICO), metafile(.WMF), JPEG(>JPG) and GIF(.GIF) files. This makes it easier to display graphics from the WorldWideWeb, as well as graphics from other popular graphics program.

Image Box Properties

Picture Same as in Picture Box control.

Stretch (*Boolean*) Indicates whether the picture has to be resized to fit the size of the control. This property is totally different from the Picture box control's AutoSize. When the AutoSize is set to False, only portion of the picture that fits into the current coordinates of the Picture box is displayed. But when Stretch is set to False the Image control is resized to the coordinates of the picture. That is, it behaves in the



similar manner of the Picture box control when its AutoSize is set to True. But when the Stretch is set to True the picture will be enlarged or compressed to the current coordinates of the Image Control.

Figure 4.8 Different appearance of a same picture due to different values of the Stretch property of the Image Control.

Note: The Image control will not display border around its contents as like the Picture box.

In the Figure 1.29 the first image control (left-top) has the value False in its Stretch property. So this control is resized to accommodate the entire contents of the picture. But the other two image controls are resized to difference coordinates- One less than and another greater than the original size of the picture, and their Stretch property is set to True. So the picture is shrunk and enlarged accordingly.

Note: When the Image control's Stretch property is set to True, resizing the control also resizes the picture it contains. But in contrast to this, resizing a Picture box control will not resize its picture, even though its AutoSize property is set to True.

4.6.7. The Timer Control

The timer control is one of the few controls always hidden at run time. This means you don't have to find room for it on a form-it can go anywhere,even on top of existing controls. The timer basically does just one thing: It checks the system clock and acts accordingly.

Timer Properties

Interval Sets the number of milliseconds between calls to the control's Timer event. That is, if this property is set to the value 1000, the control's Timer event will be executed at every second (1000 milliseconds = 1 Second).

Tag Returns or sets an expression that stores any extra data needed for the application. Unlike other properties, the value of the Tag property isn't used by Visual Basic; the application use this property for its own purpose. For example, when the timer Event finishes execution it may store a value in the Tag property, which may be used at the next occurrence of the Timer event.

Timer Event

The timer control has only one event called, appropriately, a Timer event. As already stated, this event takes place every time an interval elapses. The interval is determined by Interval property. To stop the Timer event from occurring you can set the timer's Enabled property to False at run time. Timer occurs when a preset interval (using Interval property) has elapsed.

Example

The following code shows how to change the shape of a Shape control at the interval of one second. It is assumed that the Timer1control's properties are set through Properties window as in below:

- Interval = 1000
- Tag = 1

```
Private Sub Timer1_Timer()  
    ' Changing to next Shape  
    Shape1.Shape = Timer1.Tag  
  
    ' If current shape is the last shape then...  
    ' Setting the first shape for the next display  
    If Timer1.Tag = 5 Then
```

```

    Timer1.Tag = 0
Else
    ' Increasing the Tag value to display the next shape
    ' at the next display
    Timer1.Tag = Timer1.Tag + 1
End If
End Sub

```

Timer Methods

The timer control does not support any methods

4.6.8 The List Box Control

A list box is an ideal way of presenting users with a list of data. Users can browse the data in the list box or select one or more items as the basis for further processing. The user can't edit the data in a list box directly- one way around that is to have a combo box instead; combo boxes are discussed next. When the list of data is too long for the list box then VB will add a vertical scroll bar.

List Box Properties

Many of the list box properties are shared by a combo box control, and some of them are essential for getting the best from the control:

Appearance	Enabled	HelpContextID	TabIndex
BackColor	Font	hWnd	TabStop
Columns	FontBold	Index	Tag
Container	FontItalic	Left	Text
DataField	FontName	List	ToolTipText
DataSource	FontSize	ListCount	Top
DisabledPicture	FontStrikethru	ListIndex	TopIndex
DragIcon	FontUnderline	Parent	Visible
Dragmode	ForeColor	SelCount	WhatsThisHelp
MultiSelect	Height	Selected	Width
Name	MousePointer	Sorted	
NewIndex	MouseIcon	Style	

List Sets the items for the list. To set the items click the down arrow in this property's value space in the Properties window. It will bring a drop list as in Figure 1.30. Type the items in it delimiting them by typing Ctrl+Enter.

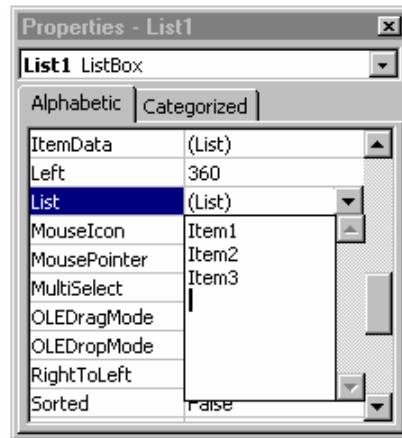


Figure 4.8 List property providing space to enter items.

This property can also be used to return a particular item. For example, to print the third item of a list, say List1, the following statement is used.

```
Print List1.List(2)
```

Note: List item's index starts from 0.

ListCount Returns the number of items in the list portion of a control. (Not available at design time).

ListIndex Returns the index of the currently selected item in the control. (Not available at design time).

Sorted Indicates whether the elements of a control are automatically sorted alphabetically.

Text Returns the selected item. (Not available at design time).

MultiSelect Indicates whether a user can make multiple selections. This can have one of the values given in the following table.

Value	Description
0-None	(Default) Multiple selection isn't allowed.
1-Simple	Simple multiple selection. A mouse click or pressing the SPACEBAR selects or deselects an item in the list.
2-Extended	Extended multiple selection. Pressing SHIFT and clicking the mouse or pressing SHIFT and one of the arrow keys

	extends the selection from the previously selected item to the current item. Pressing CTRL and clicking the mouse selects or deselects an item in the list.
--	---

Table 4.6 Values for MultiSelect property.

SelCount Returns number of selected items in the List Box control. If the MultiSelect property is set to value 0, this property will return either 0 (no selection) or 1 (only one item can be selected). If the MultiSelect property is set to value 1 or 2, the user may select more than one items in the List box control. In this case the SelCount property will return the value greater than 1.

Selected Returns the selection status of an item in the ListBox control. This property is an array of Boolean values with the same number of items as the List property. (Not available at design time). For example to know whether the second item is selected in a list, say List1, the following statements are used.

If List1.Selected(1) = True then

End if

To print the selected items of a List, say List1, loop (explained in detail in the next Chapter) can be used as in below:

```
For i = 1 to List1.ListCount
    If List1.Selected(i) = True Then
        Print List1.List(i)
    End If
Next
```

The above loop will execute for the total number of items of the list List1. Each time the current item is checked whether it is selected by using the Selected property. If so, it is printed.

Style Determines whether checkboxes are displayed inside the List box control. It can have the value either 0-Standard (default) or 1-Checkbox. If this property is set to Checkbox the list will appear as in Figure 1.31

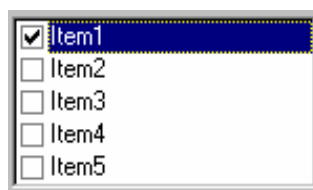


Figure 4.9 Check box style List box.

When an item is selected in this style, the item's check box will be checked.

Note: The Style property can have the value Checkbox, only when the MultiSelect property has the value 0-None.

List Box Methods

AddItem Adds an item to a ListBox at run time. For example, if the text in a text box entered by the user is to be added to a list box this method can be used to do the same.

Syntax

list.AddItem item, index

The *item* argument specifies the text to be added to the List box. The *index* argument (optional) specifies the position whether the new item is to be placed. For example, if the new item is to be inserted at the top of the list the *index* argument will have the value 0. If this argument is omitted, the new item will be placed at the bottom of the list.

Example

The following line is used to add the text, entered in the text box Text1, at the top of the list box List1.

```
List1.AddItem Text1.Text, 0
```

RemoveItem Removes an item from the List box control.

Syntax

list.RemoveItem index

The *index* argument specifies the item to be removed. For example, to remove the third item from the list box List1 the following line is used:

```
List1.RemoveItem 2
```

Clear Clears the content of the list box. For example, to clear the List1 the following line is used:

```
List1.Clear
```

Note: The properties List, ListCount, ListIndex, Sorted, and Text, and all the three methods are also available for the Combo Box control also.

4.6.9 The Month View Control

The MonthView control makes it easy for users to view and set date information via a calendar-like interface. Users can select a single date or a range of dates.

The control can be navigated using either the keyboard or mouse. Buttons at the top of the control are used to scroll months in and out of view.

In addition, the control has the ability to display up to 12 months at a time. This can be helpful to give users the ability to view date information around the date of interest. The Figure 1.32 shows a typical Month View control.

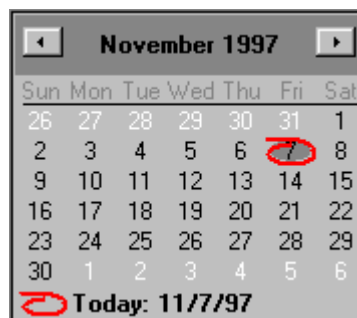


Figure 4.10 Month View control.

MonthView Properties

Property	Description
Day, Month, Week, and Year	Specifies the current Day, Month, Week and Year number.
DayofWeek	Specifies the current day of week.
ShowToday	(Boolean) Determines whether the current date is displayed at the bottom of the control.

MultiSelect	(Boolean) Determines whether multiple dates can be selected at once.
MaxSelCount	Determines the maximum number of contiguous days that can be selected at once.
SelStart and SelEnd	Return the start and end dates of a selection when the MultiSelect property is set to True.
MonthRows and MonthColumns	Specifies the number of months to be displayed horizontally and vertically. The total of the MonthRows and MonthColumns properties must be less than or equal to 12.
MonthBackColor, TitleBackColor	Various color attributes to customize the TitleForeColor, control's appearance, and TrailingForeColor.

MonthView Methods

The MonthView control has two unique methods that need to be noted, ComputeControlSize and HitTest.

The Compute ControlSize method enables you determine the height and width of a MonthView control, given its number of rows and columns. MonthView controls can get pretty large when displaying multiple months. By using the ComputeControlSize method before changing the MonthColumns and MonthRows properties, you can ensure that the control will fit on your form. The syntax of the ComputeControlSize method is:

ComputeControlSize (rows, columns, width, height)

You specify the rows and columns arguments before calling the method, then examine the variables passed as the width and height arguments. The rows and columns arguments are of type Integer, and width and height are of type Single.

The second unique method of the MonthView control is HitTest. HitTest is used to determine the part of the MonthView control that the mouse pointer is positioned on as well as the date that corresponds to the pointer's position (if any). The syntax for the HitTest method is:

ctrlpart = HitTest(x,y,date)

The value returned by the HitTest method(ctrlpart) specifies the part of the control in which the mouse cursor is positioned. This value corresponds to the *MonthViewHitTestAreas* Enum (see the table). The x and y arguments are the mouse pointer's coordinates, and the Date variable specified by the date argument will contain the date that is under the mouse pointer, if any.

The values of the month view hit test areas Enum

Enum Member	Value
mvwCalendarBack	0
mvwCalendarDate	1
mvwCalendarDateNext	2
mvwCalendarDatePrev	3
mvwCalendarDay	4
mvwCalendarWeekNum	5
mvwNoWhere	6
mvwTitleBack	7
mvwTitleBtnNext	8
mvwTitleBtnPrev	9
mvwTitleMonth	10
mvwTitleYear	11
mvwTitleTodayLink	12

MonthView Events

Event	Description
DateClick	When a date on the control is clicked.
SelChange	Occurs when the user selects a new date or range of dates.
GetDayBold	When the user wants to change the appearance of certain dates, such as making them bold.

Possible Uses

The possible uses of the MonthView control is given below:

- ❖ To present date information where a calendar representation of a date is easier to understand than that of a Label or Textbox control.

- ❖ To give users the ability to choose a date with the click of a mouse rather than typing a date value.
- ❖ To enable users to view multiple months as part of an advanced booking system such as those used by hotels and airlines.

4.6.10 The date time picker control

The DateTimePicker control displays date and/or time information and acts as the interface through which users can modify date and time information. The control's display consists of fields that are defined by the control's format string. When the DateTimePicker is dropped down, a MonthView calendar is displayed.

The control has two different modes:

- ❖ Dropdown Calendar mode (default) - enables the user to display a dropdown calendar that can be used to select a date.
- ❖ Time Format mode - enables the user to select a field in the date display (i.e. the month, day, year, etc.) and press the up/down arrow to the right of the control to set its value.

The Figure 1.33 shows these two different modes.

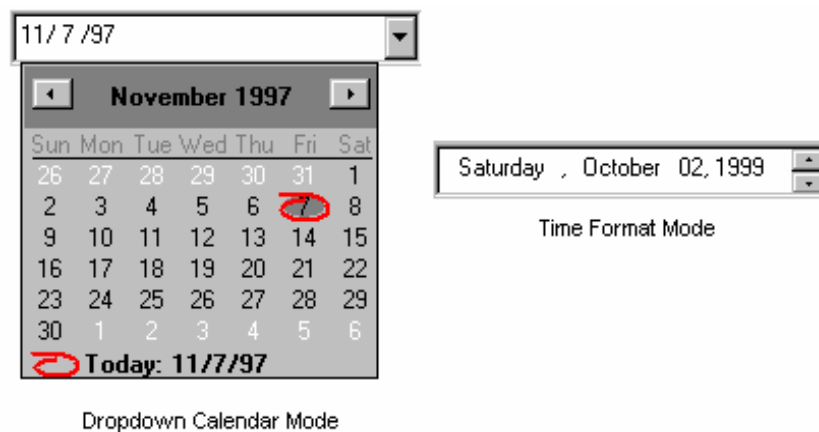


Figure 4.11 Different modes of Date time picker control.

Note : To display the Date time picker control in the Time Format Mode check the Updown check box in the general tab of its property page.

DTPicker Properties

This control also has the properties Value, Year, Month, Week, Day, DayOfWeek which are same as in the Monthview control. This control's other important properties are given below:

The MinDate and MaxDate properties

It determines the first and last date allowed by the calendar.

DTPicker Formats

It determines the format type used to display text on the control. It can have one of the values given in the following table.

Constant	Value	Description
DtpLongDate	0	Long Date Format (for example "Friday, Nov 14, 1972").
DtpShortDate	1	Short Date Format (for example "11/14/72").
DtpTime	2	Time Format (for example "5:31:47 PM").
DtpCustom	3	Custom format.

Table 4.5 Values for the Format property.

If the Format Property is set to dtpCustom the format set in the CustomFormat property (explained next) is used.

DTPicker CustomFormat

It specifies the format used by the control when displaying date/time information. The string fragment given in the following table can be used to specify the custom format.

String Fragment	Description
D	The one- or two-digit day.
Dd	The two-digit day. Single digit day values are preceded by a zero.
Ddd	The three-character weekday abbreviation.
Dddd	The full weekday name.
H	The one- or two-digit hour in 12-hour format.
Hh	The two-digit hour in 12-hour format. Single digit values are preceded by a zero.
H	The one- or two-digit hour in 24-hour format.
HH	The two-digit hour in 24-hour format. Single digit values are preceded by a zero.
M	The one- or two-digit minute.

Mm	The two-digit minute. Single digit values are preceded by a zero.
M	The one- or two-digit month number.
MM	The two-digit month number. Single digit values are preceded by a zero.
MMM	The three-character month abbreviation.
MMMM	The full month name.
S	The one- or two- digit seconds.
Ss	The two-digit seconds. Single digit values are preceded by a zero.
T	The one-letter AM/PM abbreviation (that is, "AM" is displayed as "A").
Tt	The two-letter AM/PM abbreviation (that is, "AM" is displayed as "AM").
X	A callback field. The control still uses the other valid format characters, and queries the owner to fill in the "X" portion. The owner must be ready to respond to events that request information about how to fill in these fields. Multiple 'X' characters can be used in series to signify unique callback fields.
Y	The one-digit year (that is, 1997 would be displayed as "7").
Yy	The last two digits of the year (that is, 1997 would be displayed as "97").
Yyy	The full year (that is, 1997 would be displayed as "1997").

Table 4.8 String fragments for CustomFormat property.

Examples

To display the date in the format 12-Nov-1976 the following format string used.

`"dd-MMM-yyyy"`

Body text can also be added to the format string. For example, if string has to appear as "Today is: 05:30:31 Friday Nov 14, 1997", then the required format string is

`"Today is: 'hh':'m':'s ddddMMMdd', 'yyy'".`

Body text must be enclosed in single quotes.

4.7 Short Summary

- ☞ Controls have properties that define aspects of their appearance, such as position, size, and colour, and aspects of their behaviour, such as their response to user input.
- ☞ List box and ComboBox controls present a set of choices that are displayed vertically in a single column.
- ☞ A combobox control combines the features of TextBox and a listBox. This control enables the user to select either by typing text into the ComboBox or by selecting an item from its list.
- ☞ A Control Array is a group of controls that share the same name and type. They also share the same event procedures.

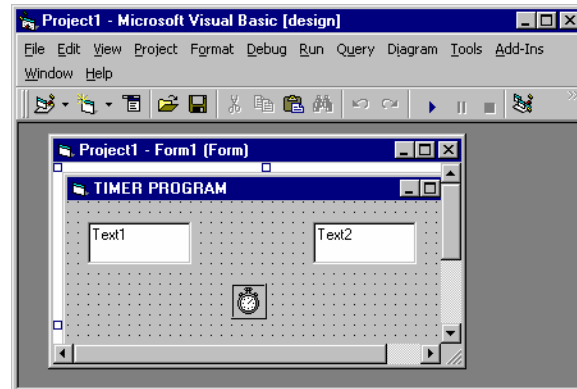
4.8 Brain Storm

1. Do all controls receive the focus? If not, specify those controls that don't and the reason(s)?
2. Do we Dbclick event for a command button?
3. How can you make difference choices using option buttons? Why?
4. Can we have more than one command button as cancel button on a single form
5. How do you make a text box control a read - only control?
6. Specify few significant differences between an option button and a check box?
7. Can the user stretch the image to the size of the picture box?
8. List out the difference between Timer Control and Date Time Picker Control?
9. How can you place a Month View Control on a form?
10. Specify few differences between the picture box and image box?

Lab Unit - 4 (2 Real Time Hrs)

1

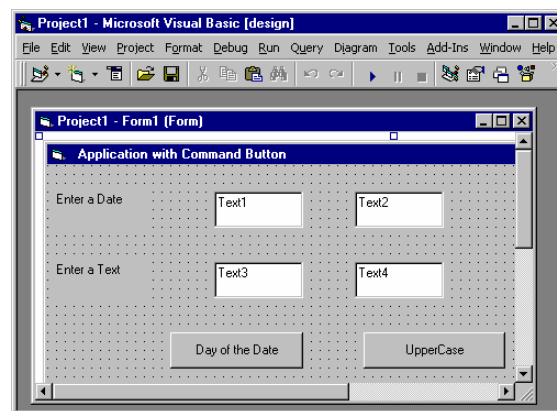
1. Open a new standard EXE project
2. Design your form as shown below



3. Current Time should be displayed on the Text1
4. Current Date should be displayed on the Text2

2

1. Open a new standard EXE project



2. Design your form as shown below
3. In the Text1 Box enter a date
4. In the Text3Box enter a text
5. If you click the Command Button1, Day of the Date should be displayed in the Text2 Box
6. If you click the Command Button2, UpperCase of the text should be displayed in the Text4 Box

3

1. Open a new Standard EXE project

2. Design your form as shown below

The screenshot shows a Visual Basic form window titled "Employee Information" with a menu bar (File, Edit, View, Project, Format, Debug, Run, Query, Diagram, Tools, Add-Ins, Window, Help). The form has a title bar "ABC and COMPANY". It contains the following controls:

- Emp NO: Text1
- Ename: Text2
- Salary: Text3
- Sex: Radio buttons for Male and Female, followed by Text4
- Allowances: A group box containing checkboxes for HRA Label5 and DA Label6
- Buttons: Ok, Cancel, and Exit
- Label7: A label at the bottom of the form.

3. Whenever the Textbox "TEXT1" gets the focus it must be cleared.
4. The Textbox "TEXT1" should accept only numbers.
5. The user must not be allowed to go out of the Text box "TEXT1" when it is empty
6. The Textbox "TEXT2" must accept only alphabets (Upper case)
7. The Textbox "TEXT3" should accept only a maximum of 8 digits and only numbers.
8. "TEXT4" is to input a special allowance to the female candidates. So the user must be allowed to enter values in it only if the candidate is a female candidate
9. If any of the allowances is applicable to the candidate it must be displayed in the respective label boxes.
 - HRA - 2500.00
 - DA - 3000.00
10. When the user clicks the "OK" button a confirmation message must be displayed in the label box "Label7"
11. When the user clicks the "Cancel" button a cancellation message must be displayed in the above mentioned label control and all the textboxes must be cleared.
12. When the user clicks the "Exit" button stop the execution of the program.

Lecture - 5

Control Arrays

Objectives

In this lecture you will learn the following

- ❖ About Control arrays
- ❖ Know how to create a control array at design time
- ❖ Know how to create a control array at run time

Lecture unit - 5

- 5.1 Snap Shot
- 5.2 Creating Control Arrays at Design Time
- 5.3 Creating Control Arrays at Run Time
- 5.4 Short Summary
- 5.5 Brain Storm

Lab unit 5 (2 Real Time Hrs)

5.1 Snap Shot

In the previous unit we have seen about the types of statements , controls and how we can placed it in the form. This unit helps to understand the concept of control arrays, why we are using Controls arrays and how the control arrays can be created during at run and design time.

5.1.1 Control Arrays

A control array is an array of controls, in the same way that an integer is an array of integers. A group of controls that share common names, types, and event procedures. Each control has a unique index. When a control in the array recognizes an event, it calls the event procedure for the group and passes the index as an argument, allowing your code to determine which control recognized the event. When you add a second control of the same type of the form, visual Basic presents you a message asking 'Do you want to create a control array' .Choose Yes to create a control array.

By grouping similar controls into an array(much like arrays that hold strings or some other data type), you can down on the amount of code in your program. Because all of the controls in the array share event handlers (for example, there is only one Click event for an array of Command Button controls no matter how many controls are in the array), you do not have to duplicate event code for each individual control.

Combining dynamic controls with control arrays offers the best of both worlds. Dynamic controls enable your program to do a little jig in step with your end user's needs: the control array enables you to easily manage the pool of dynamically created controls.

There are many benefits to using control arrays instead of several individual controls. First, control arrays use fewer resources than individual controls. Second, controls arrays share common events. For example, if you have an array of command buttons, the same Click event procedure is called no matter which button is pressed.

If you start with a text box named Text1 and elect to create a control array, Visual Basic assignsText1 an index number of 0 and creates a new member of the Text1 array, with an index number of 0 and creates a new member of the Text1 array, with an index number of 1. If you make additional copies of Text1 or Text2, their index properties are set to 3,4 and so on.

Examples - Showing how to handle a click event for control arrays

```
Private Sub cmdButtonArray_Click (Index As Integer)
```

```
    Select Case Index
        Case 0: txtIndex.Text = 1
        Case 1: txtIndex.Text = 2
        Case 2: txtIndex.Text = 3
        Case 3: txtIndex.Text = 4
        Case 4: txtIndex.Text = 5
        Case 5: txtIndex.Text = 6
    End Select
```

```
End Sub
```

Notice in this code fragment that the Index property of the control array member that received the event is passed as an argument to the event. However, instead of using index numbers directly in the Select statement, it is more maintainable to assign these to constants. That way, if for any reason control indexes change, you only need to change the constant declaration to fix all the code referencing those indexes.

Working with Control Arrays

A control array is a group of controls that share the same name and type. They also share the same event procedures. A control array has at least one element and can grow to as many elements as your system resources and memory permit; its size also depends on how much memory and Windows resources each control requires. The maximum index you can use in a control array is 32767. Elements of the same control array have their own property settings. Common uses for control arrays include menu controls and option button groupings.

Note: Visual Basic includes the ability to dynamically add unreferenced controls to the Controls collection at run time.

Why Use Control Arrays?

Adding controls with control arrays uses fewer resources than simply adding multiple controls of the same type to a form at design time. Control arrays are also useful if you want several controls to share code. For example, if three option buttons are created as a control array, the same code is executed regardless of which button was clicked.

If you want to create a new instance of a control at run time, that control must be a member of a control array. With a control array, each new element inherits the common event procedures of the array.

Using the control array mechanism, each new control inherits the common event procedures already written for the array. For example, if your form has several text boxes that each receive a date value, a control array can be set up so that all of the text boxes share the same validation code.

5.2 Creating a Control Array at Design Time

There are three ways to create a control array at design time:

- Assign the same name to more than one control.
- Copy an existing control and then paste it onto the form.
- Set the control's Index property to a value that is not Null.

Note - You must create menu control arrays in the Menu Editor.

To add a control array element by changing its name

1. Draw the controls you want to be in the control array. (The controls must all be of the same type.) Decide which control will become the first element in the array.
2. Select one of the controls and change its Name setting to the Name setting for the first element in the array.
3. When you type an existing name for a control in the array, Visual Basic displays a dialog box asking you to confirm that you want to create a control array. Choose Yes to confirm the action.

For example, if the name of the first element in a control array is cmdCtlArr, you would choose a command button to add to the array and then set its name to cmdCtlArr. The message "You already have a control named 'cmdCtlArr.' Do you want to create a control array?" is displayed. Choose Yes to confirm the operation.

Controls added this way share only their Name property and control type; all other properties remain the same as when the control was originally drawn.

To add a control array element by copying an existing control

1. Draw a control in the control array.
2. While the control has the focus, choose Copy from the Edit menu.
3. From the Edit menu, choose Paste. Visual Basic displays a dialog box asking you to confirm that you want to create a control array. Choose Yes to confirm the action.

This control is assigned an index value of 1. The first control you drew has a value of 0. The index value of each new array element corresponds to the order in which the element was added to the control array. When controls are added this way, most of the visual properties, such as height, width, and color, are copied from the first control in the control array to the new controls.

5.3 Creating control array at Run Time

You can add and remove controls in a control array at run time using the Load and Unload statements. However, the control to be added must be an element of an existing control array. You must have created a control at design time with the Index property set, in most cases, to 0. Then, at run time, use this syntax:

Load object(index%)

Unload object(index%)

Argument,Description,objectName of the control to add to or delete from the control array.index%The control's index value in the array.

When you load a new element of a control array, most of the property settings are copied from the lowest existing element in the array – in this example, the element with the 0 index value. The Visible, Index, and TabIndex property settings are not automatically copied to new elements of a control array, so to make the newly added control visible, you must set its Visible property to True.

<p>Note Visual Basic generates an error if you attempt to use the Load statement with an index number already in use in the array.</p>

You can use the Unload statement to remove any control created with Load. However, you cannot use Unload to remove controls created at design time, regardless of whether or not they are part of a control array.

Adding and Deleting Controls in a Control Array

The control array example demonstrates how controls – in this case, option buttons – are added and deleted at run time. The example allows the user to add option buttons that change the background color of a picture box.

Start with a form, and then draw a picture box, a label, two option buttons, and three command buttons, as shown in Figure 5.1

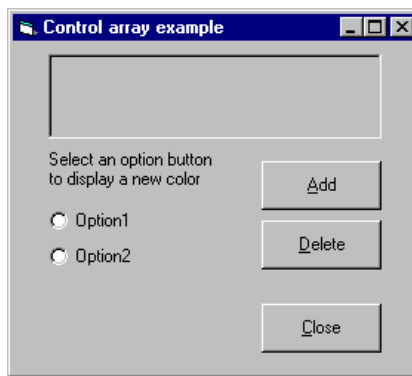


Figure 5.1 – Adding controls at run time

The following table lists the property settings for the objects in the application.

Object	Property	Setting
Form	Caption	Control Array Example
Picture box	Name	picDisplay
Label	Caption	Select an option button to display a new color
Option1	Name Index	optButton0
Option2	Name Index	optButton1
First command button	Name Caption	cmdAdd &Add
Second command button	NameCaption	cmdDelete&Delete
Thir d command button	Name Caption	cmdClose&Close

Events in the Control Array Application

Next, you need to add the event procedures for the option buttons and command buttons. Start by adding the form declaration:

```
Dim MaxId As Integer
```

The Click event procedure is shared by all the option buttons:

```
Private Sub optButton_Click (Index As Integer)
    picDisplay.BackColor = QBColor(Index + 1)
End Sub
```

New option buttons are added by the Click event procedure for the Add command button. In this example, the code checks that no more than ten option buttons are loaded before the Load statement is executed. Once a control is loaded, its Visible property must be set to True.

```
Private Sub cmdAdd_Click ()
    If MaxId = 0 Then MaxId = 1 ' Set total option
        ' buttons.
    If MaxId > 8 Then Exit Sub ' Only ten buttons
        ' allowed.
    MaxId = MaxId + 1 ' Increment button count.
    Load optButton(MaxId) ' Create new button.
    optButton(0).SetFocus ' Reset button selection.
    ' Set new button under previous button.
    optButton(MaxId).Top = optButton(MaxId - 1)._Top + 400
    optButton(MaxId).Visible = True ' Display new
        ' button.
    optButton(MaxId).Caption = "Option" & MaxId + 1
End Sub
```

Option buttons are removed by the Click event procedure for the Delete command button:

```
Private Sub cmdDelete_Click ()
    If MaxId <= 1 Then Exit Sub ' Keep first two
        ' buttons.
    Unload optButton(MaxId) ' Delete last button.
    MaxId = MaxId - 1 ' Decrement button count.
    optButton(0).SetFocus ' Reset button selection.
```

End Sub

The Close button Click event ends the application:

```
Private Sub cmdClose_Click ()
```

```
    Unload Me
```

```
End Sub
```

5.4 Short Summary

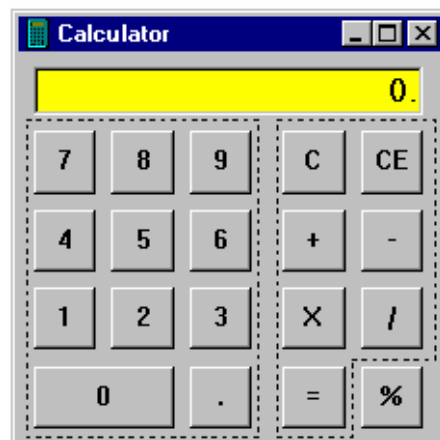
- ☞ Control arrays is a group of controls that share the same name and type.
- ☞ Control arrays share event procedures
- ☞ Control arrays can be created during design time by assigning the same name in the name property for more than one control
- ☞ During Design time set the index property to a value is not null
- ☞ Control arrays can be create at run time using the statements [Load object (index %) and Unload object (index %)]

5.5 Brain Storm

1. What are the salient features of the control arrays?
2. When a Control array can be used?
3. What is the need of control arrays
4. How a control arrays can be created at run time

Lab Unit - 5 (2 Real Time Hrs)

1. Open a new Standard EXE project
2. Design your form as shown below (19 Command Button & A Text box) and perform a Calculator Project , which can able to work as like as Windows Calculator.



Lecture - 6

Procedures

Objectives

In this lecture you will learn the following

- ❖ About Procedures
- ❖ Knowing about the features of General, Event procedures
- ❖ Understand how to create a Function procedure that accepts arguments and returns a value
- ❖ Scope of the Procedures.

Lecture Unit - 6

- 6.1 Snap Shot
- 6.2 General procedures
- 6.3 Event procedures
- 6.4 Creating and calling functions
- 6.5 Scope of procedure
- 6.6 Short Summary
- 6.7 Brain Storm

Lab unit 6 (2 Real Time Hrs)

6.1 Snap Shot

When we write a lengthy program some times we may have to write a set of statements repeatedly. It will be a time consuming one. So, to avoid that we can write the code and keep in a separate file and when ever we need to execute the statements we can call it. The advantages of using procedures in programming are (a) It is easier to debug a program with a procedures, which breaks a program into discrete logical limits. (b) Procedures used in one program can act as building blocks for other programs with slight modifications.

6.2 General Procedures

A general procedure tells the application how to perform a specific task. Once a general procedure is defined, it must be specifically invoked by the application. By contrast, an event procedure remains idle until called upon to respond to events caused by the user or triggered by the system.

Why create general procedures? One reason is that several different event procedures might need the same actions performed. A good programming strategy is to put common statements in a separate procedure (a general procedure) and have your event procedures call it. This eliminates the need to duplicate code and also makes the application easier to maintain. For example, the VCR sample application uses a general procedure called by the click events for several different scroll buttons. Figure 2.3 illustrates the use of a general procedure. Code in the Click events calls the ButtonManager Sub procedure, which runs its own code, and then returns control to the Click event procedure.

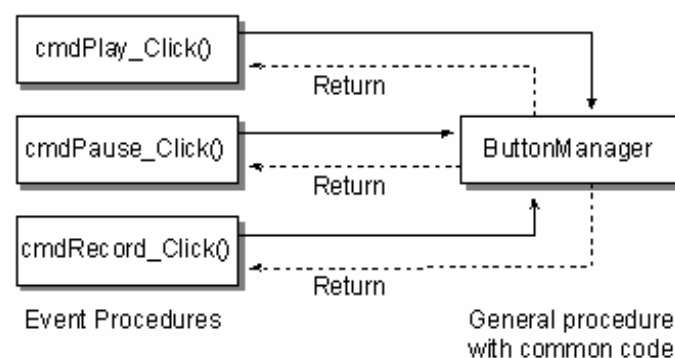


Figure 2.3 General procedures are called by event procedures

6.3 Event Procedures

When an object in Visual Basic recognizes that an event has occurred, it automatically invokes the event procedure using the name corresponding to the event. Because the name establishes an association between the object and the code, event procedures are said to be attached to forms and controls.

- An event procedure for a control combines the control's actual name (specified in the Name property), an underscore (_), and the event name. For instance, if you want a command button named cmdPlay to invoke an event procedure when it is clicked, use the procedure cmdPlay_Click.
- An event procedure for a form combines the word "Form," an underscore, and the event name. If you want a form to invoke an event procedure when it is clicked, use the procedure Form_Click. (Like controls, forms do have unique names, but they are not used in the names of event procedures.) If you are using the MDI form, the event procedure combines the word "MDIForm," an underscore, and the event name, as in MDIForm_Load.

All event procedures use the same general syntax.

Syntax for a control event

Syntax for a form event

```
Private Sub controlname_eventname (arguments )  
    statements  
End Sub
```

```
Private Sub Form_eventname (arguments)  
    statements  
End Sub
```

Although you can write event procedures from scratch, it's easier to use the code procedures provided by Visual Basic, which automatically include the correct procedure names. You can select a template in the Code Editor window by selecting an object from the Object box and then selecting a procedure from the Procedure box.

It's also a good idea to set the Name property of your controls before you start writing event procedures for them. If you change the name of a control after attaching a procedure to it, you must also change the name of the procedure to match the new name of the control. Otherwise, Visual Basic won't be able to match the control to the procedure. When a procedure name does not match a control name, it becomes a general procedure.

6.4 Creating and Calling Functions

You've now met two types of Sub procedures. The first type (Private by default) is the event Sub procedure behind forms and controls. These event procedures are already defined for you. The second type the ones you add yourself are called general Sub procedures. There are also Function procedures. These you must create yourself, and they can be Private or Public. Function procedures return a value to the procedures that call them. The best way to understand Function procedures is to try one out. Here's one that calculates the cube root of a number:

1. Open a form's code window by double-clicking it in the Form Designer
2. Create the template by typing the following on a blank line in a form's Code window and pressing Enter:

```
Public Function CubeRoot
```

This Creates the Following code:

```
Public Function CubeRoot()
```

```
End Function
```

3. Now alter the template as follows

```
Public Function CubeRoot(x As Double) As Double
```

```
    If x=0 Then
```

```
        CubeRoot = 0
```

```
        Exit Function
```

```
    End If
```

```
    CubeRoot = 10 ( (Log(Abs(x))/Log(10))/3)
```

```
    If x<0 Then
```

```
        CubeRoot = -CubeRoot
```

```
    End If
```

```
End Function
```

The math involved here isn't important. But a few other things are:

- The Function ends with an End Function rather than an End Sub
- You can jump out of a Function with Exit Function.
- The Function must return a value- the line Cube Root = 0 is one of the three possible lines that do that.
- You can specify the type of value returned- here it's a Double (a numeric variable that can handle very large and small values as well as decimals) by adding the statements As followed by the data type of the returned value.

4. To call this function, you must assign its return value to another variable. Try the following code in a click event for a form

```
Dim Y As Double
    Y= CubeRoot(27)
Print Y
```

5. Now run the application and click the command button.

This code prints the return value straight onto the form

Normally, you may want to assign the return value to a control on the form, such as a text box control

```
txtText1.Text = Y
```

The result is

Alternatively, you can have just the line

```
TxtText1.Text = CubeRoot(27)
```

Here the return value is being assigned directly to a control

Note: When you call a Function procedure, you must enclose the parameter(s) in parentheses. This contrasts with calling a Sub procedure.

6.5 Scope of procedures

Usually the code in a procedure needs some information about the state of the program to do its job. This information consists of variables passed to the procedure when it is called. When a variable is passed to a procedure, it is called an argument.

Argument Data Types

The arguments for procedures you write have the Variant data type by default. However, you can declare other data types for arguments. For example, the following function accepts a string and an integer:

```
Function WhatsForLunch(WeekDay As String, Hour _
    As Integer) As String
```

```

' Returns a lunch menu based on the day and time.
If WeekDay = "Friday" then
    WhatsForLunch = "Fish"
Else
    WhatsForLunch = "Chicken"
End If
If Hour > 4 Then WhatsForLunch = "Too late"
End Function

```

6.5.1 Passing Arguments By Value

Only a copy of a variable is passed when an argument is passed by value. If the procedure changes the value, the change affects only the copy and not the variable itself. Use the `ByVal` keyword to indicate an argument passed by value.

For example:

```

Sub PostAccounts(ByVal intAcctNum as Integer)
.
. ' Place statements here.
.
End Sub

```

6.5.2 Passing Arguments By Reference

Passing arguments by reference gives the procedure access to the actual variable contents in its memory address location. As a result, the variable's value can be permanently changed by the procedure to which it is passed. Passing by reference is the default in Visual Basic.

If you specify a data type for an argument passed by reference, you must pass a value of that type for the argument. You can work around this by passing an expression, rather than a data type, for an argument. Visual Basic evaluates an expression and passes it as the required type if it can.

The simplest way to turn a variable into an expression is to enclose it in parentheses. For example, to pass a variable declared as an integer to a procedure expecting a string as an argument, you would do the following:

```

Sub CallingProcedure()
    Dim intX As Integer
    intX = 12 * 3

```

```
    Foo(intX)
End Sub

Sub Foo(Bar As String)
    MsgBox Bar 'The value of Bar is the string "36".
End Sub
```

6.5.3 Using Optional Arguments

You can specify arguments to a procedure as optional by placing the `Optional` keyword in the argument list. If you specify an optional argument, all subsequent arguments in the argument list must also be optional and declared with the `Optional` keyword. The two pieces of sample code below assume there is a form with a command button and list box.

For example, this code provides all optional arguments:

```
Dim strName As String
Dim strAddress As String

Sub ListText(Optional x As String, Optional y _
As String)
    List1.AddItem x
    List1.AddItem y
End Sub

Private Sub Command1_Click()
    strName = "yourname"
    strAddress = 12345 ' Both arguments are provided.
    Call ListText(strName, strAddress)
End Sub
```

This code, however, does not provide all optional arguments:

```
Dim strName As String
Dim varAddress As Variant

Sub ListText(x As String, Optional y As Variant)
    List1.AddItem x
    If Not IsMissing(y) Then
        List1.AddItem y
    End If
```

```
End Sub

Private Sub Command1_Click()
    strName = "yourname" ' Second argument is not
                        ' provided.
    Call ListText(strName)
End Sub
```

In the case where an optional argument is not provided, the argument is actually assigned as a variant with the value of Empty. The example above shows how to test for missing optional arguments using the IsMissing function.

Providing a Default for an Optional Argument

It's also possible to specify a default value for an optional argument. The following example returns a default value if the optional argument isn't passed to the function procedure:

```
Sub ListText(x As String, Optional y As _
Integer = 12345)
    List1.AddItem x
    List1.AddItem y
End Sub

Private Sub Command1_Click()
    strName = "yourname" ' Second argument is not
                        ' provided.
    Call ListText(strName) ' Adds "yourname" and
                        ' "12345".
End Sub
```

Using an Indefinite Number of Arguments

Generally, the number of arguments in the procedure call must be the same as in the procedure specification. Using the ParamArray keyword allows you to specify that a procedure will accept an arbitrary number of arguments. This allows you to write functions like Sum:

```
Dim x As Integer
Dim y As Integer
Dim intSum As Integer
```

```
Sub Sum(ParamArray intNums())
  For Each x In intNums
    y = y + x
  Next x
  intSum = y
End Sub

Private Sub Command1_Click()
  Sum 1, 3, 5, 7, 8
  List1.AddItem intSum
End Sub
```

6.5.4 Using Variable Number of Arguments

The arguments for procedures you write have the Variant data type by default. However, you can declare other data types for arguments. For example, the following function accepts a string and an integer:

```
Function WhatsForLunch(WeekDay As String, Hour _
  As Integer) As String
  ' Returns a lunch menu based on the day and time.
  If WeekDay = "Friday" then
    WhatsForLunch = "Fish"
  Else
    WhatsForLunch = "Chicken"
  End If
  If Hour > 4 Then WhatsForLunch = "Too late"
End Function
```

6.6 Short Summary

- ☞ Event procedures acquire the declaration as Private default
- ☞ A general procedure is declared when several event procedures perform the same actions.
- ☞ Visual Basic uses building blocks such as Variables, Datatypes, Procedures, Functions, and
- ☞ Control Structures in its programming environment.

6.7 Brain Storm

1. What is a Procedure?
2. What are the difference between General procedure and Event procedures.
3. What is a Function?
4. What are the difference between Functions and Procedures?

Lab Unit - 6 (2 Real Time Hrs)

1. Write event procedures to convert a text from lowercase to uppercase.
2. Write an event procedure to find the sum of numbers from 1 to selected value.
Use a horizontal scroll bar to set the maximum value.
3. Write a function procedure to find Simple interest.
4. Write s function procedure to convert the temperature in Celsius into Fahrenheit.

Lecture - 7

Debugging

Objectives

In this lecture you will learn the following

- ❖ About Debugging
- ❖ Handling Windows
- ❖ Know to assign values to the variables

Lecture unit 7

- 7.1 Snap Shot
- 7.2 Debugging Tools in Visual Basic
- 7.3 Using Debugging Windows
- 7.4 Assigning Values to variables and properties
- 7.5 Modules
- 7.6 Enum (Enumerated Constant)
- 7.7 Short Summary
- 7.8 Brain storm

Lab Unit 7 (2 Real Time Hrs)

7.1 Snap Shot

When errors occur in the application code, it is referred as bugs. Minor bugs – for example, execution of some statements may never occur due the fault in the conditional statements – can be frustrating or inconvenient. More severe bugs can cause an application to stop responding to commands, possibly requiring the user to restart the application, and losing whatever work hasn't been saved.

The process of locating and fixing bugs in the application is known as *debugging*. Visual Basic provides several tools to help to analyze the operation of the application. These debugging tools are particularly useful in locating the source of the bugs.

The following sections show how to use the debugging tools included in Visual Basic and explains how to handle *run-time errors*.

Design-Time, Run-Time, and Break Modes

Understanding the three modes of operations of the Visual Basic will ease the debugging process. Applications are built in Design-time mode and run in run-time mode. This chapter introduces *break mode*, which suspends the execution of the program so that data can be examined and altered.

Identifying the Current Mode

The Visual Basic title bar always shows the current mode. The following Figures 7.1 show the title bar for design time, run time, and break mode.

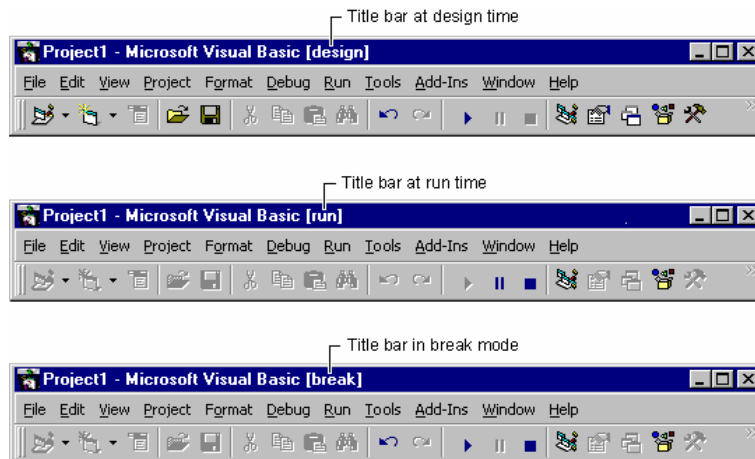


Figure 7.1 Title bars of different modes.

While an application is running, a situation may occur to switch to the suspend mode (break mode). For example, the execution of the application may be

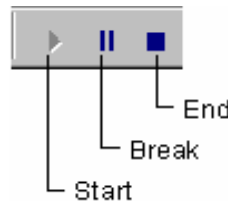
imprisoned in an indefinite loop. In this case the running application might be switched to the break mode. This is done:

- ❖ By choosing the Break from Run menu.
- ❖ By clicking the Break button.
- ❖ By pressing the CTRL+BREAK.

In the suspend mode, it is possible to view and edit code (choosing Code from the View menu, or pressing F7), examine or modify data, restart the application, end execution, or continue execution from the same point.

Using the Toolbar to Change Modes

The toolbar provides three buttons that lets to change quickly from one mode to another. These buttons appear as in Figure 7.2.



Whether any of these buttons is available depends on whether Visual Basic is in run-time mode, design-time mode, or break mode. The following table lists the buttons available for different modes.

Mode	Toolbar buttons available
Design time	Start
Run time	Break, End
Break	Continue, End (in break mode, the Start button becomes the Continue button)

Using the Break Mode

As said earlier, the Break mode enables to modify the code and to continue the execution of an application. For example, if a run time error occurs, while an application is running, the Visual Basic displays a dialog box with error message. Clicking the Debug button of that dialog box will cause the

application to switch to the break mode. Here, the cursor will wait with highlighted background at the statement that caused the error (Figure 7.3). Now that statement can be corrected and the execution can be continued.

The Frequent run time errors include misspelled names and mismatched properties or methods with objects – for example, trying to use the Clear method on a text box, or assigning a Text to an integer variable. The following example shows a typical situation.

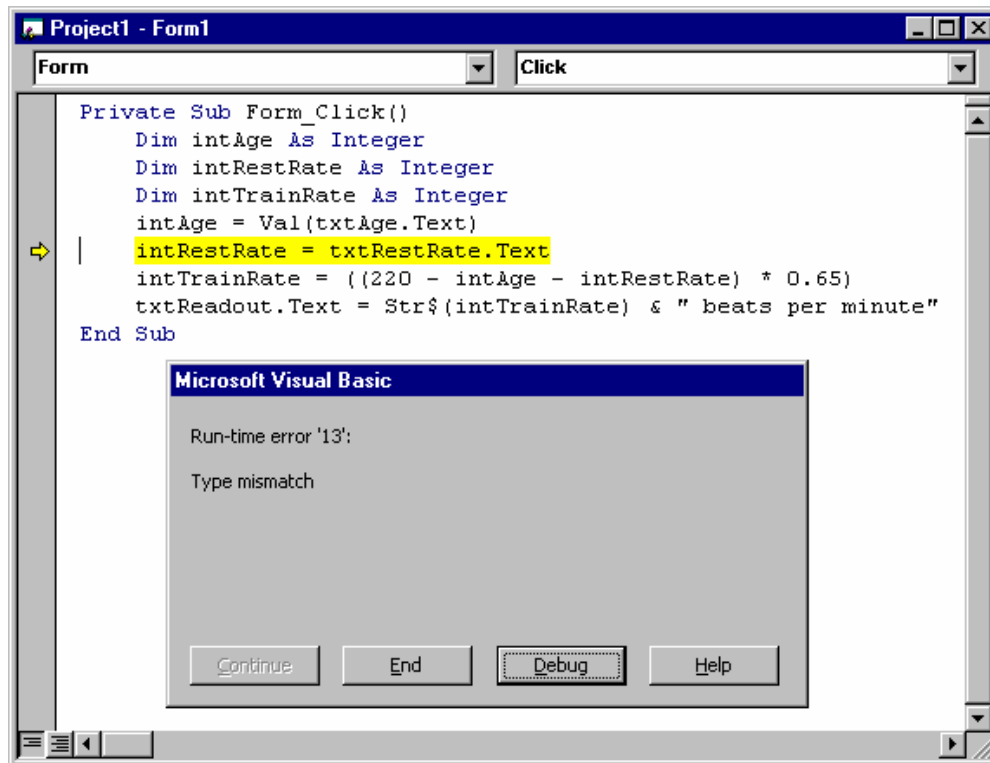


Figure 7.3 Cursor with highlighted background at the statement that caused the error.

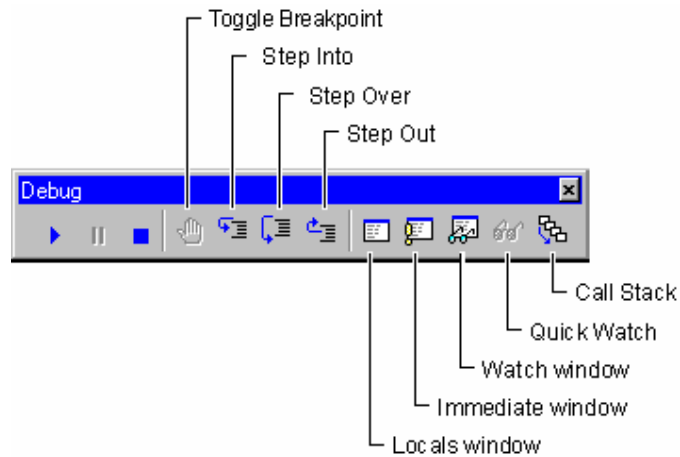
In this example the value of the txtRestRate text box, which is a string, is attempted for the assignment to the integer variable intRestRate. So Visual Basic displays the Error Dialog box, with the message Runtime error 13 – Type mismatch. When the mode is switched to the break mode, by clicking the Debug button, the current line that caused the error, can be corrected as

`intRestRate = Cint(txtRestRate.Text)`, to convert the string value to the integer value before the assignment and the execution can be continued by simply clicking the continue button on the toolbar.

Note: Though changes can be made in the Break Mode, some changes (most commonly, changing variable declarations or adding new variables or procedures) require to restart the application.

7.2 Debugging Toolbar In Visual Basic

Among its many debugging tools, Visual Basic provides several buttons on the optional Debug toolbar that are very helpful. Figure 7.4 shows these tools. Right clicking on the Visual Basic



toolbar and selecting the Debug option will display the Debug toolbar.

Figure 7.4 The Debugging Toolbar.

This section discusses the components in the second block of the Debug toolbar such as Toggle break point, Step into etc. Third block components are discussed in the next section.

Toggle Break Point

The Toggle Breakpoint option simply toggles a break point on and off at the current line. If a break point is set for a line, then that line will appear with red background and a red circle in the code window's left margin as in Figure 7.5

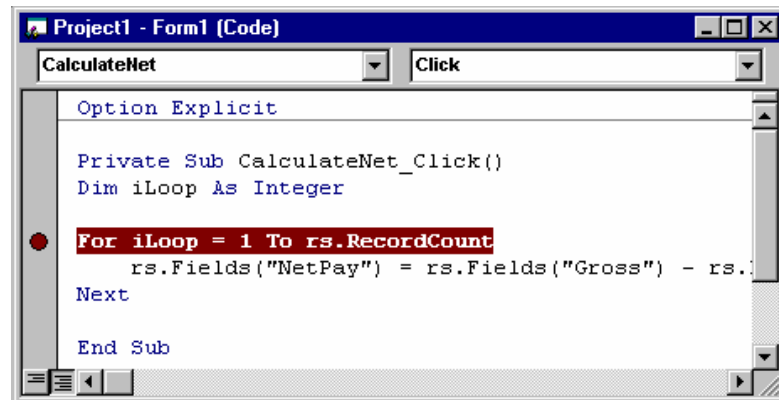


Figure 7.5 Line with a break point.

If a breakpoint is set for a line, then the application will switch automatically to break mode, when the execution reaches that line. This enables to trace the execution of the code.

To set a break point

- ❖ Place the cursor in the line where the break point is to be set.
 - ❖ Choose Break point option from Debug menu or Press F9
- Or ---

Simply click its toolbar button.

Note: Breakpoints can't be set on non-executable lines such comments, declaration statements or blank lines.

Step Into executes the current line of code. However, If the statement is a call to a procedure, the next statement displayed is the first statement in the procedure.

Keyboard shortcut: F8.

Step Over executes each line of code in the current procedure, but steps over any subsequent procedure that is called. Even though the cursor does not stop anywhere within subsequent procedures, they are actually executed.

Keyboard shortcut: SHIFT+F8.

Step Out executes all lines remaining in the current procedure and stops executing on the line after the line that called the current procedure.

Keyboard shortcut: CTRL+SHIFT+F8.

7.3 Using The Debugging Windows

Analyzing the values of variables and properties, while the application is running will be useful for debugging. Because some times the problem may be due to the assignment of incorrect values to the variables and properties. Monitoring the data will enable to detect, why those incorrect values are assigned to the variables or properties.

The debugging windows help to monitor the values of variables, properties and expressions while the application is running. The debugging windows are the Immediate window, the Watch window, the Locals window, the quick watch window and the Call stack window. They can be activated either by choosing their respective menu option from the view menu or icons from the debug toolbar.

Immediate Window

While experimenting with an application, it may be desirable to execute individual procedures, evaluate expressions, or assign new values to variables or properties. The immediate window helps to accomplish these tasks. Its usage is explained in detail in the following sections.

Printing information in the immediate window

There are two ways to print to the immediate window

- ❖ By including `Debug.Print` statements in the application code.
- ❖ By Entering Print methods directly in the Immediate window.

These printing technique gives feedback about the application performance in a separate window (Immediate window), without interfering with output that a user sees.

Printing from Application Code

The print method with `Debug` object prefix sends output to the Immediate window.

```
Debug.Print [items][:]
```

For example, the following statement prints the value of `Salary` to the Immediate window every time it is executed:

```
Debug.Print "Salary = "; Salary
```

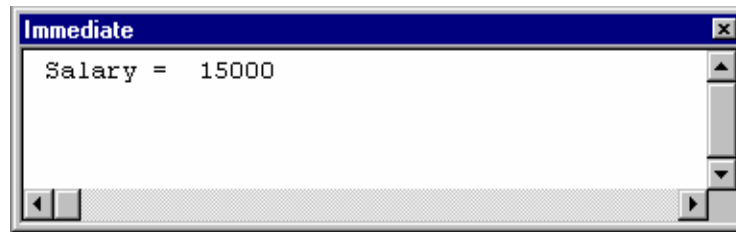


Figure 7.6 Result of Debug.Print statement in the Immediate window.

This technique works best when there is a particular place in the application code at which the variable (in this case, Salary) is known to change. For example, the previous statement might be put in a loop that repeatedly alters Salary.

Note: When the application is compiled into an .exe file, Debug.Print statements are not included in the .exe file.

Printing from Within the Immediate Window

In the break mode the print statements can directly be used in the Immediate window to examine the data.

To use the Immediate window in such a way:

- ❖ Click the Immediate window (if visible).

--or--

From the View menu, choose Immediate Window.

Once the Immediate window got focus, the print method can be used without the debug object.

- ❖ Type a print statement into the Immediate window, and then press ENTER.

The Immediate window responds by carrying out the statement, as shown as in the Figure 7.7

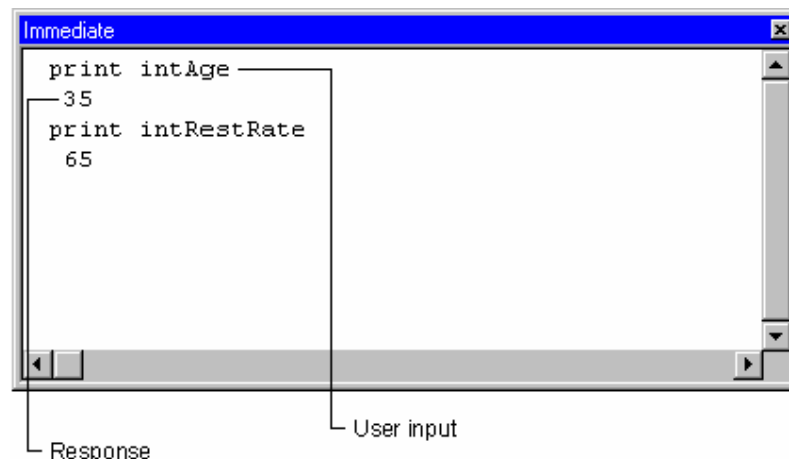


Figure 7.7 Using the Print method to print to the Immediate window.

A question mark (?) is useful shorthand for the Print method. The question mark means the same as Print, and can be used in any context where Print is used. For example, the statements in Figure 7.8 could be entered as shown in the following figure.

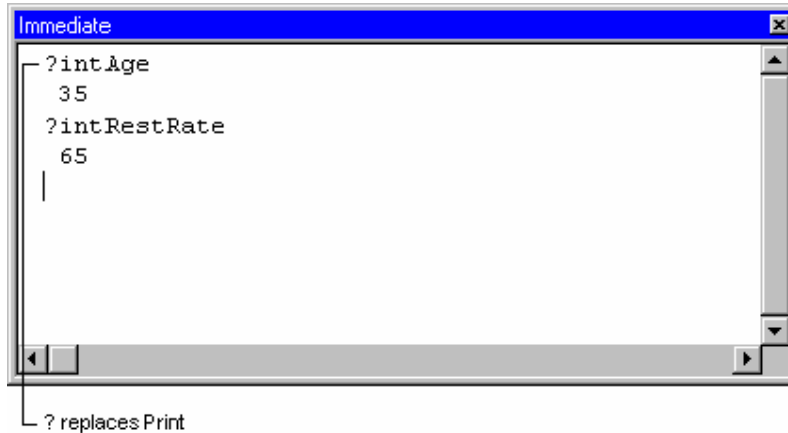


Figure 7.8 Using a question mark instead of the Print method.

Printing Values of Properties and expressions

The value of any expression or properties can also be printed in the Immediate window using the print (or a ?) statement.

Examples

Print 5*10/2

? Label1.Caption

7.4 Assigning Values to Variables and Properties

Testing the application with different data will be much useful to find out the behavior of the application in all possible cases. The immediate window allows to assign new values to the variables and properties of objects in the break mode.

The values of variables and properties can be set with statements like these in the Immediate window:

BackColor = 255

VScroll1.Value = 100

MaxRows = 50

The first statement alters a property of the currently active form, the second alters a property of VScroll1, and the third assigns a value to a variable.

After values are set to one or more variables and properties the execution can be continued to test the its behavior with the new values.

Checking Error Numbers

The Immediate window can be used to display the error message associated with a specific error number. For example to know about the error no 58, the following statement can be used in the Immediate window.

```
error 58
```

This statement will cause the appearance of the appropriate error message, corresponding to the mentioned number as in the Figure 7.9

The Watch Window

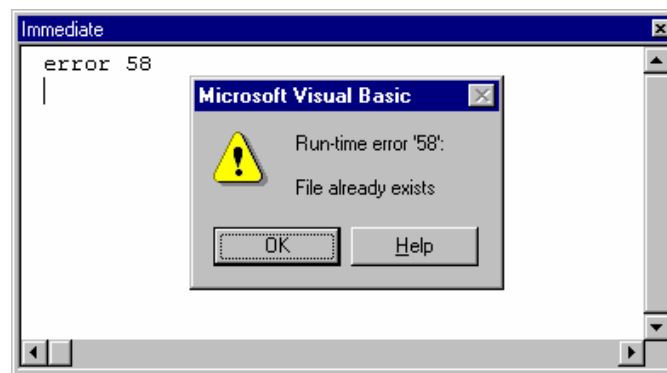


Figure 7.9 Checking error numbers using the Immediate window.

The watch window also helps to monitor the data (in the break mode), in more detailed fashion than the immediate window.

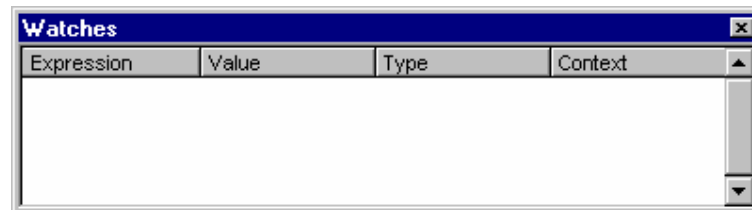


Figure 7.10 The Watch Window.

Window Elements

Expressions

Lists the watch expression with the Watch icon, on the left.

Value

List the value of the expression at the break mode. The value can be edited and then pressing Enter key, the UP ARROW key, the DOWN ARROW key, TAB will cause the validation of the change. If the value is illegal, the Edit field remains active and the value is highlighted. A message box describing the error will also appear. The change can be cancelled by pressing the ESC key.

Type- Lists the expression type.

Context -Lists the context of the watch expression. If the context of the expression isn't in the current scope the current value isn't displayed.

For example in an application there are two procedures namely Proce1 and Proce2. A variable Proce1Var of integer type is declared in Proce1. And Proce2Var of type variant is declared in Proce2. Assuming the application enters into the Break mode when Proce2 is executing, then the watch window will appear as Figure 7.11, provided Proce1Var and Proce2Var should be added to the Watch Window(Adding watches are explained in the next section):

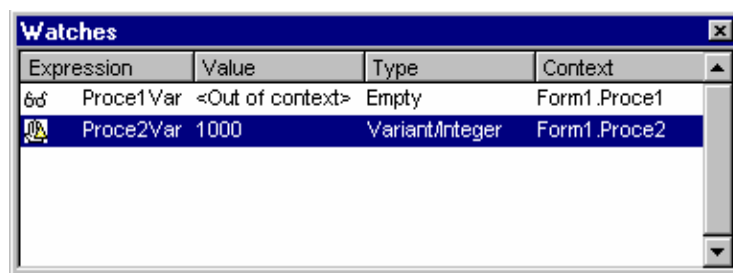


Figure 7.11 The Watch window displaying the values of variables.

As the Proce1Var is not in the current scope, its value is not displayed.

Adding a Watch Expression

An expression can be added to the Watch Window either at Design-time mode or at Break mode by using Add Watch dialog box.

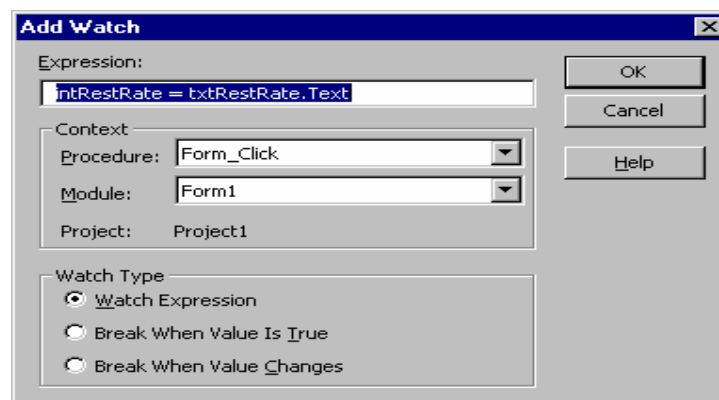


Figure 7.12 The add watch dialog box.

The following table describes the components of the Add Watch dialog box.

Component	Description
Expression box	Accepts expression which to be evaluated by the watch expression. The expression is a variable, a property, a function call, or any other valid expression.
Context option group	Sets the scope of variables watched in the expression. It is used if the application has variables of the same name with different scope.
Watch Type option group	Sets how Visual Basic responds to the watch expression. Visual Basic can watch the expression and display its value in the Watch window when the application enters break mode. Or the application can be made to enter into break mode automatically when the expression evaluates to a true (nonzero) statement or each time the value of the expression changes.

Table7.1 Components of the Add Watch dialog box.

To add a watch expression

- ❖ From the Debug menu, choose Add Watch.
- ❖ The current expression (if any) in the Code Editor will appear in the Expression box on the Add Watch dialog box. If this isn't the expression to watch, enter the expression to evaluate in the Expression box.
- ❖ If necessary, set the scope of the variables to watch.
- ❖ If necessary, select an option button in the Watch Type group to determine how you want Visual Basic to respond to the watch expression.
- ❖ Choose OK.

Note:An expression can be added by dragging it from the Code Editor and dropping at the Watch window.

Editing or Deleting a Watch Expression

Any watch listed in watch window can either be Edited or Deleted. The Edit Dialog box is shown in the Figure 7.13

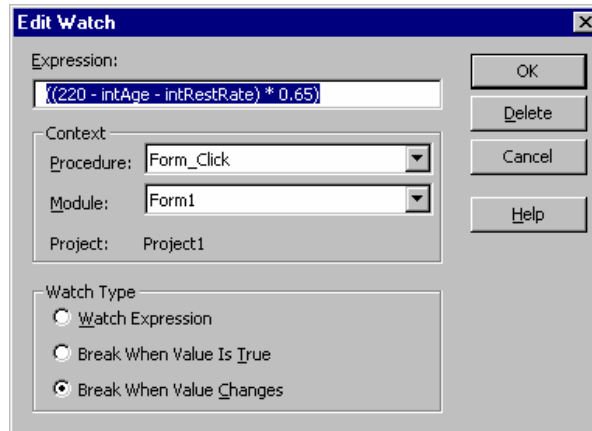


Figure 7.13 The Edit Watch window.

To edit a watch expression

- ❖ In the Watch window, double click the watch expression, which is to be edited.
--or--
Select the watch expression which is to be edited and choose Edit Watch from the Debug menu.
- ❖ The Edit Watch dialog box is displayed and is identical to the Add Watch dialog box except for the title bar and the addition of a Delete button.
- ❖ Make any changes to the expression, the scope for evaluating variables, or the watch type.
- ❖ Choose OK.

To delete a watch expression

In the Watch window, select the watch expression, which is to be deleted, then press the Delete button.

Identifying Watch Types

At the left edge of each watch expression in the Watch window is an icon identifying the watch type of that expression. Figure 7.14 defines the icon for each of the three watch types.

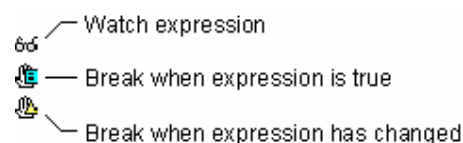


Figure 7.14 Watch type icons.

The Quick Watch Window

This window helps to check the value of a property, variable, or expression for which the watch expression is not defined. The following figure 7.15 shows a typical quick watch window.

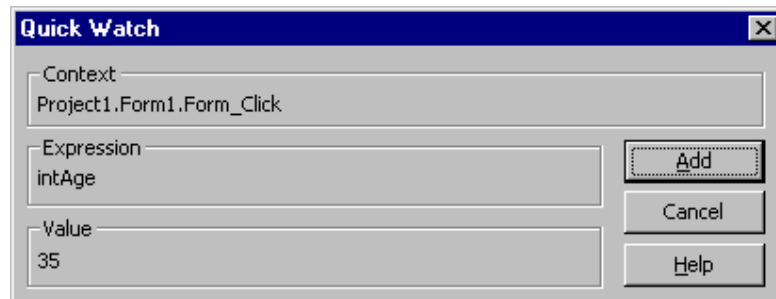


Figure 7.15 Quick Watch window.

The Quick Watch window shows the value of the expression that is selected from the Code window. The displayed expression can be added to the watch window by simply clicking the Add button. If the current expression is invalid, then the Visual Basic will flash the error message “Invalid Expression”.

To display the Quick Watch dialog box

- ❖ Select a watch expression in the Code window.
- ❖ Click the Quick Watch button on the Debug toolbar.
--or--
Press SHIFT+F9.
--or--
From the Debug menu, choose Quick Watch.

- ❖ If the expression in the Quick Watch dialog box is to be added to the Watch window, choose the Add button.

The Locals Window

This *window* shows the value of any variables within the scope of the current procedure. As the execution switches from procedure to procedure, the contents of the Locals window change to reflect only the variables applicable to the current procedure. Figure 7.16 shows the Locals window.

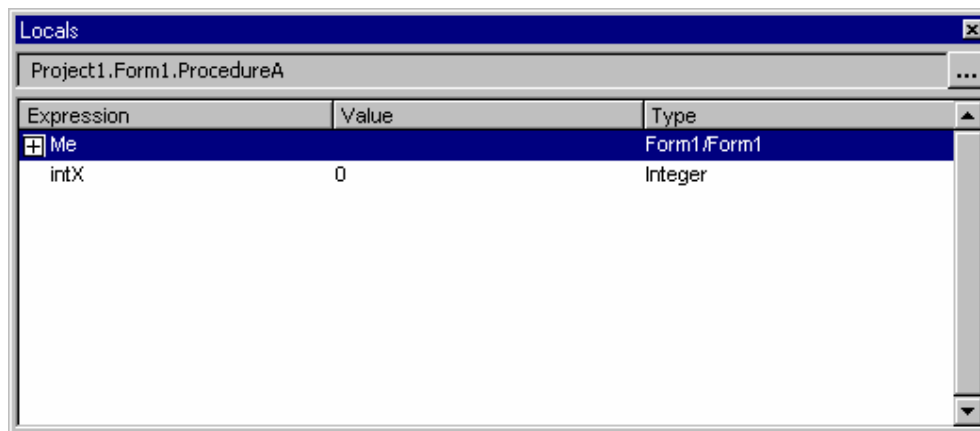


Figure 7.16 The Locals window.

Monitoring the Call Stack Using Call Stack Window

The Call Stack window shows a list of all active procedure calls. *Active procedure calls* are the procedures in the application that were started but not completed.

The Call Stack dialog box helps to trace the operation of an application as it executes a series of nested procedures. For example, an event procedure can call a second procedure, which can call a third procedure and so on – all before the event procedure that started this chain is completed. Such nested procedure calls can be difficult to follow and can complicate the debugging process. Figure 7.17 shows the Call Stack dialog box.

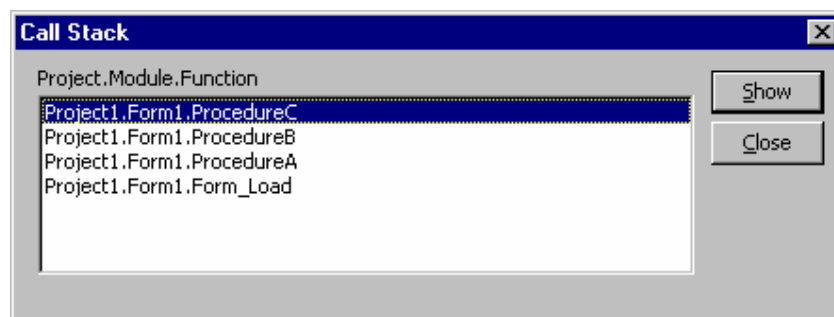


Figure 7.17 The Call Stack window.

This window can also be displayed only in the break mode.

To display the Call Stack dialog box

- ❖ From the View menu, choose Call Stack.
- or--
- Click the Call Stack button on the Debug toolbar.

- or-
- Press CTRL+L.
- or-
- Click the button next to the Procedure box in the Locals window.

Tracing Nested Procedures

The Call Stack window lists all the active procedure calls in a series of nested calls. It places the earliest active procedure call at the bottom of the list and adds subsequent procedure calls to the top of the list. The information given for each procedure begins with the project name, followed by module or form name, followed by the name of the called procedure.

This window can be used to display the statement in a procedure that passes control of the application to the next procedure in the list.

To display the statement that calls another procedure in the Calls Stack dialog box

- ❖ In the Call Stack dialog box, select the procedure call that is to be display.
- ❖ Choose the Show button.

The dialog box is closed and the procedure appears in the Code window. The cursor location in the Code window indicates the statement that calls the next procedure in the Call Stack window. For example if the ProcedureB is chosen and the Show button is clicked in the Call stack Window (Figure 7.17) code window will appear indication the line that that calls the next procedure as in Figure 7.18

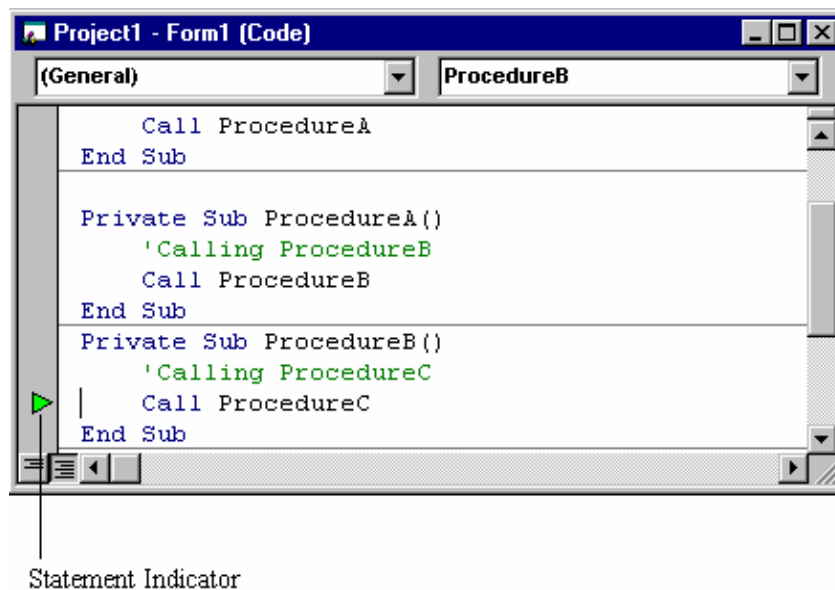


Figure 7.18 Statement indicator in a procedure that calls the next procedure.

Note: If the current procedure (top most procedure) in the Call stack window is chosen, the cursor will appear at the current statement.

7.5 Modules

Code in Visual Basic is stored in modules. There are three kinds of modules: form, standard, and class.

Simple applications can consist of just a single form, and all of the code in the application resides in that form module. As your applications get larger and more sophisticated, you add additional forms. Eventually you might find that there is common code you want to execute in several forms. You don't want to duplicate the code in both forms, so you create a separate module containing a procedure that implements the common code. This separate module should be a standard module. Over time, you can build up a library of modules containing shared procedures.

Each standard, class, and form module can contain:

- **Declarations.** You can place constant, type, variable, and dynamic-link library (DLL) procedure declarations at the module level of form, class or standard modules.
- **Procedures.** A Sub, Function, or Property procedure contains pieces of code that can be executed as a unit.

Form Modules

Form modules (.FRM file name extension) are the foundation of most Visual Basic applications. They can contain procedures that handle events, general procedures, and form-level declarations of variables, constants, types, and external procedures. If you were to look at a form module in a text editor, you would also see descriptions of the form and its controls, including their property settings. The code that you write in a form module is specific to the particular application to which the form belongs; it might also reference other forms or objects within that application.

Standard Modules

Standard modules (.BAS file name extension) are containers for procedures and declarations commonly accessed by other modules within the application. They can contain global (available to the whole application) or module-level declarations of variables, constants, types, external procedures, and global procedures. The code that you write in a standard module isn't necessarily tied to a particular application; if

you're careful not to reference forms or controls by name, a standard module can be reused in many different applications.

Class Modules

Class modules (.CLS file name extension) are the foundation of object-oriented programming in Visual Basic. You can write code in class modules to create new objects. These new objects can include your own customized properties and methods. Actually, forms are just class modules that can have controls placed on them and can display form windows.

7.6 Enum (Enumerated Constant)

Enumeration provide a convenient way to work with sets of related constants and to associate constant values with names. For example, you can declare an enumeration for a set of integer constants associated with the days of the week, then use the names of the days in code rather than their integer values.

You create an enumeration by declaring an enumeration type with the Enum statement in the Declarations section of a standard module or a public class module. Enumeration types can be declared as Private or Public with the appropriate keyword. For example:

```
Private Enum MyEnum  
    -or-  
Public Enum MyEnum
```

By default, the first constant in an enumeration is initialized to the value 0, and subsequent constants are initialized to the value of one more that the previous constant. For example the following enumeration, Days, contains a constant named Sunday with the value 0, a constant named Monday with the value 1, a constant named Tuesday with the value of 2, and so on.

```
Public Enum Days  
    Sunday  
    Monday  
    Tuesday  
    Wednesday  
    Thursday  
    Friday  
    Saturday
```

End Enum

You can explicitly assign values to constants in an enumeration by using an assignment statement. You can assign any long integer value, including negative numbers. For example you may want constants with values less than 0 to represent error conditions.

In the following enumeration, the constant `Invalid` is explicitly assigned the value `-1`, and the constant `Sunday` is assigned the value `0`. Because it is the first constant in the enumeration, `Saturday` is also initialized to the value `0`. Monday's value is `1` (one more than the value of `Sunday`), Tuesday's value is `2`, and so on.

```
Public Enum WorkDays
    Saturday
    Sunday = 0
    Monday
    Tuesday
    Wednesday
    Thursday
    Friday
    Invalid = -1
End Enum
```

By organizing sets of related constants in enumerations, you can use the same constant names in different contexts. For example, you can use the same names for the weekday constants in the `Days` and `WorkDays` enumerations.

To avoid ambiguous references when you refer to an individual constant, qualify the constant name with its enumeration. The following code refers to the `Saturday` constants in the `Days` and `WorkDays` enumerations, displaying their different values in the Immediate window.

```
Debug.Print "Days.Saturday = " & Days.Saturday
Debug.Print "WorkDays.Saturday = " & WorkDays.Saturday
```

You can also use the value of a constant in one enumeration when you assign the value of a constant in a second enumeration. For example, the following declaration for the `WorkDays` enumeration is equivalent to the previous declaration.

```
Public Enum WorkDays
    Sunday = 0
    Monday
    Tuesday
```

```

Wednesday
Thursday
Friday
Saturday = Days.Saturday - 6
Invalid = -1
End Enum

```

After you declare an enumeration type, you can declare a variable of that type, then use the variable to store the values of enumeration's constants. The following code uses a variable of the `WorkDays` type to store integer values associated with the constants in the `WorkDays` enumeration.

Dim MyDay As WorkDays

```

MyDay = Saturday           ' Saturday evaluates to 0.
If MyDay < Monday Then    ' Monday evaluates to 1,
                          ' so Visual Basic displays
                          ' a message box.

MsgBox "It's the weekend. Invalid work day!"

End If

```

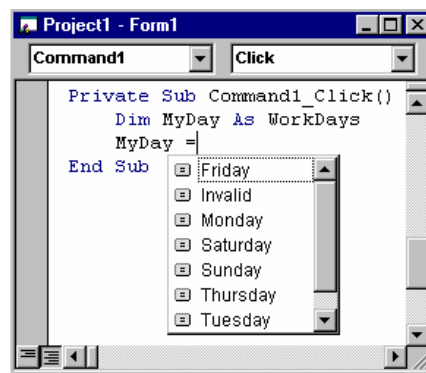


Figure 7.19 Visual Basic automatically displays an enumeration's constants. Because the constant `Sunday` also evaluates to 0, Visual Basic also displays the message box if your place "Saturday" with "Sunday" in the second line of the example:

```

MyDay = Sunday           ' Sunday also evaluates to 0.

```

7.7 Short Summary

- ☞ Debugging is process by which errors are identified and resolved in the code. Typically, debugging occurs at several levels.

-
- ☞ In visual Basic errors could occur anywhere during the entire development cycle such as Design time, Compile Time, Run time.
 - ☞ Errors that occur in the IDE and before the program is compiled are called design-time bugs.
 - ☞ Compile time bugs are those that occur when we attempt to create the program executable file (EXE) or run the project.
 - ☞ The Debug Window has two parts - the Watch Window and Immediate Window
 - ☞ Visual Basic treats constant values in an enumeration as long integers. If you assign a floating-point value to a constant in an enumeration, Visual Basic rounds the value to the nearest long integer.
 - ☞ Visual Basic provides a built-in enumeration, `vbDayOfWeek`, containing constants for the days of the week. To view the enumeration's predefined constants, type `vbDayOfWeek` in the code window, followed by a period. Visual Basic automatically displays a list of the enumeration's constants
 - ☞ When you type the second line of code in the example in the code window, Visual Basic automatically displays the `WorkDays` enumeration's constants in the Auto List Members list.
 - ☞ Although you normally assign only enumeration constant values to a variable declared as an enumeration type, you can assign any long integer value to the variable. Visual Basic will not generate an error if you assign a value to the variable that isn't associated with one of the enumeration's

7.8 Brain Storm

1. Explain the term debugging.
2. What are the different levels of debugging seen in Visual Basic 6.0?
3. What is the key stroke used to invoke a debug window
4. Define Design time debug
5. Differentiate an array over ENUM

Lab Unit - 7 (2 Real Time Hrs

1. Write a suitable program that displays a run time error on type mismatch with user giving a date input and validating the data input.

Lecture - 8

Error Handling

Objectives

In this lecture you will learn the following

- ❖ Create error handler
- ❖ Understand how to handle the errors during run-time
- ❖ Know how to handle errors in an error-handling routine
- ❖ Knowing some common error-handling styles

Lecture unit - 8

- 8.1 Snap Shot
- 8.2 Designing an error handler
- 8.3 Setting the error trap
- 8.4 Writing an Error Handling Routine
- 8.5 Exiting an Error Handling Routine
- 8.6 Testing Error handling by generating errors
- 8.7 Short Summary
- 8.8 Brain Storm

Lab unit 5 (2 Real Time Hrs)

8.1 SNAP SHOT

Though an application is thoroughly checked and debugged, still errors can occur. For example, accidental deletion of files, disk drives run out of space, or network drives disconnect unexpectedly will give run-time errors. So the application must be well armed to meet these circumstances. The Error Handling routines of an application will guard that application from abrupt termination and will meet the critical situations.

8.2 Designing an Error Handler

An *error handler* is a routine for trapping and responding to errors in an application. Error Handlers can be placed in any procedure where there is a possibility for errors. The process of designing an error handler involves three steps:

1. *Setting, or enabling,* an error trap by telling the application where to branch to (which error-handling routine to execute) when an error occurs.

The **On Error** statement enables the trap and directs the application to the label, marking the beginning of the error-handling routine.

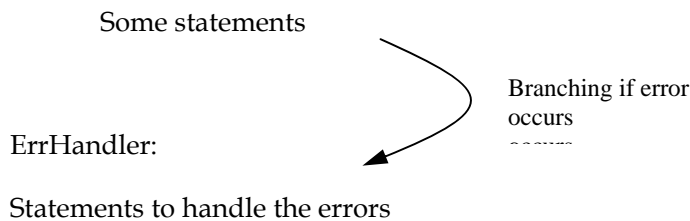
Example

On Error Goto ErrHandler

2. Writing an error-handling routine that responds to all the anticipated errors. If control actually branches into the trap at some point, the trap is then said to be *active*.

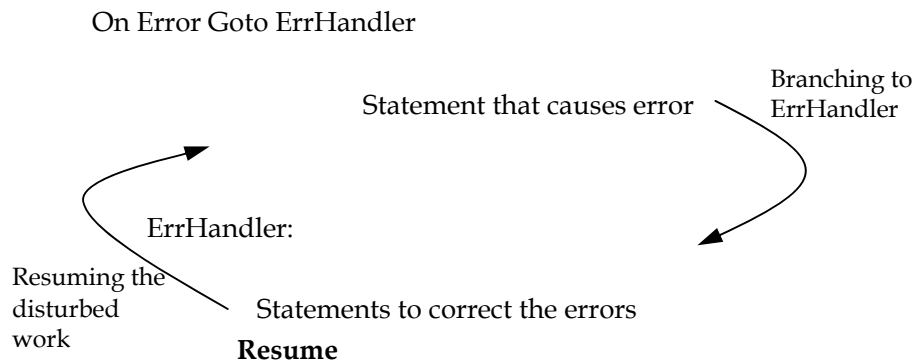
Example

On Error Goto ErrHandler



3. Exit the error-handling routine. When the error handling routine corrects the encountered error it must exit itself to resume the execution of the temporarily halted application. The exit from the error-handler is done using one of the Resume statements (Explained shortly).

4. Example



All the above steps are explained in detail in the following sections.

8.3 Setting the Error Trap

An error trap is enabled when Visual Basic executes the On Error statement, which specifies an error handler. The error trap remains enabled while the procedure containing it, is active – that is, until an Exit Sub, Exit Function, End Sub, or End Function statement is executed for that procedure.

The On Error GoTo *line* statement sets an error trap that jump to an error-handling routine. The *line* in that statement indicates the label identifying the error-handling code. In a procedure only one error trap can be enabled at a time. But the procedure can have more than one-error traps and different ones can be enabled at different times.

Example

```

Sub ProcedureA()
    If Some-condition then
        On Error Goto ErrorHandler_One
    Else
        On Error Goto ErrorHandler_Two
    End if
    -----
    -----
    ErrorHandler_One:
        -----
        -----
    ErrorHandler_Two:
        -----
        -----
  
```


8.4 Writing an Error-Handling Routine

The first step in writing an error-handling routine is adding a line label to mark the beginning of the error handling routine. The line label should have a descriptive name and must be followed by a colon. A common convention is to place the error-handling code at the end of the procedure with an Exit Sub, or Exit Function statement immediately before the line label. This allows the procedure to avoid executing the error-handling code if no error occurs.

Example

```
Sub SomeProcedure()  
  
    On Error Goto ErrHandler  
    -----  
    -----  
    Debug.Print "Execution of Procedure Completed_  
    Successfully Exiting..."  
  
    Exit Sub      'If not provided Err-Handler will be_  
                  'executed.  
  
ErrHandler:  
  
    Debug.Print "Error Occured - Handling Begins"  
    -----  
    -----  
  
End Sub
```

As the code is executed in the top-down approach (i.e. each statement is executed sequentially from the first line to last line), after the execution of the line Debug.Print "Execution of Procedure Completed successfully - Exiting...", Visual basic will start to execute the statements in the Err-Handler unless Exit Sub function is placed immediately before the label Err-Handler.

The body of the error handling routine contains the code that actually handles the error, usually in the form of a Case or If...Then...Else statement. These statements determines which error has occurred and provide course action for each error, for example, prompting the user to insert a disk in the case of a "Disk not ready" error. An option should always be provided to handle any unanticipated errors by using the Else or Case Else clause. For example a typical error handler may be as follow:

Err_Handler:

```

If err.Number =6          ' Over flow
    -----
Elseif err.Number = 7    ' Out of Memory
    -----
Elseif err.Number = 11   ' Division by zero
Else                      ' Known error
    -----
End if

--- Or ---
Select Case Err.Number
    Case 6
    -----
    Case 7
    -----
    Case 11
    -----
    Case Else
    -----
End Select

```

8.5 Exiting an Error-Handling Routine

There are three kinds of Resume statements and one of them can be used to exit from an Error -Handling routine, based on the circumstances. They are explained in the following table 2.1.

Statement	Description
Resume	Program execution resumes with the statement that caused the error. It is used, to repeat an operation after correcting the condition that caused the error.
Resume Next	Resumes program execution at the statement immediately following the one that caused the error. It is used when the error handler is not able to correct the error.
Resume <i>line</i>	Resumes program execution at the label specified by <i>line</i> , where <i>line</i> is a label that must be in the same procedure as the error handler.

Table 8.1 Resume Statements.

The Figure 2.1 shows the execution flow when the Resume or Resume Next statement is used.

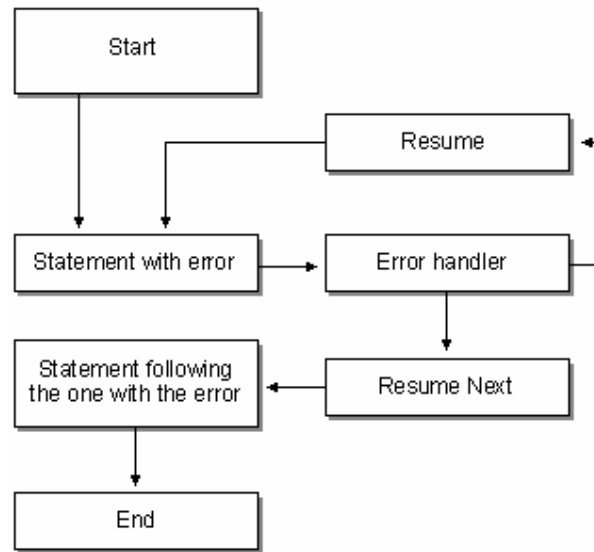


Figure 8.1 The program flow with Resume and Resume Next statements.

The Resume Statement Example

```

Sub SomeProcedure()

    ' Enabling the error trap
    On Error Goto ErrHandler

    -----
    ' If denominator is equals to 0 -- branching to
    ' ErrHandler
    ' After error-correction execution resumes here
    divide = number / denominator
    -----

    Exit Sub
ErrHandler:
    If err.Number = 11 then      ' Division by zero
        ' Changing the denominator to its default value,
        ' say 10
        denominator = 10
        Resume
    Else
        ' Some unknown error
        Debug.Print "Unknown error - Exiting"
        Exit Sub
  
```

```

    End if
End Sub

```

The Resume Next Statement Example

```

Sub SomeProcedure()

Dim intVar as Integer

' Enabling the error trap
On Error Goto ErrHandler

-----
' If inVar gets value greater than its max
' range(32767) - error occurs - branching to ErrHandler
intVar = some expression

' After error-correction execution resumes here
-----
Exit Sub
ErrHandler:

If err.Number = 6 then      ' Overflow
    ' Setting intVar to its max range
    intVar = 32767
    Resume Next
Else
    ' Some unknown error
    Debug.Print "Unknown error - Exiting"
    Exit Sub
End if
End Sub

```

Resuming Execution at a Specified Line

Some times, after correcting the error, it may be desirable to resume the execution at some specific point. The Resume *line* statement is handy for it. The *line* specifies a label name, which should be written in the same procedure as the error handler. Here is an example for Resume *line* statement.

```

Sub SomeProcedure()

```

```
' Enabling the error trap
On Error Goto ErrHandler

-----
' assuming error occurs here and resumption of
' execution is desirable at the place marked with
' label ResumeHere.
-----

Exit Sub

' After error correction execution resumes here
ResumeHere:
-----

Exit Sub

ErrorHandler:

' Assuming the error is corrected - exiting the error
' handler
Resume ResumeHere

End Sub
```

8.6 Testing Error Handling by Generating errors

To see the error handling routines work well, the expected errors can be generated in the code. The Raise method of Err object is used to generate errors.

Syntax

Err.Raise *errnumber*

The Err object is Visual Basic's globally defined error object. Programmers need not to declare them. The *errnumber* specifies the error that is to be simulated. For example the number 71 is associated with the error "Disk Not Ready". To simulate this error the following statement is used.

```
Err.Raise 71
```

When this statement is executed, Visual Basic will generate the error 71 and the error handler (if provided) will catch the error and will handle it. After it is assured that error handler, handles the error well, the statements that raises errors must be removed, before the application is compiled.

Error Object

As soon as an error occurs in Visual Basic code, the special built-in Err Object becomes active.

The Err Object includes properties that report the number of error, the error's description, as well as other useful information.

Using the Err.Description Property

The Err.description property returns the default description of the error that has occurred.

Using the Err.Number Property

The most important property of the Err object is the number associated with the error that occurred.

8.7 Short summary

- ☞ The Err object contains information about run - time errors. Its number property returns a numeric value specifying the occurred error
- ☞ Err.Number contains the value of the last error that occurred
- ☞ On Error statement enables error trapping and specifies where execution jumps when an error occurs
- ☞ On Error statement included in your error - handling code does not take effect until the end of the error - handling code
- ☞ If a run- time error occurs within an error-handling routine, the error is passed up to the calling procedures

8.8 Brain Storm

1. How do you disable a user defined error handler?
2. Do you explicitly need to declare an error object? Why?
3. What are the types of argument?
4. How do you arrive at the specific error message using an immediate window?
5. How do you disable the error message(s) shown to you by Visual Basic?
6. What is the difference between a compile time error and a runtime error?

Lab Unit 8 (2 Real Time Hrs)

1. Place an On Error GoTo procedure statement as the first line of your program and write an error procedure
2. Have the error handling code in each procedure call a function that identifies the error and indicates the appropriate action
3. Create a project using On Error Resume Next Statement , get a number and divide that number by zero so that an error message should be displayed (Division by zero).

Lecture - 9

Handling Data Base

Objectives

In this chapter you will learn the following

- ❖ About Visual Data manager
- ❖ Know to create Data Base table
- ❖ Know to add, delete records
- ❖ Manipulating records using Visual Data Manager

Lecture Unit - 9

- 9.1 Snap Shot
- 9.2 Visual Data Manager
- 9.3 Creating a Data base using Visual Data Manager
- 9.4 Manipulating Records using Visual Data Manager
- 9.5 Introduction to Data Control
- 9.6 Data Control Property
- 9.7 Data Form Wizard
- 9.8 Short Summary
- 9.9 Brain Storm

Lab Unit - 9 (2 Real Time Hrs)

9.1 Snap Shot

Computers are mainly used for keep track of information. In this unit the will go on creating files by using Visual Data Manager and adding, deleting the records and so on...

9.2 Visual Data manager

VisData is an application that was created using Visual Basic itself. A built version of VisData is used as an add-in, accessible from the Add-Ins menu in the Visual Basic Figure 2.4.

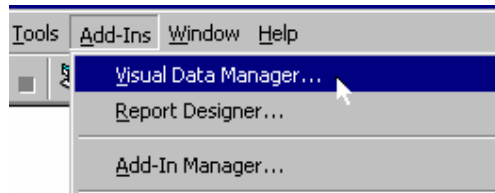


Figure 9.1 Visual Data Manager option in Visual Basic Add-Ins menu, which is used to activate the Visual Data Manager.

It is used to create databases, tables, queries, and indexes and to modify them. When an existing database is opened the Visual Data manager displays the contents and properties of that database in “**Database Window**”. The following figure 2.5 shows the visual data manager window with the BIBLIO database open in design mode.

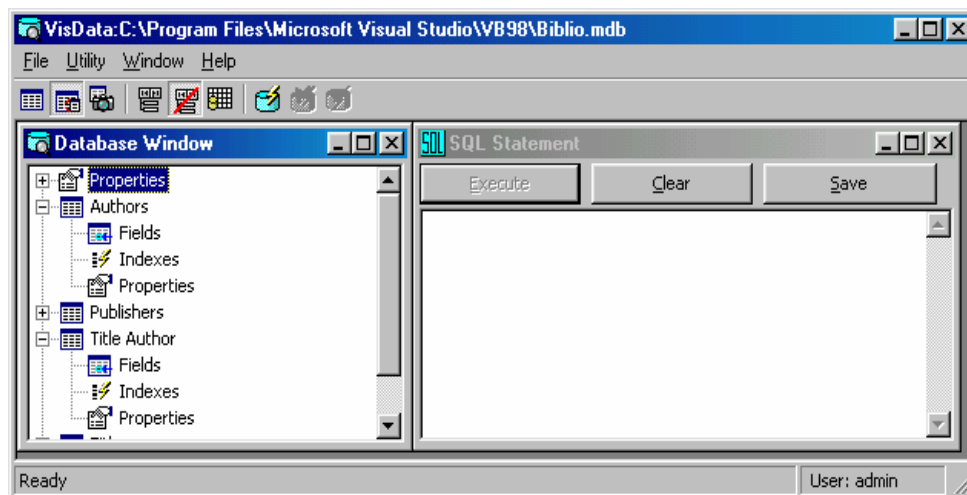


Figure 9.2 VisData Manager

9.3 Creating a Database using a Visual Data Manager

To create a new database, choose File | New | Microsoft Access | Version 7.0 MDB. A dialog box will appear asking path and filename to store the database being created. Give a filename and choose desired location to save the database.

There is one more option in the Microsoft Access submenu, Version 2.0 MDB, which is used to create databases in version 2.0 format. Still many concerns use applications, which are built using earlier versions of Visual Basic. They may require databases in older format. In this case alone, the required database can be created using Version 2.0 MDB option. Otherwise databases can be created in latest version format (why to look back the Stone Age world).

Note: To open an already existing access database choose File | Open Database | Microsoft Access. And select the database name in the Open Microsoft Access database dialog box.

Creating a Simple database

This section shows how to create a simple database Students with a table StudentsDtl (Students' Details), which contains fields StudentId, StudentName and TotalMarks. It will be useful for rest of this chapter.

Choose File | New | Microsoft Access | Version 7.0 MDB.

Type Students in the file name text box and click OK.

Right Click on the database window and choose New Table.

In the Table Structure box type StudentDtls in the Table Name text box.

Click Add Field button in the Table Structure box to add fields.

Type the values as given below in the corresponding text boxes.

<i>Text Box</i>	<i>Value</i>
Name	StudentId
Type	Text
Size	10
Validation Text	Student Id is not specified.
Validation Rule	<> ""

- ❖ Click OK to add the field Name. The Field Dialog box will still remain to accept another field. Proceed for the field StudentName with the values given below:

<i>Text Box</i>	<i>Value</i>
Name	StydentName
Type	Text
Size	30
Validation Text	Student Name is not specified.
Validation Rule	<> ""

Now click OK and proceed for the field TotalMarks with the values given in the following:

<i>Text Box</i>	<i>Value</i>
Name	TotalMarks
Type	Integer
Validation Text	TotalMarks is not specified or negative.
Validation Rule	>= 0

Click OK and the Close to close the Add Field Dialog box.

Click Build Table button in the Table Structure dialog box to complete the creation of table Students.

Creating a Table

Once a database is created, it is ready to give birth to tables. To create a table, Right click in the database window and choose New Table from the context menu Figure 9.3.

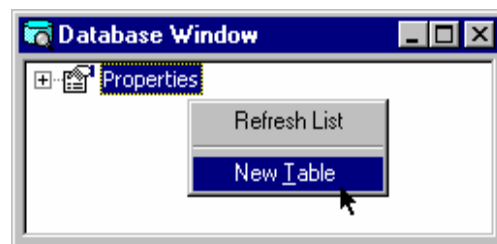


Figure 9.3 Context menu used to create a new table.

Choosing the above option will bring a dialog box called Table Structure dialog box. Enter the name of the table being created in the table name text box. And to add fields to the table follow the steps given in the next section.

Adding Fields

In the table structure dialog box click the **Add Field** button to display the Add Field dialog box. It appears as in Figure 9.4

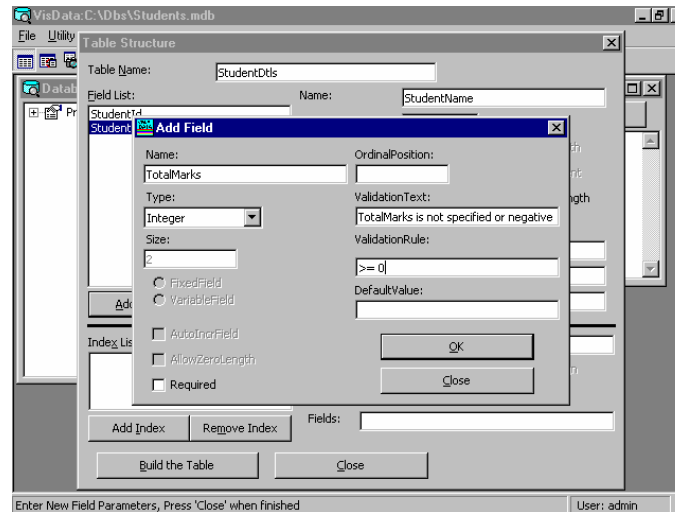


Figure 9.4 VisData manager with Add Field Dialog box.

Add Field Dialog Box Options

The options of the Add Field dialog box are given below.

Name

Allows to type the name of the field being added.

Ordinalposition

Allows to determine the relative position of the field.

Type

Lets to determine the operational or data type of the field such integer, text etc.

Validationtext

Lets to add a text of the message that the application displays if a user tries to enter an invalid value for a field.

Size

Lets to determine the maximum size, in bytes, of the field.

- FixedField — If selected, creates a field with a fixed size.
- VariableField — If selected, allows the user to modify the size of the field by dragging its borders.

Validation rule

Lets to determine what data is valid in a field as it is added. It is a String that describes a comparison in the form of an SQL WHERE clause, but without the WHERE reserved word. For example, if the value of the CustomerAge must be positive, then its ValidationRule string will be “ > 0 ”.

Default value

Lets to determine the default value for the field.

Autoincrfield

Automatically gets value that is incremented by one from the last record when a new record is added.

Allowzerolength

Allows having a zero-length string as a valid setting.

Required

Indicates if the field requires a non-NULL value.

Ok

Appends the current Field definition to the current table.

Close

Closes the form when you are finished adding fields.

Opening the tables

To open a table select the table in the database window, right click on it , and choose open.

-- or --

Simply double click the table name.

This will open the table in a separate form (can be termed as table form) as shown in the following figure 9.5.

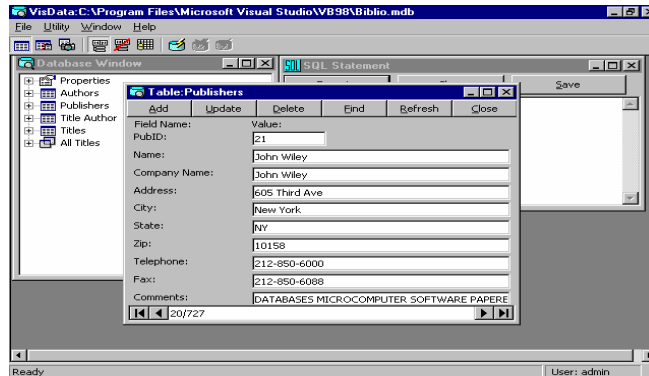
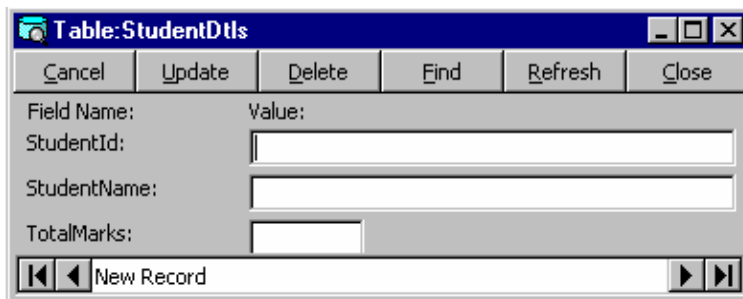


Figure 9.5 Opened Publishers table in VisData.

In this form records can be added, deleted and updated. The following steps show how to add, delete and Edit the records to the StudentDtls table.

9.4 Manipulating records using Visual Data Manager

Double click on the table name StudentDtls in the database window to open it.



As no record is added to the table, the StudentDtl table form will appear in ready state to accept values for new record as given below Figure 9.6.

Figure 9.6 StudentDtls table form in ready state to accept data.

Enter the following values in the corresponding text boxes:

<i>Field Name</i>	<i>Value</i>
StudentId	1005
StudentName	Malarvizhi
TotalMarks	450

Click Update button to commit new record addition.

Now the Cancel button will be replaced by Add button.

From now onwards to add records, Add button is to be clicked first, values to be entered next and finally Update button is to be clicked. By following these steps enter the data given below to add five more records.

StudentId	StudentName	TotalMarks
1001	Kumar	475
1003	Raju	480
1002	Shanmugam	466
1004	VaniSaraswathy	450
1000	SelvaSudari	455

Note: While entering records instead of clicking update button, cancel button can be clicked to cancel the record addition process.

At the bottom of the StudentDtls, there is a control called Data Control. It contains four buttons that enables to navigate the records such as moving to first record, moving to previous record, moving to next record and moving to last record. The following figure 9.10 shows the data control with its buttons' description.



Figure 9.10 Data control.

To delete a record, use the data control to move to the record, which is to be deleted and click delete button. When VisData manager asks confirmation, click yes for deletion. Clicking No will abort the deletion.

To edit a record, use data control to move to the required record, modify the record and either click Update or any one of navigation button. VisData will ask whether to commit changes as shown below Figure 9.11

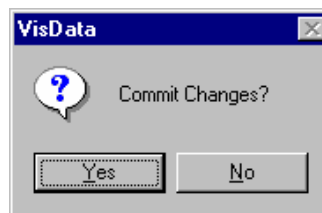


Figure 9.11 VisData asking confirmation to commit changes.

Click yes to commit changes or No to cancel the changes.

9.5 Introduction to Data Control

Many modern applications are concerned with the storage, organization and retrieval of data. To address the needs of the software developers creating these products, Visual Basic provides a rich set of data access features. VB provides many standard controls, such as text boxes and labels, that can be bound to the Data control, in addition to the DBCombo, DBGrid and DBList. These controls are provided specifically for binding to a data control.

A data control binds data aware controls to Microsoft Access or to other ODBC data sources, enabling to move from record to record and to display and manipulate data from the records in bound controls. A bound control is a control that is assigned to a field in a database.

Most data-access operations can be performed with a data control without writing any code. If a database and a record source – a table, a view, a SQL statement is set to a data control, then it is automatically populated with data from that database when the form that has the data control, is activated.

The following figure 9.12 shows the data control and DAO with bound controls.

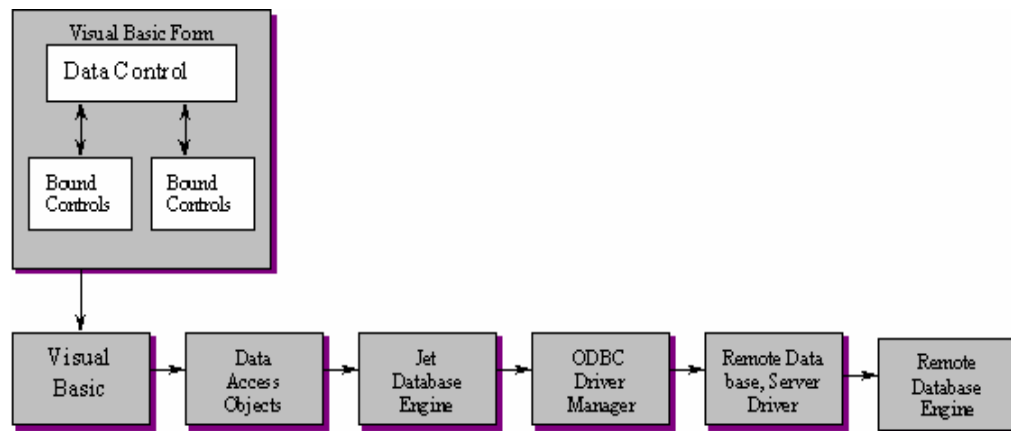


Figure 9.12 The data control, DAO, and bound controls

The intrinsic Data control on the Visual Basic Toolbox, implements data access by using the Microsoft Jet Database engine –the same database engine that powers Microsoft Access. This technology gives seamless access to many standard database formats and allows to create data-aware applications without writing any code. The intrinsic Data control is best suited to smaller (desktop) databases, such as Access and ISAM databases.

Data control's Icon



The intrinsic Data control can be used to create applications that display, edit, and update information from many types of existing databases, including Microsoft Access, Btrieve, dBase, Microsoft FoxPro, and Paradox.

9.6 Data Control Properties

Align	Set or return a value that determines how the control is placed on the form
Appearance	which determines whether or not to use 3D appearance
BackColor	a numeric value that determines the control's background color.
BOFAction	Which determines how the data control reacts to EOF and BOF conditions.
Caption	set or return a string representing the object's caption
Connect	set or return the type of database being used
DataBase	which is a read-only property that exposes the data controls underlying Database object
DatabaseName	which specifies the database to be used.
DataField	Which sets or return a value that binds the control to a field in the current record.
DragIcon	set or return the icon to be used for drag operation
DragMode	set or return the type of drag mode available
EditMode	which specifies the editing state of the current record in the Data Control's Recordset.
EOFAction	Which determines action when the recordset's EOF is reached

Exclusive	This reads or sets whether a database is opened for single-user or multiuser access. Read and write at both design time and runtime.
Options	set or return one or more characteristics of the Recordset object
Recordset	The data (set of records) available to the Data control
RecordsetType	sets or return the type of Recordset object of the data control
RecordSource	which specifies the part of the database seen by the control.

9.7 Data form wizard

The Data form Wizard is designed to automatically generate that contain individual bound controls and procedures used to manage the data obtained from the database. You can use the Data form Wizard to create single-query forms to access data from a single table or query, master/detail-type forms to manage more complex one -to-many data relationships, or grid forms to manage data from a data control.

As It takes you through several steps, the Wizard will prompt you for the information it needs to build the data forms properly. However, because the forms the Wizard builds are fairly generic, you will probably modify them to more closely resemble the remaining forms in your application. To see how this Wizard works, start anew project, and open the Data Form Wizard form the Add-Ins menu.

The first dialog to be displayed is the Introduction dialog, which allows you to select a data form profile you have previously saved. For this demo, click Next to choose the database type with which you are working. At this time, there are only two database types to choose from: Microsoft Access and open database connectivity (ODBC). Choose Access and click Next to continue. After you've selected your database type, you must specify the actual database location or connection information. For Access, you are prompted for the database location as shown in Figure 9.13, however, if you are using ODBC, you will be prompted for the appropriate connection information as shown in Figure 9.14

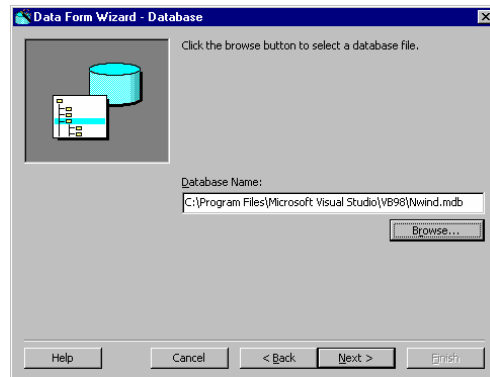


Figure - 9.13 Specifying the location of the Access.mdb file

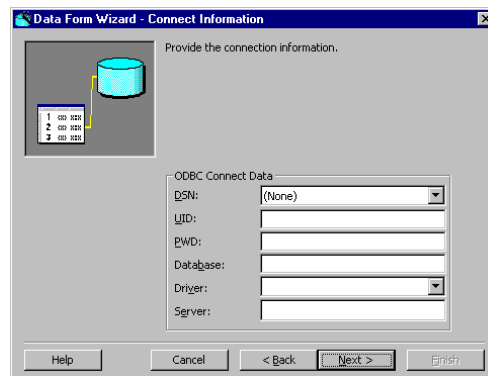


Figure 9.14 - Setting the information required to connect to an ODBC database

Locate and select the North Wind database (NWIND.MDB) in the Visual Basic directory and then click Next to continue. As you move to the next dialog, the Wizard actually connects to and opens the specified database. The form dialog (Figure 9.15) asks you for the name of the new form, the form layout you want to create, and the type of data access connection to use.

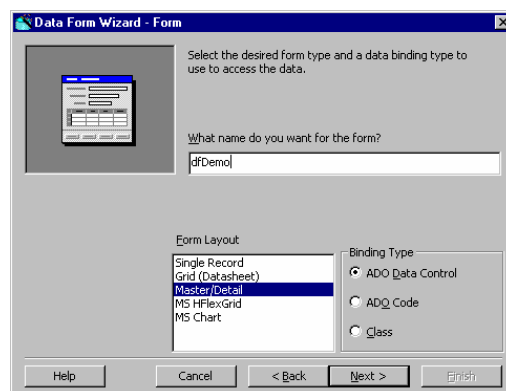


Figure 9. 15 – Specifying the Form layout and data access type

Because each of the form layouts requires slightly different information, the remaining steps will vary depending on the form layout you have chosen. For this example choose the Master/Detail form layout and leave the default Binding type of ADO Data Control. Finally, name the new form "CH16Demo" and click the Next Button. Because you selected the Master/Details layout, you will now be prompted for the record source and fields for both the master and detail queries. The first dialog, Master Record Source (see Figure 9.16) requests the master query source.

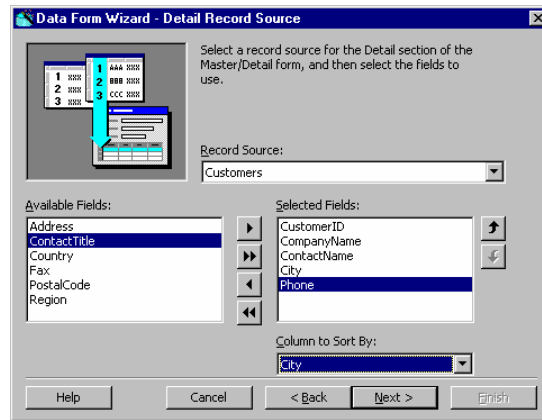


Figure 9. 16– Selecting the master record source for the Wizard to use

From the record source drop-down, select the Customers table; then select the fields shown in fig. Finally, set the sort column to City and click the next button to define the detail query information. As you see from figure 9.17, the Detail Record Source dialog looks exactly like the Master dialog.

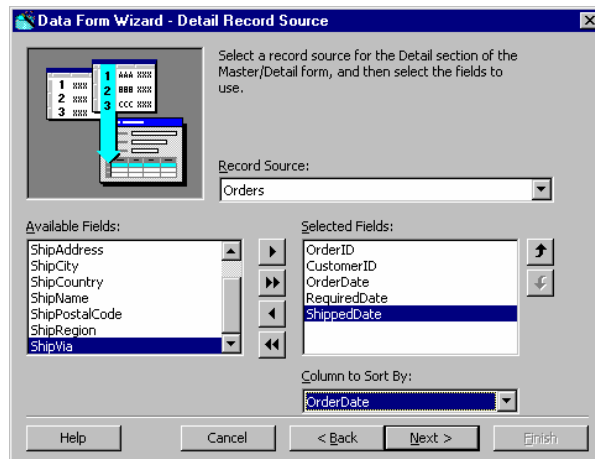


Figure 9.17 – Defining the join relationship between the two queries

On this form, select the Orders table from the Record Source drop-down, and then select the fields shown in figure. Again, set the sort column to OrderDate and click

Next to continue. The Record Source Relation dialog(see figure 9.18) requires you to set the relationship or “join” between the Master and Detail queries.

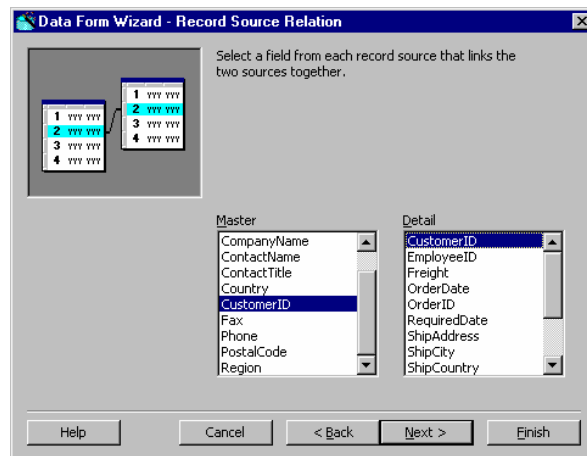


Figure 9.18 – Defining the join relationship between the two queries

Set the CustomerID as the fields that links the two sources and click Next. The Control Selection dialog (see fig 9.19) lets you choose the data navigation and manipulation controls you want to appear on the form and enables the Wizard to create the associated code for each selected control. The available controls to choose from are the following :

- Add - Places an Add button on the form with the associated code to add new records to the database
- Update - Adds the Update button and its associated code to update the database from the form. This button is available only if you are using the Data Control Binding Type.
- Delete - Provides the ability to delete records from the database.
- Refresh - Will add the code and command button to allow the user to request a database refresh.
- Close - Includes the code and command button to close the data form
- Show Data Control - If this checkbox is clicked, the Data Control will be displayed on the form.

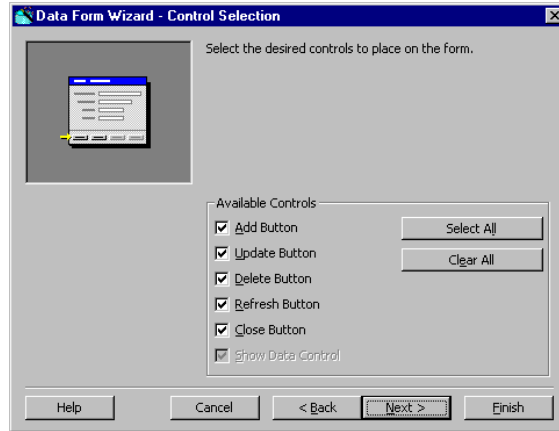


Figure 9.19 Choosing the data command buttons to place on the form

For this example, leave the defaults and click Next. You can now save all of the settings you have selected as part of a profile. Then click the Finish button to allow the Wizard to perform its magic. The form shown in figure 9.20 is the end result of all of the options you selected.

Try running this program to see how the Master/Detail form works. Basically, you will see one customer at a time with all of the customer's orders listed in the grid, as shown in figure 9.21

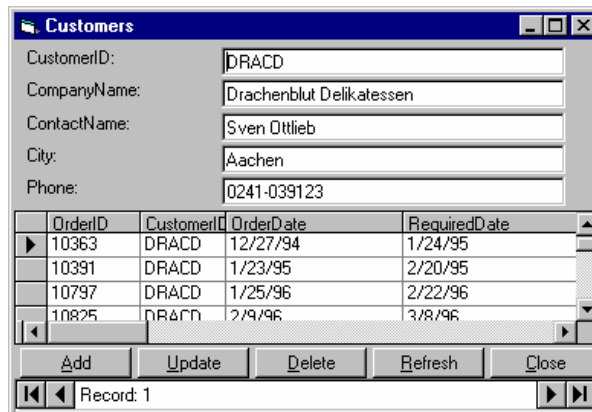


Figure 9.20 – The finished data form crated by the Data Form Wizard

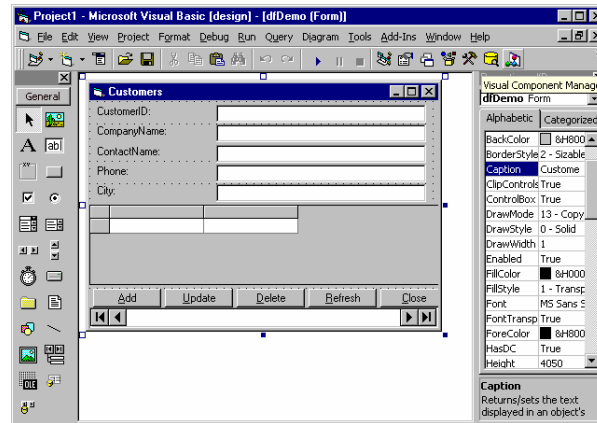


Figure 9.21– Using the Finished form to display information from the database

Using the data Control, switch to a new customer and you will see the order information change as well. By using the Data form Wizard, you can create data-aware applications with very little effort. However, after the Wizard has created the forms, you must maintain the form layout and its associated code. Finally, remember that each of the different layouts will require slightly different information to create its associated form.

9.8 Short Summary

- ☞ Visual Data Manager is used to create databases tables, queries and indexes and to modify them.
- ☞ Intrinsic Data control can be used to create applications that display, edit, and update information from many types of existing databases, including Microsoft Access, Btrieve, dBase, Microsoft FoxPro, and Paradox.
- ☞ To open already existing access database choose File/Open Database/Microsoft Access. And Select the database name in the Open Microsoft Access database dialog box

9.9 Brain Strom

1. What are the advantages of Visual Data Manager?
2. Explain briefly Visual Data Control

Lab Unit 9.0 (2 Real Hours)

1. Create a Data base with the following given fields using Data Visual Manager Student number, name, tamil, english, maths, physics, chemistry marks.

Lecture - 10

Data Access Object

Objectives

In this lecture you will learn the following

- ❖ About Data Access Object
- ❖ Understanding the object model of DAO
- ❖ Knowing about Microsoft Jet database engine locking

Lecture Unit - 10

- 10.1 Snap Shot
- 10.2 DAO Object Model
- 10.3 Accessing a database using DAO
- 10.4 Microsoft Jet Database Engine Locking
- 10.5 Short Summary
- 10.6 Brain Storm

Lab Unit 10 (2 Real Time Hrs)

10.1 Snap Shot

10.1.1 Data Access Objects

Data Access Objects (DAO) communicate with Microsoft Access and other ODBC-complaint data sources through the JET engine. They provide properties and methods that allow to perform all the operations necessary to manage such a system, including the ability to do the following figure 10.1

- ❖ Create databases
- ❖ Define tables, fields, and indexes.
- ❖ Establish relations between tables.
- ❖ Navigate and query the database, and so on.

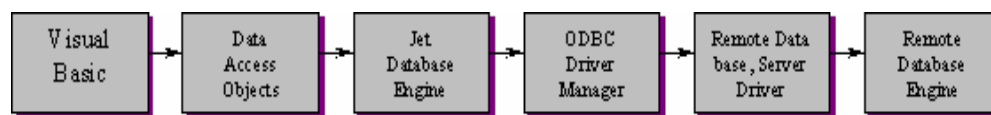


Figure 10.1 A typical remote data access using DAO and the JET engine.

The Data Access Objects (DAO) model is a collection of object classes that provide properties and methods for database programming. This model furnishes facilities for creating databases, defining tables, fields and indexes, establishing relations between tables, navigating and querying the database, and so on.

The Microsoft Jet database engine translates operations on data access objects into physical operations on the database files, handling all the mechanics of interfacing with the different supported databases. This approach simplifies access to the database and insulates from the underlying mechanics of retrieving and updating data. It affords great flexibility because the same objects, properties and methods can be used with a wide variety of supported database formats.

Through DAO and the JET engine, Visual Basic recognizes three categories of databases, and these are summarized below:

Category	Description
Visual Basic Databases (Native MS Jet Databases)	Also called native databases, these database files use the same format as Microsoft Access. These databases are created and manipulated directly by the Jet engine and provide maximum flexibility and speed.
External Databases	These are Indexed Sequential Access Method (ISAM) databases in several formats, including Btrieve, dBase III, dBase IV, Microsoft FoxPro version 2 and 2.5, and paradox versions 3.x and 4.
ODBC Databases	These include client/server databases that conform to the ODBC standard, such as Microsoft SQL Server and Oracle.

The DAO programming model provides the following features:

- ❖ Advanced resultset management.
- ❖ A universal programming model that can access any ODBC database regardless of the ODBC compliance level.
- ❖ Keyset, static, and forward-only scrolling snapshot cursor implementation.
- ❖ The ability to create updateable cursors against complex resultsets created as a join product.
- ❖ Universal error management.

When to use DAO

- ❖ DAO is the only data access technology that supports 16-bit operations. If an application must run within a 16-bit environment, then DAO is the only choice.
- ❖ If the application must access both native Microsoft Jet and ODBC resources, DAO provides a consistent programming model.

DAO Library

Visual basic includes DAO version 3.51. To access any of these objects, a reference to the DAO object library must be included. To include this reference choose Project | References. This will display References dialog box that resembles Figure 10.2

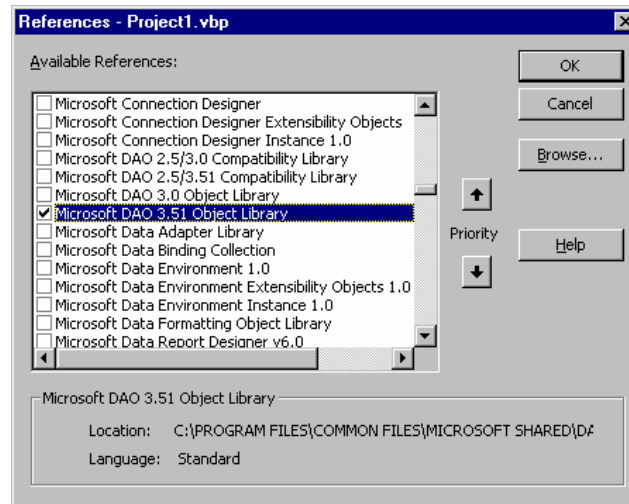


Figure 10.2 References dialog box.

- ❖ Select Microsoft DAO 3.51 object library and click OK. This will include a reference to DAO library to the current project.

10.2 The DAO Object Model

In DAO object model, Microsoft Jet database engine translates operations on data access objects into physical operations on the database files, handling all the mechanics of interfacing with the different supported databases.

Note : The Database engine is what actually reads, writes, and modifies the database, and handles all the housekeeping chores like indexing, and security etc.

The Figure 10.3 shows the DAO object model.

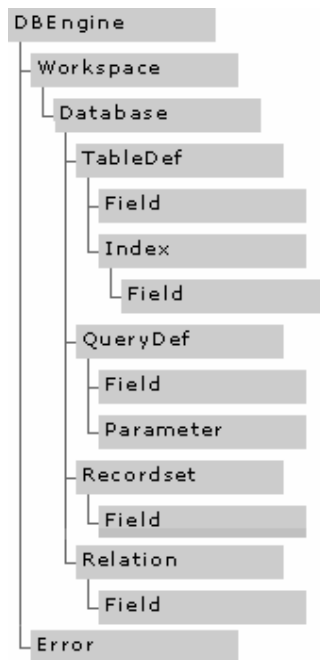


Figure 10.3 DAO Object model.

The DAO objects and their important method(s) are briefed below:

- ❖ **DBEngine object** This base DAO object holds all other objects and maintains engine options.

Method	Description
CreateWorkspace	Creates a new Workspace object.

- ❖ **Workspace object** Defines and manages the current user session. This object contains information on open databases and provides mechanisms for simultaneous transactions.

Method	Description
CreateDatabase	Creates a new Database object, saves the database to disk, and returns an opened Database object.
OpenDatabase	Opens a specified database in a Workspace object and returns a reference to the Database object that represents it.

BeginTrans	Begins a new transaction.
CommitTrans	Saves any changes and ends the current transaction.
Rollback	Cancels any changes made during the current transaction and ends the transaction.

- ❖ **Database object** Represents a database with at least one open connection. This can be a Microsoft Jet database or an external data source.

Method	Description
OpenRecordSet	Opens a recordset.
CreateTableDef	Creates a new TableDef object (Microsoft Jet workspaces only).
CreateQueryDef	Creates a new QueryDef object in a specified Connection or Database object.
CreateRelation	Creates a new Relation object (Microsoft Jet workspaces only).

- ❖ **TableDef object** Contains both Field and Index objects to describe the database tables.

Method	Description
CreateIndex	Creates a new Index object (Microsoft Jet workspaces only).
CreateField	Creates a new Field object

- ❖ **QueryDef object** Represents a stored SQL query statement, with zero or more parameters, maintained in a Microsoft Jet database.

Method	Description
Execute	Runs an action query or executes an SQL statement.

OpenRecordSet Creates a new Recordset object

- ❖ **Recordset object** Represents a query result set with a cursor. DAO has five types of Recordset objects: table, dynaset, snapshot, forward-only, and dynamic.

Manipulation Method	Description
AddNew	Creates a new record for an updatable Recordset object.
Edit	Copies the current record's contents to the buffer for editing.
Update	Saves any changes made to the current Record.
CancelUpdate	Cancels any changes made to the current record or to a new record prior to calling the Update method.
Delete	Deletes the current record

Navigation Method	Description
MoveFirst	Moves to the first record.
MoveNext	Moves to the next record.
MovePrevious	Moves to the previous record.
MoveLast	Moves to the last record.
Move	Moves the position of the current record based on its argument.
Search Method	
FindFirst	Moves to the first record that satisfies specified criteria.
FindNext	Moves to the next record that satisfies specified criteria.
Find Previous	Moves to the previous record that

satisfies specified criteria.

FindLast	Moves to the last record that satisfies Specified criteria.
-----------------	---

Miscellaneous Method	Description
Clone	Creates a duplicate Recordset

- ❖ **Relation object** Represents a relationship between fields in tables and queries.

Method	Description
CreateField	Creates a new Field object

- ❖ **Field object** Represents a field in a table, query, index, relation, or recordset. A Field object contains data, and it can be used to read data from a record or write data to a record.
- ❖ **Index object** Represents an index on a table in the database.

Method	Description
CreateField	Creates a new Field object

- ❖ **Parameter object** Represents a value associated with a QueryDef object. Query parameters can be input, output, or both.
- ❖ **Error object** Contains information about an error that occurred during a DAO operation. When more than one error occurs during a single DAO operation, each individual error is represented by a separate Error object.

10.3 Accessing a database USING DAO

A DAO-based application generally follows the logic given below while accessing a data source:

- ❖ **Create the workspace** Defines the user session, including user identification, password, and database type (such as Microsoft Jet or ODBC).
- ❖ **Open the database** Specifies a connection string for a particular Workspace object, with information such as data source name and database table.

- ❖ **Open the recordset** Runs an SQL query (with or without parameters) and populates the recordset.
- ❖ **Use the recordset** The query result set is now available to the application. Depending on the cursor type, the row data can be browsed and changed.
- ❖ **Close the recordset** Drops the query results and closes the recordset.
- ❖ **Close the database** Closes the database and releases the connection.

The following code shows how to open the database Students and to close it.

```

Dim ws as Workspace
Dim db as database

' Setting ws to default workspace
Set ws = DBEngine.Workspace(0)

' Setting db to Students database using workspace's opendatabase ' method
Set db = ws.OpenDatabase("Students.mdb")

' Some work with students database

' Closing students database
db.Close

' Closing workspace
ws.Close

```

The Database object

The example in the previous section showed that a **Database** object represents an open database. This object helps to:

- ❖ Use the **CreateTableDef** and **CreateRelation** methods to create tables and relations, respectively.
- ❖ Use the **CreateQueryDef** method to create a temporary query definition.
- ❖ Use the **OpenRecordset** method to open a table or to execute a select query and create a **Recordset** object.

Syntax :

Set rs = object.OpenRecordset (source, type)

Here rs is Recordset object. The source argument in the OpenRecordSet method is a string that specifies the source of records for the new recordset. The source can be a table name, a query name, or a query that returns records.

The argument *type* is a constant that indicates the type of **Recordset** to open, which can have any one of the following values.

Constant	Description
DbOpenTable	Opens a table-type Recordset object (Microsoft Jet workspaces only).
DbOpenDynaset	Opens a dynaset-type Recordset object. I.e, it specifies recordset type that can contain fields from more than one table.
DbOpenSnapshot	Opens a snapshot-type Recordset object. It similar to dynaset-type, except that it gives read-only records.
DbOpenForwardOnly	Opens a forward-only-type Recordset object. It specifies recordset in which navigation can take place in forward direction only.

Table 10.1 Type argument's options.

The following example shows how to open StudentDtIs table of Students database in two ways.

```

Dim db as Database
Dim rs as Recordset

' Setting database in short cut method
Set db = DBEngine.Workspace(0).OpenDatabase("Students.mdb")

' Opening the StudentDtIs table
Set rs = db.OpenRecordSet("StudentDtIs",dbOpenTable)

--- or ---

' Opening the StudentsDtIs table with selected records
Set rs = db.OpenRecordSet("Select * from StudendDtIs where_

```

```
TotalMarks > 450", dbOpenDynaset)
```

```
-- Some work with recordset rs
```

```
' Close RecordSet and Database
```

```
rs.Close
```

```
db.Close
```

10.4 Microsoft Jet Database Engine Locking

All data base management systems provide some sort of locking mechanism to prevent two users from updating data at the same time.

The Microsoft jet database engine provides page level locking. Pages are blocks of records that are 2048 bytes (2k) in size. Visual basic stores as many records as will fit on each page. When Visual Basic locks the page containing a record you're editing, all the other records on that page will also be locked .

Pessimistic Vs Optimistic Locking

The value of the **LockEdits** property of the **Recordset** object determines when a lock is placed on the records in a recordset.

Pessimistic locking

If the **LockEdits** property is **True**, pessimistic locking is in effect. The page containing the current record is locked as soon as you use the **edit** method. The page is unlocked when you use the **update** method.

The default locking strategy locks records for a longer period of time, but ensures that once the **Edit** method has been executed another user cannot change the data.

Optimistic locking:

If the **lockEdit** property is **False**, optimistic locking is in effect. The page containing the record is locked only while the record is being updated.

Locks are in place for a shorter period of time, and multiple users can use the **Edit** method without locking the page. However when you use optimistic locking, you'll need to handle possible errors when the user executes the **update** method.

If you do not expect users to attempt to modify the same record very often, consider setting **LockEdits** to **False**.

Unlocking

The Microsoft jet database engine marks a page to be unlocked as soon as the update. Completes. However, Locks are not removed until other actions are complete.

You can use the **Idle** method of the **DBengine** object with the `dbFreeLocks` options to suspend processing and allow the Microsoft jet database engine to release locks and catch up on other background tasks, as shown in the following code.

```
DBengine. Idle dbFreeLocks
```

Handling Locking Errors.

When multiple users are updating a database, you'll need to add code that traps for the following errors.

3260 - Couldn't update; currently locked

Occurs on the **Edit** method when the record is locked. Your code should wait for a short interval, and then call **Edit** again, or inform the user of the error.

3186 - Couldn't save; currently locked

Occurs on the **Update** method when the record is locked. Your code should wait for a short interval, and then call **Update** again or inform the user of the error.

3197 - Data has changed; operation stopped

Occurs on the **Edit** or **Update** method if another user has changed data since you last accessed it. Your code should refresh the form with the latest data or inform the user of the error.

For a complete list of errors available I Visual Basic, see **Trappable Errors** in Visual Basic.

10.5 Short Summary

- ☞ A DAO is a collection of object classes that model the structure of a relational database system

- ☞ When you access databases through ODBC, the locking mechanism is handled by the database management
- ☞ DAO is the only data access technology that supports 16-bit operations

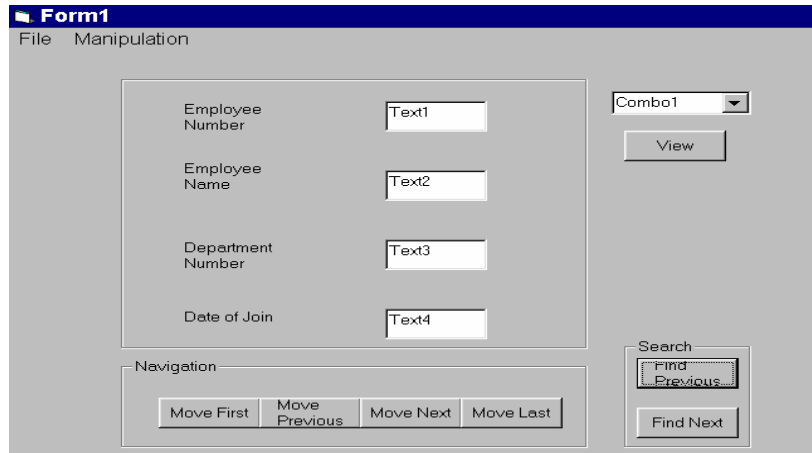
10.6 Brain Storm

1. What is the difference between connecting a database through DAO and data control
2. When DAO concept is introduced and state the features
3. Through DAO can we connect non database objects

Lab Unit 10 (2 Real Time Hrs)

1. Open a new project and add a form to it
2. Select →Project→References→ Microsoft Data Access Objects 3.5 library
3. Create a menu with the following items and sub items:
 - **File**
 - Open database
 - Open recordset
 - Open qryreset
 - **Manipulation**
 - Addnew
 - Edit
 - Update
 - Delete

Design your form as shown below:



4. Write code for each of the above mentioned sub menus to perform its corresponding task accordingly with the below given details:
 - Create a database by name **“Sample.mdb”** in Visual Data Manager in your Folder
 - Create a table under **“Sample.mdb”** as **“Emp”** with the following fields of mention datatypes:

Field Name	Data type
Eno	Integer
Ename	Text (30)
Deptno	Integer
Doj	Date

-
- Index on **“Deptno”** (Primary) field
 - Open recordset for **“emp”** table with dynaset as its type
 - After completing the above said task, try entering records, deleting records, editing records to the emp table
 - Use the Find methods to find the records for a given employee name
 - Use Move methods to navigate through the records
 5. In the Command Button named **“View”**, Open a record set to fetch only the **“Eno”** and add it on to **“Combo 1”**. Once when an **“Eno”** is selected from the Combo1, the corresponding Eno’s information should be shown in the text boxes.

Lecture - 11

Remote Data Control

Objectives

In this lecture you will learn the following

- ❖ Understand the concept of RDC
- ❖ Difference between MSRDC and Data control
- ❖ Know how to use the RDC in an application

Lecture unit - 11

- 11.1 Snap Shot
- 11.2 About Remote Data Control (RDC)
- 11.3 Remote Data Control vs Data Control
- 11.4 Short summary
- 11.5 Brain storm

Lab unit 11 (2 Real Time Hrs)

11.1 Snap Shot

The Remote Data Control is another way for accessing remote data in visual basic application. the user can interact data through the user interface controls like browsing the data, performing updates and adding records remote data control binds data-aware controls to an ODBC remote database, and the remote data object.

11.2 About Remote Data Control (RDC)

The Remote Data Control (RDC) binds controls to an ODBC remote database. The Remote Data Control is similar to the data control, except that it creates and manipulates RDO objects. RDO and RDC helps to access ODBC data sources through data -aware, or bound controls without going through the JET engine, resulting in significantly higher performance and greater flexibility when accessing remote data sources.

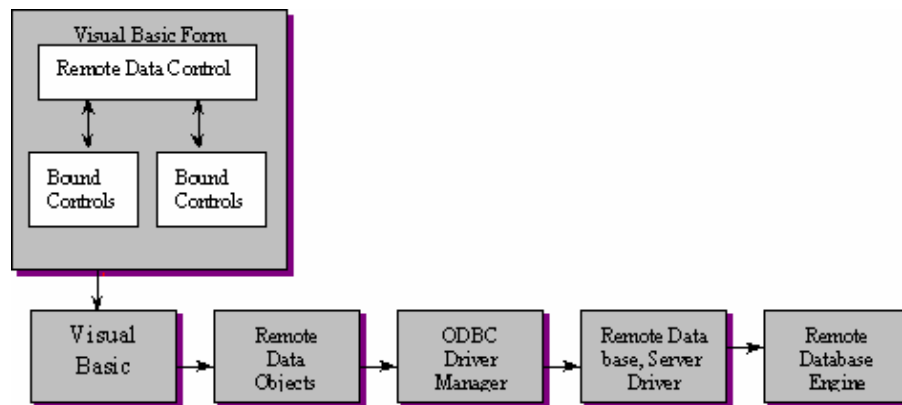


Figure 11.1 The Remote Data control, RDO and bound controls.

The Remote Data control performs all operations on the current row. The Remote Data Control automatically handles a number of contingencies, including empty resultsets, adding new rows, editing and updating existing rows, converting and displaying complex data types, and handling some types of errors.

The **RemoteData** control provides an interface between Remote Data Objects (RDO) and data-aware bound controls. With the RemoteData it is possible to:

- Establish a connection to a data source based on its properties.
- Pass the current row's data to corresponding bound controls.
- Permit the user to position the current row pointer.
- Pass any changes made to the bound controls back to the data source.

Using the Remote Data Control (RDC)

To use a Remote Data Control

- In Visual Basic, select Components item from Project menu
- From the list of Components, select Microsoft Remote Data Control 2.0. Also add a reference to the Microsoft Data Bound Grid Control
- Click OK. The Remote Data and Data bound Grid controls appear in the Visual Basic Toolbox
- Click the Remote Data Control from the tool box and add it to the form
- Also add a Data bound Grid Control to the form
- Change the Remote Data Control's User name & Password property to the user's choice
- Set the Remote Data Control's SQL property to Select * from emp
- In the Data source name property of the remote data control, select the DSN namely XYZ company for the SQL server database
- Set the DBGrid control's Datasource property to MSRDC1 which is the name of the Remote Data Control
- Execute the application

11.3 The Remote Data Control Vs Data Control

The RemoteData control is similar to the Visual Basic Data control in most respects. Both the Data control and the RemoteData control are designed to connect data-aware bound controls to a data source. The primary difference is that the RemoteData control uses RDO to connect to the ODBC driver manager, while the Data control uses DAO to connect to the Jet database engine, which can connect to native Jet databases or IISAM data sources. The Data control can also connect indirectly to RDO by using the ODBCdirect interface. Even though both controls use the same ODBC driver manager and data source entries, RDO, ODBCdirect, and DAO cannot share data source connections, data objects, or other resources.

Without a RemoteData control, a Data control, or its equivalent, data-aware (bound) controls can't automatically access data. You can perform many simple remote data access operations using the RemoteData control without writing any code at all.

However, most applications rely on additional RDO code to perform most data access operations, as RDO provides more flexibility and additional options not exposed by the RemoteData Control alone. In most respects, the RemoteData Control behaves very much like the Data control.

Data-aware controls bound to a RemoteData control automatically display data from one or more columns for the current row or, in some cases, for a set of rows on either side of the current row. The RemoteData control performs all operations on the current row.

If the RemoteData control is instructed to move to a different row, all bound controls automatically pass any changes to the RemoteData control to be saved by the ODBC data source. The RemoteData control then moves to the requested row and passes data back from the current row to the bound controls, where it is displayed.

11.4 Short Summary

- ☞ The Data control supports the ability to use the RDO DLLs via the ODBCdirect option. In this case, your code uses Data Access Objects (DAO) instead of RDO to manage the Data control and its result sets.
- ☞ RDC property can be set as an alternative to the Connect property
- ☞ The remote data control behaves like the Jet-driven data control and RDC is efficient than DC when dealing with the Remote Databases
- ☞ The Remote Data control's SQL property is like the Data control's Record Source property except that it cannot accept the name of a table

11.5 Brain Storm

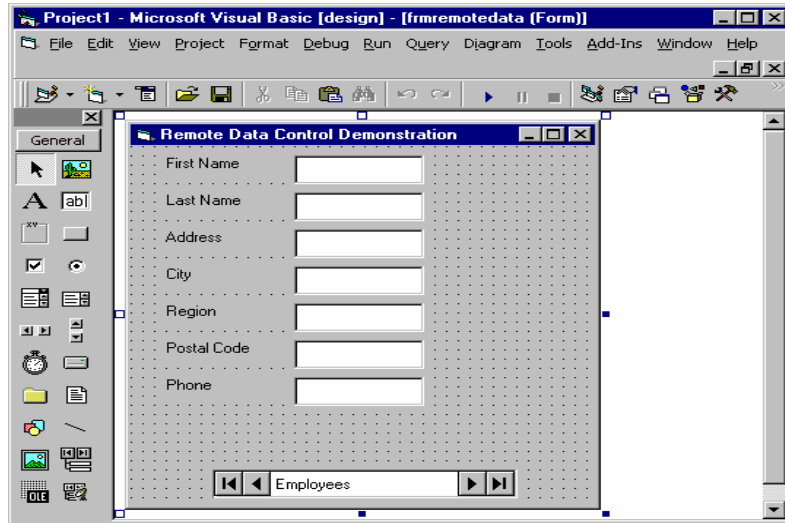
1. What is the default property in Remote Data Control ?
2. Difference between connecting a Back end through Remote Data Control and remote data object (coding)

Lab Unit 11 (2 Real Time Hrs)

Lab Unit - 11.1

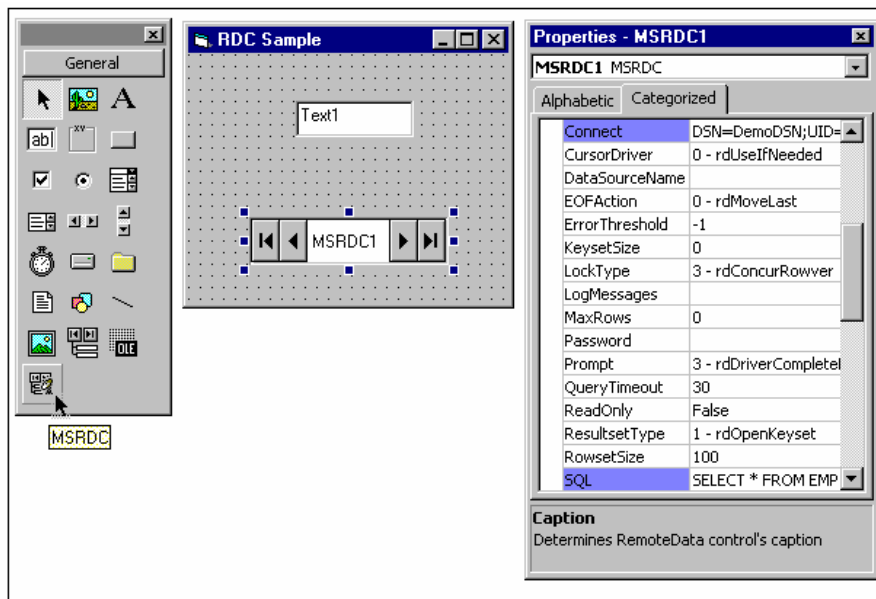
1. Open a New standard EXE project
2. Design your form as shown below
3. Create a Data Source named Trading Company

4. Link the data source(Trading Company) to this form. The data source can be got from the file Nwind.mdb in your VB folder



Lab Unit - 11.2

1. Open a New standard EXE project
2. Design your form as shown below



3. Create a Data Source named Authors
4. Link the data source(Authors) to this form. The data source can be got from the file Nwind.mdb in your VB folder
"SELECT * FROM AUTHORS"

Lecture - 12

Remote Data Objects

Objectives

In this lecture you will learn the following

- ❖ Understanding the concept of Remote Database objects
- ❖ Know how to Connect the remote database
- ❖ Manipulating the Database (inserting, deleting, updating)

Lecture Unit - 12

- 12.1 Snap Shot
- 12.2 Remote Data Object
- 12.3 RDO hierarchy
- 12.4 Short Summary
- 12.5 Brain Storm

Lab Unit 12 (2 Real Time Hrs)

12.1 Snap Shot

Remote Data Objects (RDO) and collections provide a framework for using code to create and manipulate components of a remote ODBC database system. RDO is Visual Basic's flagship interface to relational ODBC data sources.



Figure 12.1 A typical remote data access using RDO.

RDO is specifically designed to deal with remote intelligent data sources such as SQL server and Oracle. RDO can execute ordinary table-based queries, but it is especially good at building and executing queries against stored procedures.

The following advantages are associated with the RDO programming model:

- It is an universal programming model that can access any 32-bit level ODBC data source.
- All remote databases such as SQL Server and Oracle can virtually be accessed.
- Keyset, static, dynamic and forward only cursors are implemented on the server side.
- It gives the ability to create standalone rdoQuery and rdoConnection objects and to associate queries with connections at design time.
- It provides advanced resultset management, including the ability to limit the number of returned rows, and to handle the input, output and return value arguments of stored procedures.
- It provides universal error management that exposes not only ODBC error messages but also native errors and messages.

12.2 Remote Data Objects

RDO is a very thin (less than 250K) object layer on the top of the ODBC API. This makes it very fast but still easy to use. RDO does not have its own query processor, so all work is done by the ODBC datasource. This means no heterogeneous operations. RDO cannot connect to ISAM datasources; the Access ODBC driver must be used to connect to an Access or ISAM database. This approach is slower and does not have access to all the functionality provided by using DAO to access these types of datasources.

RDO does not have its own version of the data control called the remote data control, with abilities similar to the data control based on DAO. The remote data control can be used to generate applications fast without a lot of code.

RDO does require that its datasources have 32-bit Level II-compliant driver. Since RDO is a layer on top of the ODBC API, you can often use RDO in conjunction with the ODBC API for even more flexibility and power. Mixing and matching data access methods this way is considered risky, and Microsoft warns that general protection faults and data loss may result.

Though Remote Data Objects (RDO) are designed to work with ODBC, they can be used in similarly as DAO. Using the RDO

- Queries can be submitted.

- Result sets can be created.

- Forms containing the same bound controls recognized by the Data control can be created.

- And result set can be processed with little or no code.

Using RDO is very much like using DAO. However, there are some differences. These differences and similarities are given in below.

Remote Data Objects refer to table *rows* instead of *records* and *columns* instead of *fields* – the generally accepted terminology for relational databases. The data returned from a query is in the form of result sets, which can contain zero or more data rows composed of one or more columns.

DAO supports the creation and modification of the database schema, referential integrity (RI) by creating relations through DAO methods and properties. RDO does not support any type of RI or schema modification because they are fully supported in the tools and utilities provided with the server systems. However RDO methods are more useful while executing stored procedures.

The following table lists RDO 2.0 objects and their equivalent DAO/Jet objects.

RDO object	Equivalent DAO/Jet object
RdoEngine	DBEngine
RdoError	Error
RdoEnvironment	Workspace
RdoConnection	Database
RdoTable	TableDef
Not Implemented	Index
RdoResultset	Recordset
Not implemented	Table-type
Keyset-type	Dynaset-type
Static-type (r/w)	Snapshot-type (r/o)
Dynamic-type	(none)
Forward-only - type	Forward-only-type
(cursorless)	(none)
RdoColumn	Field
RdoQuery	QueryDef
RdoParameter	Parameter
Not Implemented	Relation

12.3 The RDO Hierarchy

Remote Data Objects (RDO) is a thin layer over the ODBC application-programming interface (API). RDO depends on the ODBC driver and the database engine for much of its functionality. Data access using RDO is intended for only ODBC relational databases.

The following Figure shows the RDO hierarchy

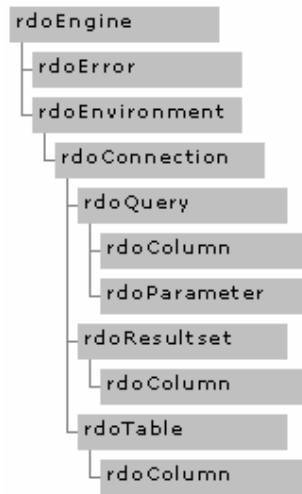


Figure 12.2 RDO hierarchy

The RDO objects and their important method(s) are briefed below:

rdoEngine object The base RDO object. This object is created automatically.

Method	Description
rdoCreateEnvironment	Creates a new rdoEnvironment object.

rdoError object Handles all ODBC errors and messages generated by RDO. This object is created automatically.

rdoEnvironment object Defines a logical set of connections and transaction scope for a particular user name. This object contains both open and allocated (but unopened) connections. This object is created automatically.

Method	Description
OpenConnection	Opens a connection to an ODBC data source and returns a reference to the rdoConnection object that represents a specific database.

rdoConnection object Represents either an open connection to a remote data source, or an allocated, but as yet unopened, connection.

Method	Description
OpenResultSet	Creates a new rdoResultSet object.
Execute	Runs an action query or executes an SQL statement that does not return rows.

rdoQuery object An SQL query statement with zero or more parameters.

Method	Description
OpenResultSet	Creates a new rdoResultSet object.
Execute	Runs an action query or executes an SQL statement that does not return rows.

rdoColumn object Represents a column of data, including data type and common properties.

rdoParameter object Represents a parameter associated with an **rdoQuery** object. Query parameters can be input, output, or both.

rdoResultSet object A set of rows returned from a query.

Method	Description
Requery	Updates the data in an rdoResultSet object by re-executing the query on which the object is based.

Other methods such as **AddNew**, **Update**, and **Move** etc. are same as their counterparts in **DAO**.

rdoTable object Represents the stored database definition of a table or view.

12.4 Using RDO to insert, update and delete

Records

Examples

CREATE A NEW PROJECT CALLED EMPLOYEE DETAILS . DESIGN YOUR FORM AS SHOWN BELOW

ADD THE FOLLOWING CODE IN THE DECLARATION SECTION OF THE FORM

```
DIM ENV AS RDOENVIRONMENT
```

```
DIM CN AS RDOCONNECTION
```

```
DIM RS AS RDORESULTSET
```

Enter the following code in the form_load event procedure

```
Set env = rdoenvironments(0)
```

```
env.Cursordriver = rdUseOdbc
```

```
Set cn = env.openconnection ( dsname:="",prompt:=rdodriverprompt,
```

```
Readonly:=false,connect:="")
```

```
Dim s as string
```

```
s = "select * from emp1"
```

```
set rs = cn.OpenResultset (Name:=s, Type:=rdOpenDynamic,_LockType:=rdConcrRowVer)
```

```
If rs.EOF <> True Then
```

```
Call Displayrecord
```

```
Else
```

```
MsgBox " No record found"
```

```
End If
```

```
End sub
```

When the user click the Add commandbutton, it clears the form and allows the entry of new information by the user.

```
Private Sub Add_Click()
```

```
Call.ClearRecord
```

```
rs.AddNew
```

```
End sub
```

The Update routine is used to save the changes that are made to the current record

```
Private Sub Update_Click ()
```

```
rs.edit
```

```
rs("dept_no") = txtdeptno.Text
```

```
rs("dept_name") = txtdeptname.Text  
rs.update  
end sub
```

When the user click the Delete command Button, the current record must be deleted from the table.

```
Private sub Del_click()  
rs.delete  
Call Displayrecord  
End sub
```

```
Private sub Displayrecord()  
Txtdeptno.Text = rs( dept_no)  
Txtdeptname.text = rs(dept_name)  
End sub
```

```
Private sub ClearRecord()  
Txtdeptno.Text = ""  
Txtdeptname.text = ""  
End sub
```

12.5 Short Summary

RDO is an object-oriented way of accessing client/server data sources. The `rdoQuery` object provided and method for creating and executing defined queries or views on the remote data source.

While handling more than one resultsets, RDO provides a method called "MoreResults".

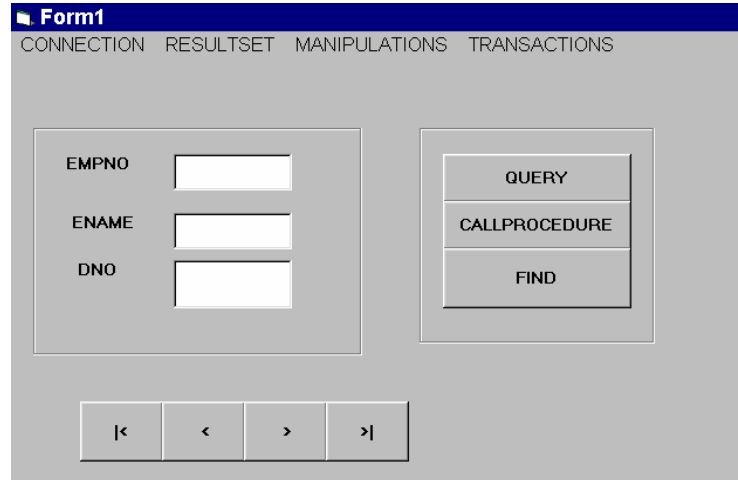
It can be used with `rdoResultset` object..

12.6 Brain Storm

1. Discuss various types of `rdoObjects`.
2. How do you insert the records using RDO?
3. State the features and advantages of RDO?.
4. How to connect a back-end table(Access) with vb form using RDO?.

Lab Unit - 12 (2 Real Time Hrs)

1. Create a new form and name it as **RDO** form.



2. Design the form as given below:
3. Refer the Microsoft Remote Data Objects 2.0 from Project/References.
4. Declare the variables RDO Environment, RDOConnection, RDO Resultset, RDO query.
5. Register a Data Source Name (DSN).
6. To Register the DSN follow the steps as mentioned below :

Start/Settings/ControlPanel/32bitODBC & select the Microsoft ODBC Driver for Oracle.
7. Using the Create Environment, Open Connection methods create the environment & open the connection respectively.
8. Use the Open Resultset method to display the records from the respective table in the textboxes.
9. Perform the manipulations such as addnew, edit & update with the respective coding addnew, update & edit methods.
10. Using the Create RDOQuery method try to insert a value.
11. Find a record based on the Empno using the RDO Query.
12. Write a code segment to call a procedure (Remote Procedure) that returns the summation of two numbers.

Lecture - 13

ActiveX Data Objects

Objectives

In this lecture you will learn the following

- ❖ About ActiveX Data Objects
- ❖ Working with ADO Data control
- ❖ Knowing about the ADO control events

Lecture Unit -13

- 13.1 Snap Shot
- 13.2 ActiveX Data Objects
- 13.3 Working with ActiveX Data Objects
- 13.4 ADO Object Model
- 13.5 ADO Data Control
- 13.6 ADO Data Bound Control
- 13.7 ADO Control Events
- 13.8 Short Summary
- 13.9 Brain Storm

Lab Unit 13 (2 Real Time Hrs)

13.1 Snap Shot

In the Cyber world new and new things are being invented to replace the already existing one. The Active X Data Objects provides very easy access to any data source with high speed via the OLE DB. And the ADO Data control allows to quickly create a connection to a database using ActiveX Data Object (ADO). By using a DAO-like approach, developers can now access an even border variety of data sources using both OLE DB service providers and existing ODBC drivers through its OLE DB for ODBC intermediate interface.

13.2 Active X Data Objects (ADO)

It is emerging as another data access alternative, which may replace the need for other interfaces. ADO enables to write a client application to access and manipulate data in a database server through a provider. ADO's primary benefits are ease of use, high speed, low memory overhead, and a small disk footprint.

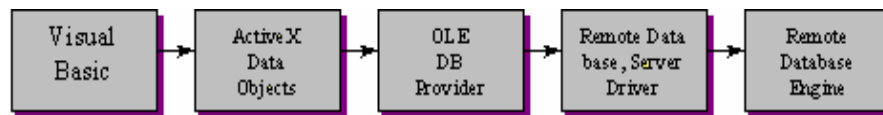


Figure 13.1 An ADO connection to a remote data source.

In ADO, the object hierarchy is de-emphasized. Unlike Data Access Objects (DAO) and Remote Data Objects (RDO), ADO allows creating objects independently; no longer having to navigate through a hierarchy to create objects. The ADO model also results in fewer objects, which allows for a smaller working set.

ADO is designed to eventually negate the need for all other interfaces. ADO is not specifically designed for ISAM or relational database access, but as an object interface to any data source. ADO is built around a set of core functions that all data sources are expected to implement.

The ADO programming model provides the following features:

- Advanced recordset cache management.
- Different cursor types, including the potential for support of back-end specific cursors.
- Independently created objects.

Support for stored procedures with in/out parameters and return values.
Support for limits on the number of returned rows and other query goals.

Visual basic also includes ADO Data Control (ADODC), which works in the similar manner to the Remote Data control, except that it uses OLE DB provider instead of ODBC Driver.

Advantages of ActiveX Data Objects

Flat objects Model: ADO does not impose hierarchical object creation. That is to create a Recordset object in DAO, a Workspace object and a Database object are to be created first. Then alone Recordset object can be created. In ADO either this kind of hierarchical creation can be followed or its objects can be created independently.

Less number of objects: ADO does not have large amount of objects like DAO and RDO, which leads difficulty in coding. ADO objects set is lesser than DAO and RDO, but more powerful.

Direct assignment to Data bound controls: To fill data bound controls its data source property can be set to a Recordset object instead of setting to a data control (explained later in this chapter).

Enhanced performance by Recordset Object: Indexed fields greatly enhance the performance of the **Recordset** object using **Find** method, and **Sort** and **Filter** properties.

Recordset persistence: Recordset data can be saved in a file. Later this persisted file can be used to recreate the RecordSet object.

13.3 Working with ActiveX Data Objects

The ADO works with OLE DB data providers to provide a mechanism for manipulating the data they contain. The ADO can be thought of as a specialized OLE DB data consumer. The relationship between the ADO and OLE DB is just like the relationship between RDO and ODBC. The ADO accesses OLE DB data sources, and when using the OLE DB ODBC data provider, the ADO can access ODBC data sources also.

Visual Basic 6.0, includes ADO version 2.0. To access any of these objects, a reference to the ADO object library must be included. To include this reference, Choose Project | References to display References dialog box.

Select Microsoft ActiveX Data Objects 2.0 Library in the Available References list and click OK.

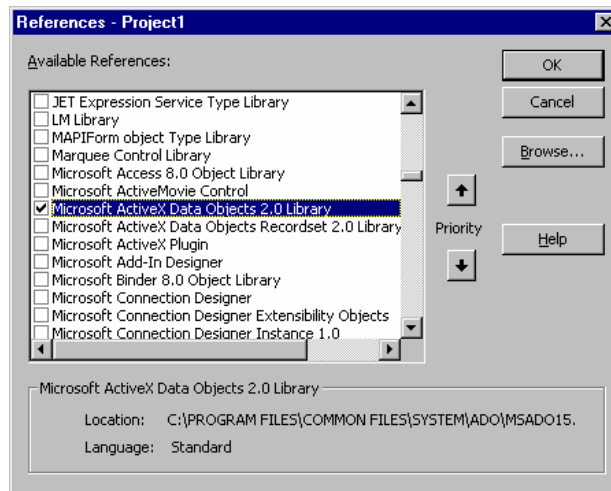


Figure 13.3 References dialog box.

13.4 The ADO Object Model

The ADO object model's hierarchy is given below.

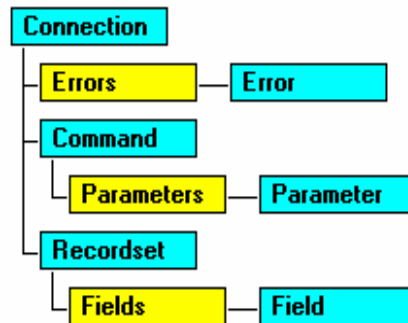


Figure 13.3 An ADO object model.

Each of the **Connection**, **Command**, **Recordset**, and **Field** objects also has a **Properties** collection.

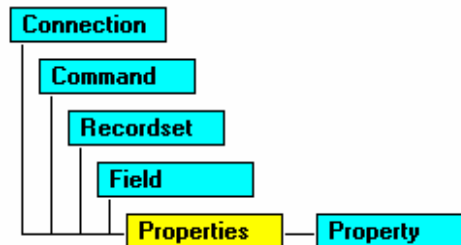


Figure 13.4 All ADO objects contains Properties collection

The Connection Object

It represents an open connection to a data source. A **Connection** object represents a unique session with a data source. In the case of a client/server database system, it may be equivalent to an actual network connection to the server. Depending on the functionality supported by the provider, some collections, methods, or properties of a **Connection** object may not be available.

The collections, methods, and properties of a **Connection** object help to:

- Configure the connection before opening it with the **ConnectionString**, and **Mode** properties.
- Set the default database for the connection with the **DefaultDatabase** property.
- Specify an OLE DB provider (explained later in this chapter) with the **Provider** property.
- Establish, and later break, the physical connection to the data source with the **Open** and **Close** methods.
- Set the location of the cursor using **CursorLocation** property which can have any one of the following value:
 - **adUseClient** - Uses client-side cursors supplied by a local cursor library.
 - **AdUserServer**- Default. Uses data-provider or driver-supplied cursors.
These cursors are sometimes very flexible and allow for additional sensitivity to changes others make to the data source.

The Command Object

It is used to query a database and return records in a Recordset object, to execute a bulk operation, or to manipulate the structure of a database. Depending on the functionality of the provider, some Command collections, methods, or properties may generate an error when referenced.

The collections, methods, and properties of a Command object help to:

- ☞ Define the executable text of the command (for example, an SQL statement) with the **CommandText** property.
- ☞ Execute a command and return a **Recordset** object if appropriate with the **Execute** method.

- ☞ Specify the type of command with the **CommandType** property prior to execution to optimize performance, such as `adCmdText`, `adCmdTable`(explained later in this chapter).
- ☞ Manage transactions on the open connection, with the **BeginTrans**, **CommitTrans**, and **RollbackTrans** methods. These methods are briefed below and used in the example code listing at end of this chapter.
 - **BeginTrans** begins a new transaction.
 - **CommitTrans** saves any changes and ends the current transaction.
 - **RollbackTrans** cancels any changes made during the current transaction.

The Error Object

Any operation involving ADO objects can generate one or more provider errors. As each error occurs, one or more **Error** objects are placed in the **Errors** collection of the Connection object. When another ADO operation generates an error, the **Errors** collection is cleared, and the new set of **Error** objects is placed in the **Errors** collection.

The properties, which give details about each error, are:

The **Description** property, which contains the text of the error.

The **Number** property, which contains the **Long** integer value of the error constant. The **Source** property, which identifies the object that raised the error. This is particularly useful when there are several **Error** objects in the **Errors** collection.

Each error object in the Errors collection can be accessed separately. The following example shows how to display the description of all created error objects.

Example:

```
Dim errobj As ADODB.Error
Dim con As ADODB.Connection

For Each errobj In con.Errors
    Debug.Print errobj.Description
Next
```

The Field Object

A Recordset object has a Fields collection made up of **Field** objects. Each **Field** object corresponds to a column in the **Recordset**. The **Value** property can be used to set or return data depending on the functionality the provider exposes, some collections, methods, or properties of a **Field** object may not be available.

The collections, methods, and properties of a **Field** object help to:

- Get the name of a field with the **Name** property.

- View or change the data in the field with the **Value** property.

- Get the basic characteristics of a field with the **Type**, **Precision**, and **NumericScale** properties.

- Get the declared size of a field with the **DefinedSize** property.

- Get the actual size of the data in a given field with the **ActualSize** property.

The Parameter Object

It represents a parameter or argument associated with a **Command** object based on a parameterized query or stored procedure. Using this, a desired action can be defined once, and variables (or parameters) are used to alter some details of the command. For example, an SQL SELECT statement could use a parameter to define the matching criteria of a WHERE clause, and another to define the column name for a SORT BY clause.

The collections, methods, and properties of a **Parameter** object help to:

- Set or return the name of a parameter with the **Name** property.

- Set or return the value of a parameter with the **Value** property.

- Set or return parameter characteristics with the **Attributes** and **Direction**, **Precision**, **NumericScale**, **Size**, and **Type** properties.

The Property Object

It represents a dynamic characteristic of an ADO object that is defined by the provider.

ADO objects have two types of properties: **built-in** and **dynamic**.

Built-in properties are those properties implemented in ADO and immediately available to any new object, using the `SomeObject.Property` syntax.

They do not appear as **Property** objects in an object's Properties collection. For example, the connection object has a built-in property `ConnectionString` and it can't be accessed in the hierarchy `con.Properties(0).ConnectionString`.

Dynamic properties are defined by the underlying data provider, and appear in the **Properties** collection for the appropriate ADO object. For example, a property specific to the provider may indicate, if a `Recordset` object supports transactions or updating. These additional properties will appear as **Property** objects in that **Recordset** object's **Properties** collection. Dynamic properties can be referenced only through the collection, using like `MyObject.Properties(0)` or `MyObject.Properties(name)` syntax.

A dynamic **Property** object has four built-in properties of its own:

- The **Name** property is a string that identifies the property.
- The **Type** property is an integer that specifies the property data type.
- The **Value** property is a variant that contains the property setting.
- The **Attributes** property is a long value that indicates characteristics of the property specific to the provider.

The Recordset Object

A **Recordset** object represents the entire set of records from a base table or the results of an executed command. At any time, the **Recordset** object refers to only a single record within the set as the current record.

The collections, methods, and properties of a **Parameter** object help to:

- Get number of rows in the current recordset object using **RecordCount** property.
- Limit the number of records, the provider returns from the data source using **MaxRecords** property.
- Get the editing status of the current record using **EditMode** property. This property may have one of the following values:

-
- | | | |
|---------------------------|---|--|
| 1. adEditNone | - | Indicates that no editing operation is in process |
| 2. adEditInProcess | - | Indicates that current record is modified but not yet saved. |

- | | | |
|------------------------|---|---|
| 3. adEditAdd | - | Indicates that Addnew method is called but the new record is not yet saved. |
| 4. adEditDelete | - | Indicates that the current record is deleted. |
-

To Save changes to the current record **Update** method is used. In the ADO model to change the data of the current, there is no need to call Edit method (actually the Edit Method is not the Method of Record Set object of ADO) as in DAO and RDO.

Sort and Filter the rows of a RecordSet object using **Sort and Filter** properties. These properties are briefed below and used in the example.

Sort - This property determines the order in which rows of a **Recordset** are traversed.

Filter-This property determines which rows are accessible when traversing rows.

Example:

' Assuming rs is a object of Recordset..

' rs contains a field StudentName

' Sorting on StudentName field

rs.Sort = "StudentName"

' rs contains a field TotalMarks

' Filtering records whose TotalMarks field value is

' gertter than 450

rs.Filter = " Totalmarks > 400 "

- ❖ Find a particular record by specifying criteria using Find method.
- ❖ Refresh the entire contents of a **Recordset** object from the data source using **Requery** method.

Example:

Dim rs as ADODB.RecordSet

```
-----
rs.AddNew
-----
```

```
rs.Update
```

```
' Retrieving the current data by requering the recordset.
```

```
rs.Requery
```

ADO Methods

This section explains important ADO methods.

The Open Method (adoConnection)

Opens a connection to a data source.

Syntax:

```
connection.Open ConnectionString, UserID, Password ,options
```

<i>Parameters</i>	<i>Description</i>
ConnectionString	Optional. A String that contains the arguments used to establish a connection to a data source such as DSN, SERVER etc., which are same as used in RDO.
UserID	Optional. A String containing a user name to use when establishing the connection.
Password	Optional. A String containing a password to use when establishing the connection.

Example:

```
Dim con1 as New ADODB.Connection
Dim con2 as New ADODB.Connection
Dim con3 as New ADODB.Connection
Dim con4 as New ADODB.Connection
```

```
' Opening a connection without using a Data Source Name
```

```
' (DSN).
```

```
Con1.Open "Driver={Microsoft ODBC for Oracle};_
```

```
server=Or8;uid=sa;pwd=pwd"
```

' Opening a connection using a DSN and ODBC tags.

```
Con2.Open "DSN=MyDSN;UID=sa;PWD=pwd;"
```

' Opening a connection using a DSN and OLE DB tags.

```
Con3.Open "Data Source=MyDSN;User ID=sa;Password=pwd;"
```

' Opening a connection using a DSN and individual

' arguments instead of a connection string.

```
Con4.Open "MyDSN","sa","pwd"
```

Here the connection object is declared with the keyword `New` in the declaration. All the ADO objects must be declared in similar manner. Because the ADO is based on Component Object Model technology (COM). So they can't simply be declared. They must be created using new keyword.

If ADO objects are declared without the `New` keyword then before using them, they must be set as new objects. The following code shows how the connection object is set as new connection object after it is declared without new keyword.

```
Dim cnn As ADODB.Connection
```

```
Set cnn1 = New ADODB.Connection
```

```
cnn.ConnectionString = "DSN=MyDSN;UID=sa;PWD=pwd;"
```

```
cnn.Open
```

In all the above examples the Provider argument is not specified. So, while opening the connection the default provider parameters are used. The following section explains about providers and their classifications.

13.5 ADO Data Control

The ADO Data control uses Microsoft ActiveX Data Objects (ADO) to quickly create connections between data-bound controls and data providers (any controls that feature a `DataSource` property). Data providers can be any source written to the OLE DB specification.



Figure 13.5 The Ado Data Control

Creating a sample data base application with ADO data control

This section describes steps involved in creating a sample data base application with ADO data control. The behavior of this application will be similar to those applications, which are created using data control, and RDC in the previous chapters. But the different lies in the design process. The following steps explain to design a simple application with the ADO data control.

To create a simple database application that uses ADO Data Control

Draw an **ADO Data Control** on a form. (The icon's ToolTip is "ADODC.")

If the control is not available in the Toolbox, press CTRL+T to display the **Components** dialog box. In the **Components** dialog, click **Microsoft ADO Data Control**.

On the Toolbox, click the **ADO Data Control** to select it. Then press F4 to display the **Properties** window.

In the **Properties** window, click **ConnectionString** to display the **ConnectionString** dialog box.

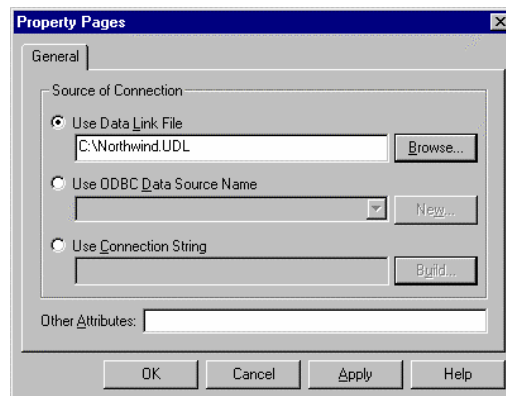


Figure13.6 Property pages of ADO data control.

The connection string can be specified in any one of the following ways:

- If a Microsoft Data link file (.UDL) is available for the require database, select **Use OLE DB File** and click **Browse** to find the file (NorthWind.UDL) on the computer.
- If using DSN is desired, then **Use ODBC Data Source Name** and select a DSN from the combo box.

- Or, to create a connection string, select **UseConnectionString**, and then click **Build**, and use the **Data Link Properties** dialog box to create a connection string (in the same way as used in creating Data link file). After creating the connection string, click **OK**. The **ConnectionString** property will be filled with a string like:

Provider=MSDAORA.1;User Id= Ragu;Data Source=Or8;_
Persist Security Info=False

To set the **RecordSource** property click the button with the caption "..."



A dialog box will appear as in the following Figure:

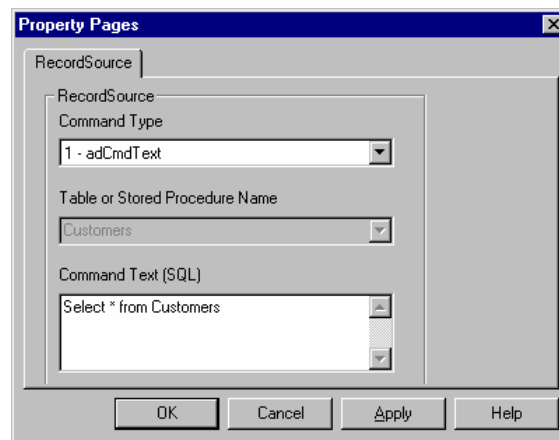


Figure 13.7 Record Source's Property page.

- ❖ Select *adCmdTable* in **CommandType** box and *Customers* in **Table or Stored Procedure** name box.
--Or--
Select *adCmdText* in **CommandType** box and type "*Select * from Customers*" in **Command Text (SQL)** box.
- ❖ Draw a **TextBox** control on the form to display the database information.
- ❖ In the **Properties** window, set the **DataSource** property for Text1 to the name of the ADO Data control (ADODC1). This binds the text box to the ADO Data control.
- ❖ In the **Properties** window, click **DataField** and a list of available fields will drop down. Click the name of the field, which is desired for display.
- ❖ Repeat last three steps for each additional field.
- ❖ Run and examine the application.

13.6 ADO Data bound controls

Microsoft supplied new data bound controls with Visual Basic 6.0 namely DataGrid, DataList, and DataCombo. They are ADO versions of DBGrid, DBList and DBCombo respectively, and can be used in the same way of later. But the ADO version of data bound controls have one advantage. So far data bound controls are filled up using a data control. Now instead of data controls, recordset objects can behave as data supplier to the data bound controls. The following example shows how a recordset object can supply data to a Data bound grid control.

Example - binding controls using Recordset :

```
Dim rs as New ADO.Recordset
' The data source name AccessDSN contains the Nwind
' database
' Opening the recordset

rs.Open "Select * from Products", "DSN=AccessDSN", _
        adOpenKeyset

' Assuming the form contains Data Grid control with the
' name DataGrid1...
' Setting rs to DataGrid's Datasource property
Set DataGrid1.DataSource = rs
```

The above statement will cause the DataGrid1 be filled up by the products table of Nwind database.

There is one more data bound control, the Hierarchical Grid control, which has more flexibility to display the data.

13.7 ADO Control Events

ADO Data Control includes a number of events that are user programmable. The events listed in Table lists the events and when they occur.

ADO Data Control's Programmable Events

Events	Occurs
WillMove	On Recordset.Open,

Recordset.MoveNext	Recordset.Move
Recordset.MoveLast	Recordset.MoveFirst,
Recordset.MovePrevious	Recordset.Bookmark,
Recordset.AddNew	Recordset.Delete,Recordset.Requery Recordset.Resync
MoveComplete	After WillMove
WillChangeField	Before the Value property changes
FieldChangeComplete	AfterWillChangeField
WillChangeRecord	On Recordset.Update,Recordset.Delete, Recordset.CancelUpdate,Recordset.UpdateBatch Recordset.CancelBatch
RecordChangeComplete	After WillChangeRecord
WillChangeRecordset	On
Recordset.Requery,Recordset.Resync,Recordset.Close	Recordset.Open,Recordset.Filter
RecordsetChangeComplete	After WillChangeRecordset
InfoMessage	When the data provider returns a result.

13.8 Short Summary

- ☞ Active X Data Object is the object based interface to OLEDB.
- ☞ ADO uses a new database connection framework called OLEDB, which allows us faster, more flexible access to multiple data providers. ADO wraps it all into one ease-to-use interface.
- ☞ If requery method is called in the middle of editing or adding new record, i.e before calling Update method, an error occurs. After issuing the Update method, the Requery method must be called to reflect the changes.
- ☞ If user and password information are specified both in the ConnectionString argument and in the optional UserID and Password arguments, the UserID and Password arguments will override the values specified in ConnectionString.

13.9 Brain Storm

1. What are the features of ADO?
2. In which version onwards, ADO Concept is available ?
3. Can We connect a text file with a front end using RDO? If yes, explain.
4. How many types of result sets are available in ADO?
5. What is the mode we use in ADO?
6. What are data consumers, which support ADO?

Lab Unit - 13 (2 Real Time Hrs)

1. Add a New Project
2. Design the Form as shown below

The screenshot shows a Visual Basic form titled "Form1" with a menu bar containing "Connection", "Recordset", and "Manipulations". The form is divided into three sections:

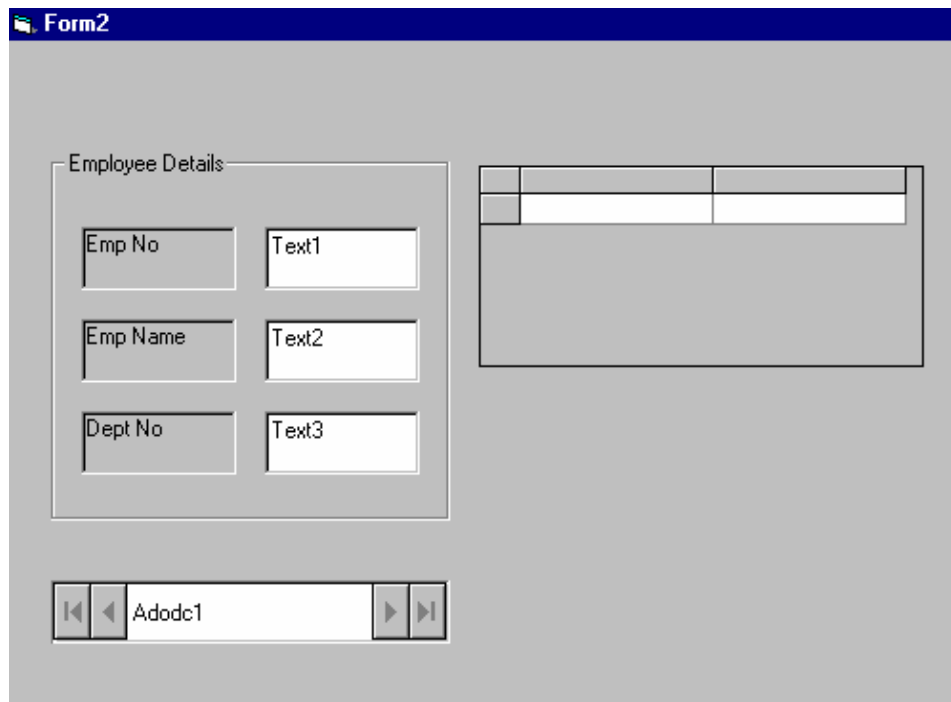
- Employee Details:** A group box containing three pairs of text boxes. The first pair is labeled "Emp No" and "Text1", the second "Emp Name" and "Text2", and the third "Dept No" and "Text3".
- Queries:** A group box containing two buttons: "Insert" and "Call Procedure".
- Navigation:** A group box containing four buttons: "|<", "<", ">", and ">|".

3. Invoke ActiveX Data Object Library File
Project → References → MS ActiveX Data Object 2.0

Establish Connection

4. Open Recordset and display records in the Text boxes Perform Data Manipulations like Addition, Deletion, Modification Perform Navigation methods
5. Try using Action Queries with the help of Command Object Using Command Object, try executing a Remote Procedure that gives the sum of two numbers
6. Add a Form
7. Choose Microsoft ADO Data Control
Project → Components → Microsoft ADO Data Control 6.0 (OLE DB)

8. Design your Form as shown below



9. Set the following properties of ADODC1
 - Connect String
 - User Name
 - Password
 - Record Source
10. Set the following properties for your Text boxes to display the fields
 - Data Source
 - Data field

Invoke MS DataGrid Control 6.0 (OLEDB)

11. Set the following property of MS Data Grid Record Source
12. Now try data manipulations through the MS Data Grid, by setting the Design time properties.

Lecture - 14

Connecting to a ADO Database

Objectives

In this lecture you will learn the following

- ❖ Establishing ADO Database connection
- ❖ Knowing about BOF & EOF properties

Lecture Unit - 14

- 14.1 Snap Shot
- 14.2 Creating an ADO Database with a Front End
- 14.3 BOF & EOF properties
- 14.4 Short Summary
- 14.5 Brain Storm

Lab unit 14 (2 Real Time Hrs)

14.1 Snap Shot

14.1.1 Creating an ADO Database with a Front End

Using the ADO Data control in your applications is no more difficult than using the standard DAO Data control. In the following example, you will create a sample front end for an ADO database. Although this application is very simplistic, it does illustrate how easy it is to create an application using ADO. To create the front end for the Database, perform the following steps:

1. Confirm that you have a valid .MDL file if you will be using an OLE DB Data source.
2. Start a new project. On the form, place an ADO control. If the ADO control is not available in your toolbox, you can add it from the Components dialog box and select the Microsoft ADO Data Control option shown in Figure 14.1

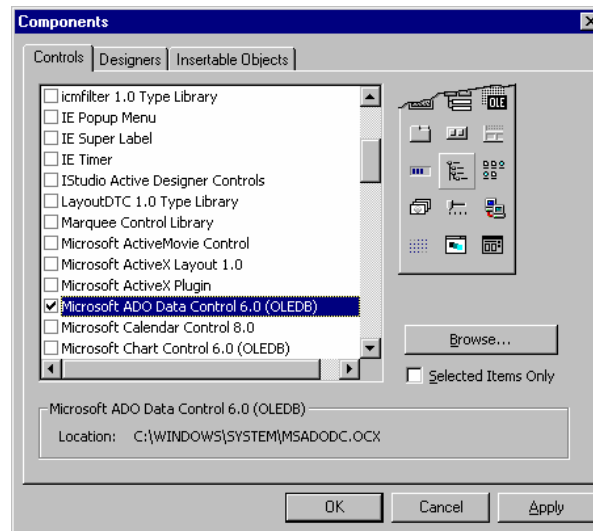


Figure 14.1 – The ADO Data control is added to the toolbox after the Components dialog box is closed

3. In the Properties window of the ADO control, click the ellipsis button on the ConnectionString property to display the ConnectionString dialog box. This is shown in Figure 14.2

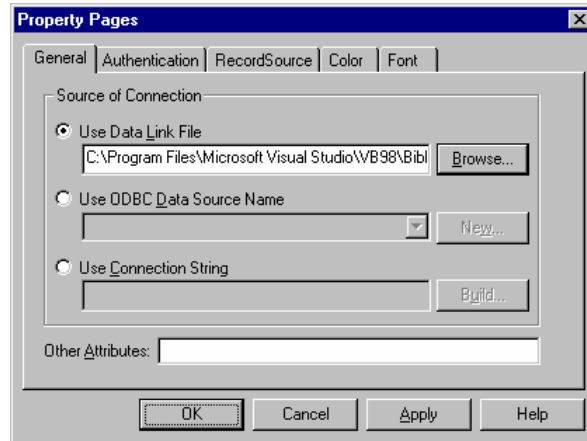


Figure 14.2 : Property pages allow you to enter the ODBC source and connecting string information

4. Select the General tab. If you have created a Data Link file (.MDL), select the Use Data Link File check box and click the Browse button to locate the .MDL file. Alternatively, you could select either the Use ODBC Data Source Name or Use Connection String check boxes to create a connection to your database.
5. In the properties window of the ADO control, click the ellipsis button to display the Record Source dialog box. In the Command Text(SQL) text box, enter the following SQL statement:

```
SELECT * FROM Authors WHERE Au_ID = 72
```

This sets the RecordSource property to a SQL statement. This is shown in

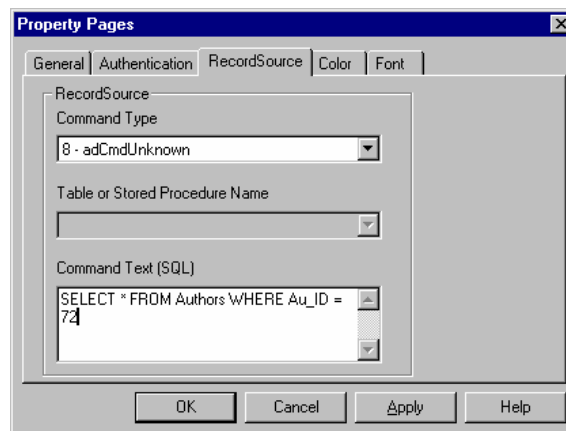


Figure 14.3- SQL statements can also be enter via the RecordSource tab on the Property Pages dialog box

6. Place two labels and two text boxes on the form. The captions of the text boxes should be Author and Author ID

7. Set the DataSource property for Text1 and Text2 to the name of the ADO control on the form.
8. In the properties window of Text1, set the DataField property to Author
9. Set the DataField property of Text2 to Au_ID
10. Press F5 to run the application. The Completed form should resemble the one in Figure 14.4

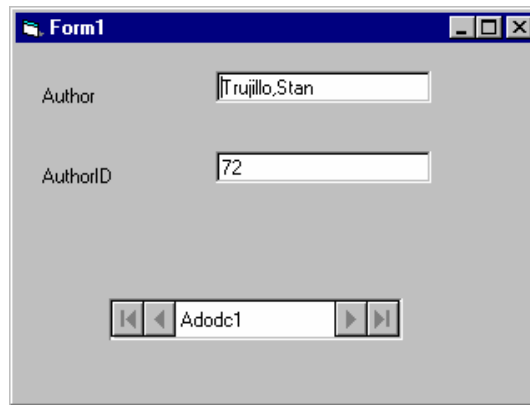


Figure 14.4: The Complete ADO Sample

The ADO Data Control can be major resource hog. The ADO control will automatically use at least two connections for the first ADO control on a form. Additional ADO controls on the same form require one connection each.

An alternative to using the ADO data control is to set the properties programmatically. The sample code segment is listed below.

Example

```
Private Sub Form_Load()
    With ADODC1
        .ConnectionString = "driver={SQL Server};" & _
            "server=orion;uid=sa;pwd=pwd;database=pubs"
        .RecordSource = "Select * From Authors Where Au_ID = 72"
    End With
    Set Text1.DataSource = ADODC1
    Text1.DataField = "Author"
End Sub
```

14.1.2 Connecting to a Database

ADO can be used to connect with a OLE DB datasource and non data base types also

A sample code is given to connect with SQL Server database

```
Dim con as ADODB.connection  
Dim cmd as string  
Cmd = "Provider = SQLOLEDB.1;"
```

14.2 BOF and EOF properties

- ❖ **BOF** (Beginning Of File): This property returns a Boolean value that indicates whether the current record position is before the first record in a **Recordset** object.
- ❖ **EOF** (End Of File): This property returns a Boolean value that indicates whether the current record position is after the last record in a **Recordset** object.
- ❖ The **BOF** property returns **True** if the current record position is before the first record, and **False** if the current record position is on or after the first record.
- ❖ The **EOF** property returns **True** if the current record position is after the last record, and **False** if the current record position is on or before the last record.

The BOF and EOF properties can be used to determine whether a Recordset object contains records or whether the cursor has gone beyond the limits of a RecordSet object when one of a Move method is used to move the record pointer from record to record.

If both the **BOF** or **EOF** property is **True**, it means there is no current record. If an attempt is made to move beyond the limits of a RecordSet object then an error will occur. So these properties can be used in conjunction with the move methods. For example, to prevent the move operation beyond the last record the following code may be used.

```
rs.MoveNext  
If rs.EOF Then  
    MsgBox "End of the file! , Pointer being moved to last_ record."  
    rs.MoveLast  
/End If
```

At the end of the RecordSet rs, its EOF property is set to True. So, as the if condition satisfies the MoveNext method will be roll backed. Similarly the BOF property can be used with the MovePrevious method.

14.3 Short summary

- ☞ OLE DB is a low-level object base programming interface designed to access a wide variety of data source
- ☞ ADI can be implemented using three tools
 - ADO Connection object
 - ADO Data control
 - ADO Recordset object

14.4 Brain storm

1. State the properties of BOF?
2. What is the main technology behind ADO?
3. Why ADO is beneficial than RDO?
4. Why we go for less cursor drives in ADO?

Lab Unit - 14 (2 Real Time Hrs)

1. Consider a Railway database in SQL Server with three tables namely Passenger_details, Train_Details and Train_Pass_Details with fields given below

Passenger Details

Passenger Id, Passenger Name, Age, Sex, Train name, Class booked

Train Details

Train name, Train no, I class fare, II class fare, Unreserved fare, I seats, II seats

Train Pass Details

Passenger id, Coach No, seat no, Date of journey

2. Create a project to list the passenger ids in the combobox and to list the details of the passenger (Train number, seat number and ticket fare)

3. Design your form as shown below

The screenshot shows a Windows-style application window titled "Railway Ticket Chart". The form is divided into two main sections: "List Passenger Details" and "Train Particulars".

List Passenger Details:

- Passenger ID:
- Passenger Name:
- Age:
- Gender:
- Class [I,II,III]:
- A dropdown menu next to the Age field is set to "Psg000001".

Train Particulars:

Train No	Train Name	Seat No	Journey Date	Ticket Fare
<input type="text" value="S5"/>	<input type="text" value="Nellai Express"/>	<input type="text" value="56"/>	<input type="text" value="07/07/2k"/>	<input type="text" value="124.50"/>

Lecture - 15

Multiple Document Interface

Objectives

In this lecture you will learn the following

- ❖ About Multiple Document Interface
- ❖ Introduction to MDI form
- ❖ About MDI child form property

Lecture Unit - 15

- 15.1 Snap Shot
- 15.2 Multiple Document Interface
- 15.3 Introduction to MDI form
- 15.4 MDI child forms property
- 15.5 Short Summary
- 15.6 Brain Storm

Lab unit 15 (2 Real Time Hrs)

15.1 Snap Shot

A Multiple Document Interface is used for opening many windows at the same time. All the document windows are contained in a parent window, which provides a workspace in the application. Visual Basic applications can have only one MDI form, which contains all the child forms. A child form is an ordinary Form that has its Child property set to True. Child forms are displayed within the internal area of a MDI Form at run time.

15.2 Multiple Document Interface

Mainly there are two styles of interfaces in the windows based applications: the *single-document interface (SDI)* and the *multiple-document interface (MDI)*. An example of the SDI interface is the WordPad application included with Microsoft Windows. In WordPad, only a single document may be open and in order to open another, the currently opened document must be closed. The Figure 5.12 shows the Wordpad application, which is a SDI application.

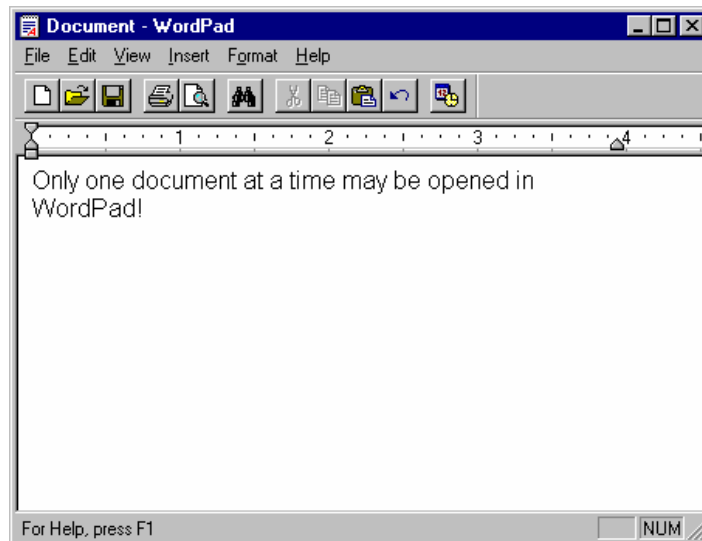


Figure 5.1 WordPad, a single-document interface (SDI) application.

Applications such as Microsoft Excel and Microsoft Word for Windows are MDI interfaces; they allow displaying multiple documents at the same time, with each document displayed in its own window. A MDI application can be recognized by the inclusion of a Window menu item with submenus for switching between windows or documents. The Figure 5.2 shows the Excel, which is a MDI application.

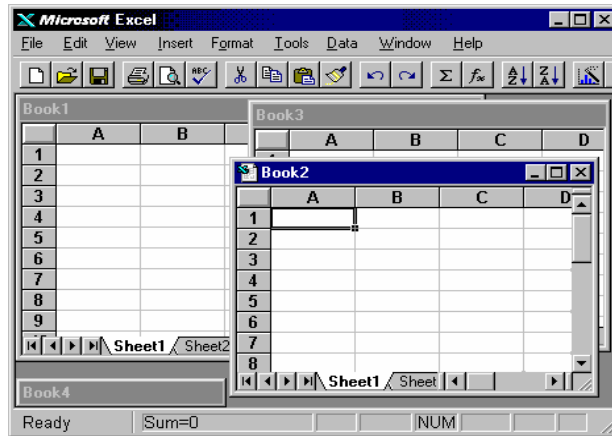


Figure 5.2 Microsoft Excel, a multiple-document interface (MDI) application.

In Multiple Document Interfaces, documents or *child windows* are contained in a *parent window*, which provides a workspace for all the child windows in the application. For example, Microsoft Excel allows to create and display multiple-document windows of different types. Each individual window is confined to the area of the Excel parent window. When the Excel is minimized, all of the document windows are minimized as well; only the parent window's icon appears in the task bar.

Menu

Any program with more than a few simple functions or features can benefit from the addition of a well-built menu. When designing a program, the main goal is to make its features as easy as possible. A well-designed menu will accomplish this goal. With the help of good menu, programs will seem natural and familiar, as they offer a convenient and consistent way to group commands and an easy way for users to access them. The Figure 5.3 illustrates the elements of a menu interface on a Form.

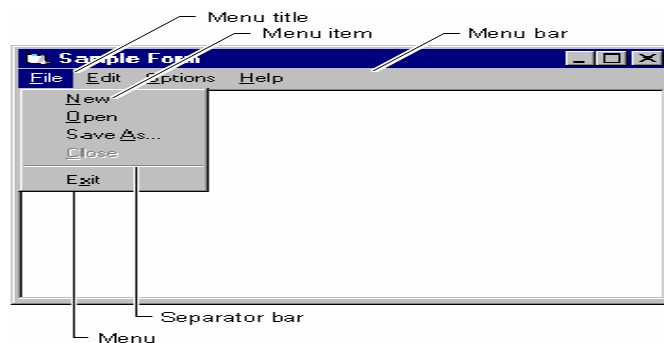


Figure 5.3 The elements of a menu interface on a Form.

The *menu bar* appears immediately below the *title bar* on the Form and contains one or more *menu titles*. When a menu title is clicked (such as File), a menu containing a list of menu items drops down. Menu items can include commands (such as New and Exit), separator bars, and submenu titles. To make application easier to use,

menu items must be grouped according to their function. In the above, for example, the file related commands New, Open, and Save As are all found on the File Menu.

Dynamic Data Exchange

DDE is the basic foundation for interprocess communication among Microsoft Windows applications. In a DDE conversation, the application creates the link is known as the destination application, and the application that responds is known as a source application. (These used to be referred to as client and server respectively). Any application that supports DDE can serve as either a source or a destination. One application can serve as both destination and a source with several other applications at the same time. Only one active DDE link should be established between a Visual Basic control or form and another application. For example, a link between a Microsoft Excel spreadsheet cell and a text control is acceptable as long as the parent form does not already have a link with the same cell. You might cause an infinite loop of updates if there were more than one link between the same two applications.

Object Linking And Embedding

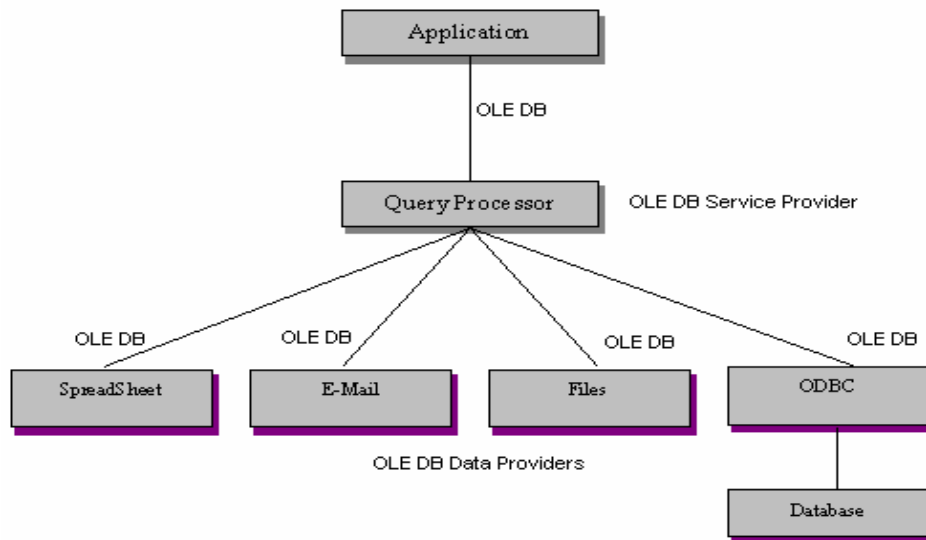


Figure 5.3 Topology of OLE DB applications.

The primary focus of the ODBC is to provide a consistent interface to database data sources. OLE DB is designed with an even broader goal in mind; to provide a methodology to access data regardless of the data source. As shown in Figure 5.3, OLE DB becomes the data access bridge for documents, e-mail systems, file systems, spreadsheets, and database sources using ODBC drivers.

15.3 Introduction to MDI Form

In Visual Basic a child form is an ordinary Form that has its **MDIChild** property set to **True**. An application can include many MDI child forms of similar or different types. At run time, child forms are displayed within the *workspace* of the MDI parent form (the area inside the form's borders and below the title and menu bars). When a child form is minimized, its icon appears within the workspace of the MDI form instead of on the taskbar, as in Figure 5.4.

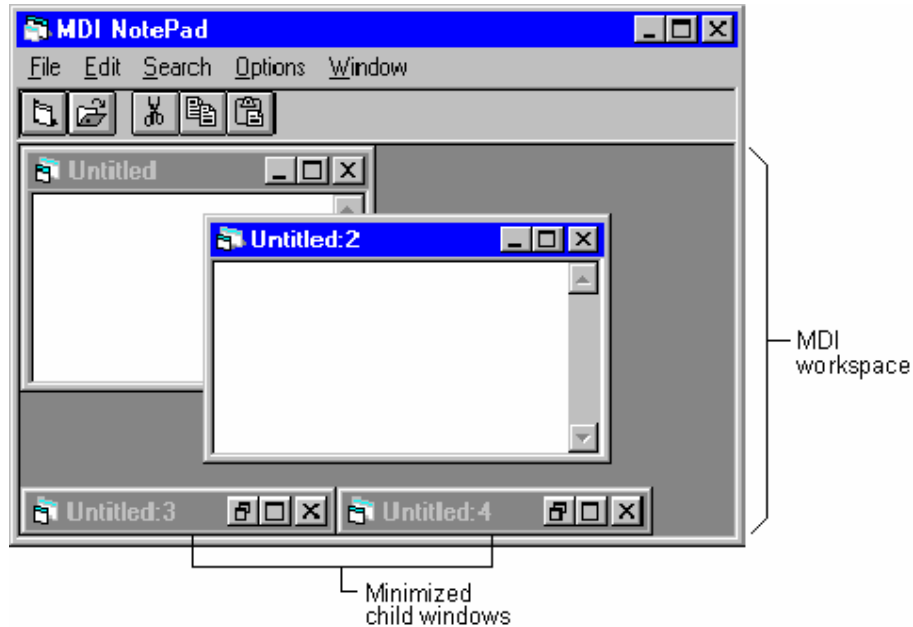


Figure 5.4 Child forms displayed within the workspace of the MDI form.

An MDI application can also include standard, non-MDI forms that are not contained in the MDI form. A typical use of a standard form in an MDI application is to display a modal dialog box.

An MDI form is similar to an ordinary form with one restriction. Intrinsic controls can't be placed on a MDI form unless they have the **Align** property (such as a picture box control) or invisible interface (such as a timer control). More over an application can have only one MDI form. The MDI Form's main property and method are given below:

Property	Description
ActiveForm	It specifies the active MDI child form.

Method	Description
Arrange	Arranges the windows or icons within a MDIForm object

Syntax

MDIForm.**Arrange** arrangement

A value or constant that specifies how to arrange windows or icons on a **MDIForm** object, as described in Table 5.1.

Constant	Value	Description
VbCascade	0	Cascades all nonminimized MDI child forms
VbTileHorizontal	1	Tiles all nonminimized MDI child forms horizontally
VbTileVertical	2	Tiles all nonminimized MDI child forms vertically
VbArrangeIcons	3	Arranges icons for minimized MDI child forms

Table 5.1 Values of *arrangement* argument.

Example

To cascade the child Forms of the MDIForm1 the following statement is used.

```
MDIForm1.Arrange vbCascade
```

15.4 MDI Child Forms Properties

MDI form and all of its child forms take on special characteristics:

- ❖ All child forms are displayed within the MDI form's workspace. The user can move and size child forms like any other form; however, they are restricted to this workspace.
- ❖ When a child form is minimized, its icon appears on the MDI form instead of the taskbar. When the MDI form is minimized, the MDI form and all of its child forms are represented by a single icon. When the MDI form is restored, the MDI

form and all the child forms are displayed in the same state they were in before being minimized.

- ❖ When a child form is maximized, its caption is combined with the caption of the MDI form and is displayed in the MDI form's title bar (see Figure 5.5).
- ❖ The active child form's menus (if any) are displayed on the MDI form's menu bar, not on the child form.

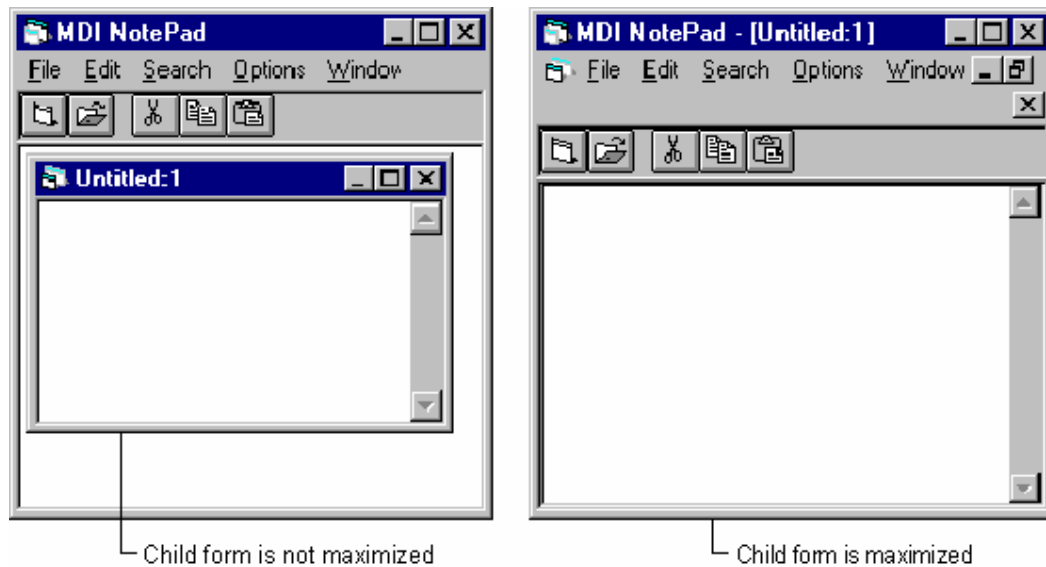


Figure 5.5 A child form caption combined with the caption of an MDI form.

15.6 Short Summary

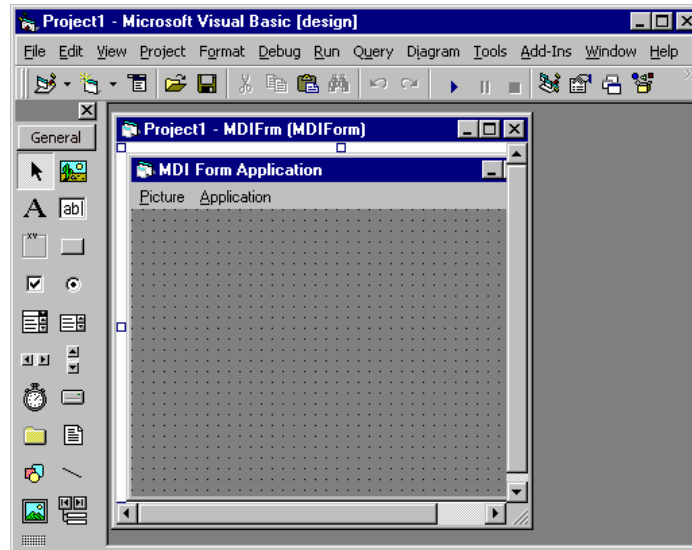
- ☞ MDI stands for Multiple Document Interface, Which is used for opening several windows at the same time.
- ☞ In Word pad application only a single document may be open and in order to open another, the currently opened document must be closed.
- ☞ MicroSoft is a multiple document Interface.

15.7 Brain Storm

1. Differentiate SDI and MDI.
2. What are the properties of MDI form.
3. What are the advantages of MDI form,

Lab Unit - 15 (2 Real Time Hrs)

1. Open a New standard EXE project
2. Design your form as shown below



3. On clicking the Picture & Application button another form should be loaded.

Lecture - 16

Menus

Objectives

In this lecture you will learn the following

- ❖ About Menu
- ❖ About Menu Editor
- ❖ Understanding to create Menu
- ❖ About Menu Control Arrays.

Lecture Unit - 16

- 16.1 Menus - Snap Shot
- 16.2 Menu Editor
- 16.3 Menu Control Arrays
- 16.4 Pop-up menu
- 16.5 Short Summary
- 16.6 Brain Storm

Lab unit 16 (2 Real Time Hrs)

16.1 Snap Shot-Menus

Menus are one of the most common and characteristic elements of the Windows user interface. Even in the old days of character-based displays, menus were used to display methodically organized choices and guide the user through an application. Despite the visually rich interfaces of Window applications and the many alternatives, menus are still the most popular means of organizing a large number of options. Many applications duplicate some or all of their menus in the form of icons on a toolbar, but the menu is a standard fixture of a Form. You can turn the toolbars on and off, but not the menus.

16.2 Menu Editor

Visual Basic provides Menu Editor, which helps to create new menus, add new commands to existing menus, and change and delete existing menus. Menus can be attached only to forms, and you design them with the Menu Editor. To see how the Menu Editor works, start a new Standard EXE project, and when form1 appears in the design window, choose Tools>Menu Editor to open the Menu Editor. Alternatively you can click the Menu Editor button on the toolbar

Generally the following conventions are considered while creating menus.

Feature	Convention
Caption	One or two short specific words such as Save As .
Organization	Menu items should be grouped logically by function and allow for a minimal number of levels to access each feature.
Access Keys	Each menu item should be assigned a access key (the underlined letter in a menu or menu selection) to allow for keyboard access to the menu choices. The key should be unique in each section of the menu and is normally the first letter of the caption.
Shortcut Keys	Any menu features that are frequently used or need to be available from any part of the program should be assigned a shortcut key. For example, the Ctrl+N is the shortcut key for the New item of the File menu in the most of the windows applications. Each shortcut key can be assigned to only one menu item.

Check Box	Menu items that simply set or clear a single program option should contain a checked feature directly in the menu.
Ellipsis	Each menu item that opens a dialog should be followed by an ellipsis (...).

To display the Menu Editor

❖ From the **Tools** menu, choose **Menu Editor**.

- Or -

Click the **Menu Editor**  icon on the toolbar.

This will open the Menu Editor as in Figure 16.1.

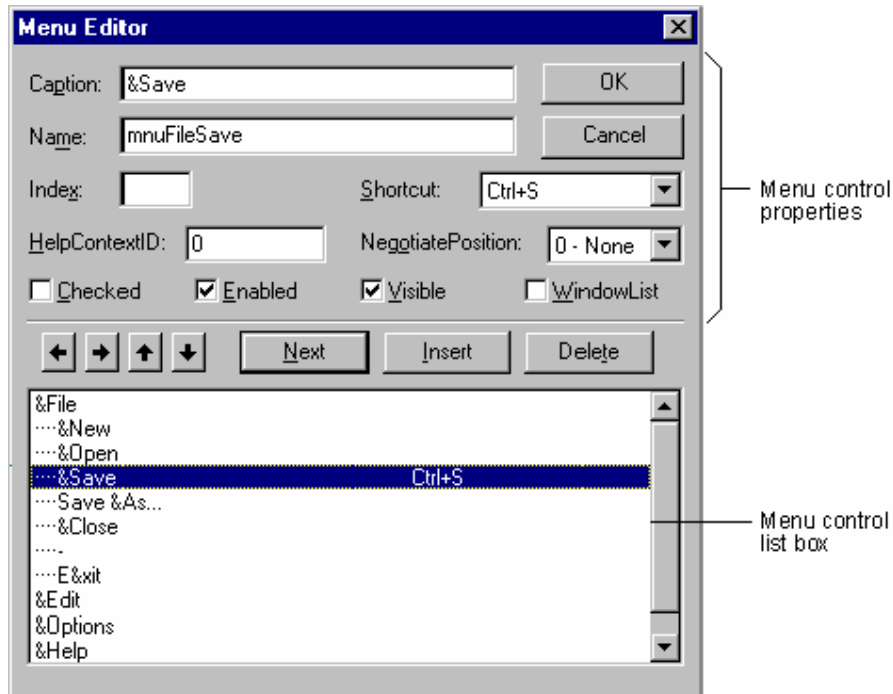


Figure 16.1 The Menu Editor.

While most menu control properties can be set using the Menu Editor, all menu properties are available in the Properties window. The two most important properties for menu controls are:

- **Name** This is the name used to refer the menu control from code.
- **Caption** This is the text that appears on the control.
- **Index** This is to create an array of menu commands
- **Checked** Checked to indicate that they are on or unchecked to indicate that they are off.

- **Enabled** This is to set the initial status of a command by checking or clearing the enable box.
- **Visible** This is to make a command invisible.
- **Window** This is used to maintain a list of all open windows.

To create menu controls in the Menu Editor

- ❖ Select the Form, which requires the menu.
- ❖ From the **Tools** menu, choose **Menu Editor**.
-Or-
Click the **Menu Editor** button on the toolbar.
- ❖ In the **Caption** text box, type the text for the first menu title that has to appear on the menu bar. Also, place an ampersand (&) before the letter to assign the access key for that menu item. This letter will automatically be underlined in the menu. The menu title text is displayed in the menu control list box.
- ❖ In the **Name** text box, type the name that will be used to refer to the menu control in code.
- ❖ Click the left arrow or right arrow buttons to change the indentation level of the control.
- ❖ Choose **Next** to create another menu control.
-Or-
Click **Insert** to add a menu control between existing controls. Up arrow and down arrow buttons can be used to move the control among the existing menu controls.
- ❖ Choose **OK** to close the Menu Editor when all the menu controls for that form have been created. The menu created is embedded in the form, which is something like in the Figure 16.2.

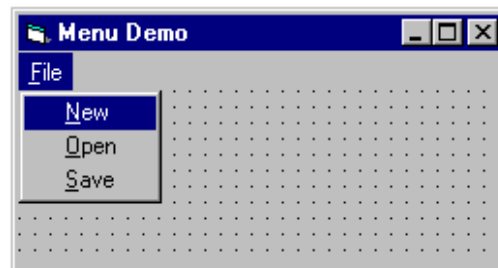


Figure 16.2 Embedded menu in a Form.

- ❖ Dropdown the menu and click on one of its item. The code window for its click event procedure will appear, which is something like in the following:

```
Private Sub itmNew_Click()
```

```
End Sub
```

- ❖ Type the code required to execute when that item is clicked at run time. Follow these steps for all items in the menu.

16.3 Menu Control arrays

In the Menu Editor window add a menu option and set its Index property to 0. You can then add commands with the same name and consecutive Index values. At design time, you don't have to add more than one option. One command with its Index property set to 0 is adequate to create the menu control array. You can use this array's name and an Index value to add new options at runtime.

Figure shows the RTMenu application, which demonstrates how to add to and remove items from a menu at runtime.

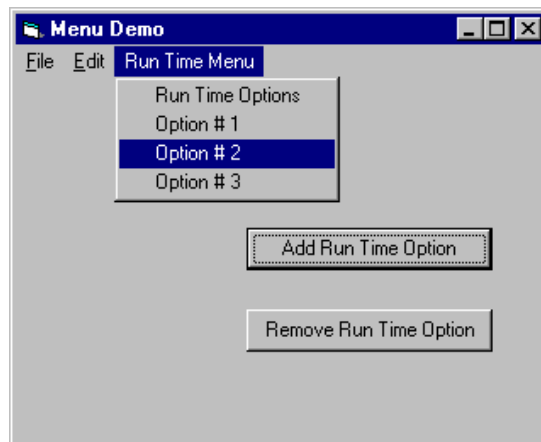


Figure 16.3 The RT Menu Application

Initially, the form's menu contains the following items:

File

Open

Save

Exit

Edit

Copy

Cut

Paste

Run Time Menu

Commands grouped in submenus are indented from the left. The last command's name is RunTimeOptions and its index is 0. Once a command option is specified as a control array, you can easily add commands to the RunTimeOptions array with the Load method. After all, menu commands are very similar to controls.

The two buttons at the bottom of the Form add commands to and remove commands from the Run Time Menu. Each new command is appended at the end of the menu, and the commands are removed from the bottom of the menu(the most recently added commands).

16.4 POP Up Menu

A *pop-up menu* is a floating menu that is displayed over a form, independent of the menu bar. The items displayed on the pop-up menu depend on where the pointer is located when the right mouse button is pressed; therefore, pop-up menus are also called *context menus*. Pop-up menus provide an efficient method for accessing common, contextual commands. For example, if the right mouse button is clicked in a text box, a contextual menu would appear, as in Figure 16.4.

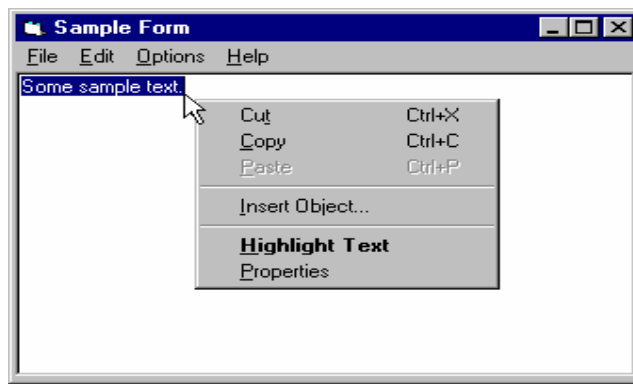


Figure 16.4 Popup menu.

Any menu that has at least one menu item can be displayed at run time as a pop-up menu. Popupmenu method is used to display pop-up menus, and its syntax is as follows:

PopupMenu *menuname* [, *flags* [, *x* [, *y* [, *boldcommand*]]]]

Only the first argument is required, and it is the menu’s name, as specified in the Menu Editor window. The other arguments are optional. The *x* and *y* arguments are the coordinates of a point on the Form (or control) where the menu will be displayed. The *flags* argument defines the location behavior of a Pop-up menu and can have one of the values shown in Table 16.1.

Location constants	Description
VbPopupMenuLeftAlign	Default. The specified <i>x</i> location defines the left edge of the pop-up menu.
VbPopupMenuCenterAlign	The pop-up menu is centered around the specified <i>x</i> location.
VbPopupMenuRightAlign	The specified <i>x</i> location defines the right edge of the pop-up menu.

Table 16.1 *flags* argument’s options.

The following code displays a popup menu named `mnuFile` with its top border centered on the form when the user clicks the form with the right mouse button.

```
Private Sub Form_MouseUp (Button As Integer, Shift As_
    Integer, X As Single, Y As Single)
    If Button = 2 Then ' Check if right mouse button was
        ' clicked
        PopupMenu mnuEdit, vbPopupMenuCenterAlign, X, Y
    End If
End Sub
```

Note : To create a menu that will not display on the menu bar, its top-level menu item's Visible property is to be unchecked at design time. When Visual Basic displays it as a pop-up menu, its Visible property is ignored.

The Bold command Argument

The *boldcommand* argument is used to specify the name of a menu control in the displayed pop-up menu, to appear in bold to emphasize the most frequently used menu item. For example, assume that there is a menu named as `mnuMenu` and one of its items is `Explore` named as `itmExplore`. The following `PopupMenu` statement.

```
PopupMenu mnuMenu , , , itmExplore
```

Will display the popup menu as in Figure 16.5.



Figure 16.5 A Popup menu with a bold item

16.5 Short summary

- ☞ Visual Basic applications can be enhanced by adding menus. A menu interface consists of menu title, menu item, separator bar, menu bar and title bar.
- ☞ A menu control array is a set of menu items that share the same name and event procedure.
- ☞ Separator bar is displayed as a horizontal line between items on a menu bar.

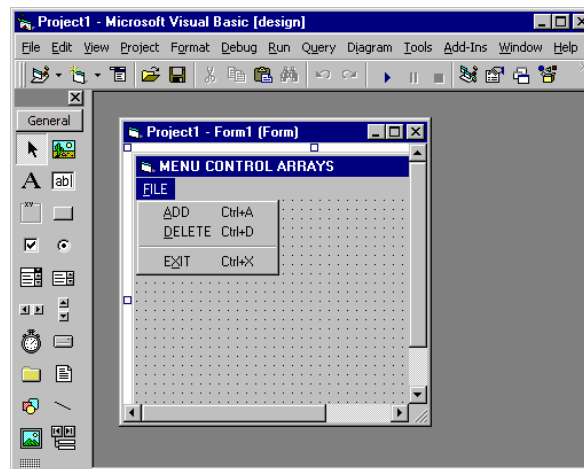
16.7 Brain Storm

1. Define Menu
2. What is a Menu Control Array? What is the usage of it?

Lab Unit - 16 (2 Real Time Hrs)

1

1. Open a new standard EXE project
2. Design your form as shown below



3. On clicking the ADD button Item should be added on the file menu
4. On clicking the Item button MsgBox should be displayed

2

1. Open a New standard EXE project
2. Create the pop-up menu with the following menu items
Bold, Italic, Underline, Blue, Green, Red, Black, Exit
3. Place a text box in the form
4. On selecting an item in the pop-up menu the corresponding effect should be displayed.

Lecture - 17

DDE & ActiveX Controls

Objectives

In this lecture you will learn the following

- ❖ About Dynamic Data Exchange and Object Linking
- ❖ About Object Embedding
- ❖ About Object Linking
- ❖ How to create an ActiveX control?

Lecture Unit - 17

- 17.1 Snap Shot
- 17.2 Dynamic Data Exchange and Object Linking
- 17.3 Object Embedding
- 17.4 Object Linking
- 17.5 Inserting a OLE Object
- 17.6 Creating an ActiveX control
- 17.7 Short Summary
- 17.8 Brain Storm

Lab unit 17 (2 Real Time Hrs)

17.1 Snap Shot

AN Important features of Microsoft Windows Operating System is its ability for applications to share information. OLE is a means of communication, which gives applications the power to directly use and manipulate other windows applications. OLE is an important Windows topic available in Visual Basic. Any Object that supports OLE can be linked. OLE specification permits the user to link and embed objects, and also edit the object with in the container application.

17.2 Dynamic Data Exchange and Object Linking and Embedding

Programmers are not historians. They don't like to read lengthy histories about a specific technology; they just want to learn how to use it. But when talking about a specific technology As all-encompassing as Active X, knowing a little about its previous history is a must. Before you can understand where Active X is going , you first need to know where it came from.

ActiveX can trace its roots all the way back to Dynamic Data Exchange (DDE), a way of passing data and commands between applications. Though DDE is still supported by Visual Basic to maintain backward compatibility, it is seldom used.

Out of DDE evolved OLE1.0, Object Linking Embedding. True to its name, OLE brought the ability to link Objects to one another or to embed one object inside another. The classic example of using OLE is the word processor/spreadsheet combination. A reference to a spreadsheet file can be added to a word processing document. If the user was to click on the reference , the spreadsheet program that is used to edit the spreadsheet would be automatically loaded and changes could be made. In this way , the spreadsheet is linked to the word processing document. With OLE, the spreadsheet can also be embedded into the word processing document. A window that shows the contents of the spreadsheet would be contained within the document. Changes to the spreadsheet can then be made inside that window. The spreadsheet program is still used to make the changes, but this is transparent to the user. He or She never has to leave the word processor to make changes to the spreadsheet.

DDE and OLE1.0 are technologies that were designed to change the way we use computers. They bridged the gab between application-centric use and data-centric use. In application-centric Computing, to work on a certain kind of file(such as word processing document , a graphic image, or a spreadsheet), you would call up

whichever program was used to edit that kind of file. To work on a word processing document, you need only click on the document itself. The operating system then determines which application is associated with that file type and loads it, along with the document .

Although the concept of data-centric computing sounds like it does nothing more than make things easier for the user, it goes far beyond that. It also facilitates the embedding of different object types within other objects. A web browser , for example , can display an HTML document that contains many different object types: images , sounds, videos, animation, and others. If the web browser does not natively support the object type , the external program (typically, a “plug-in” for Netscape browsers or an ActiveX control in Internet Explorer) that is associated with the object type can be used to display the object within the page. Again , this is transparent to the user.

After that OLE2.0 and COM and DCOM came to the world of computer era.

17.3 Object Embedding

With this technique, you can insert an object from one application (the server application) into Another application (the container application). The inserted object is a copy of the original and can be manipulated and stored separately and apart from the original object. For example, you can embed a range of cells from an Excel worksheet in a Word document. To edit the cells , you switch to Excel by double-clicking the embedded Excel object. If the container application supports in-place editing, you’ll see the menus of the server application right in the container application.

17.4 Object Linking

This technique is similar to embedding, except that the embedded data are also linked to the document form which they come. Changes to the object in the server application are reflected automatically in the container application. Linking doesn’t store the object, it makes a reference to the object exposed by the server application. Each time you open the document that contains the linked object exposed, the container application contacts the server application, which actually opens the most up-to-date version of the linked object. Linked objects are not copies. They are the originals, viewed from within different containers.

17.5 Inserting a OLE Object

An OLE object is an item that is exposed, or made available, by an OLE server application. OLE server applications expose different types (or Classes) of objects.

OLE objects are used in container applications. As Figure-17.1 illustrates, Excel (the OLE server) can expose a worksheet (the OLE object) that can be inserted as is in a Word document (the container application).

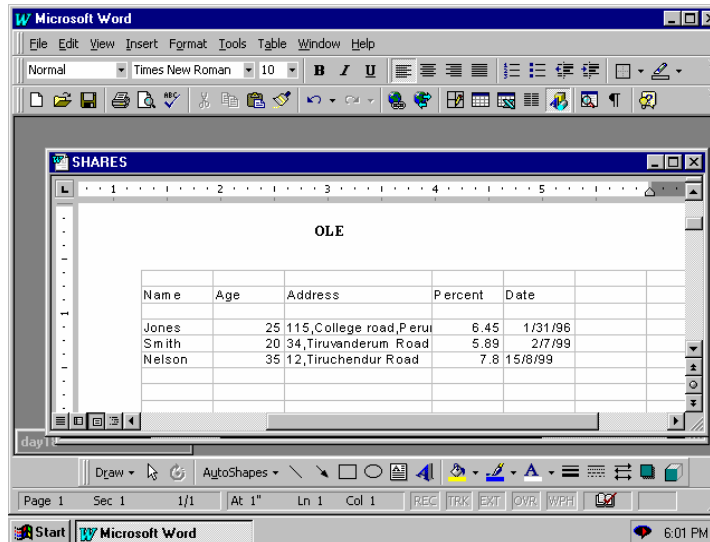
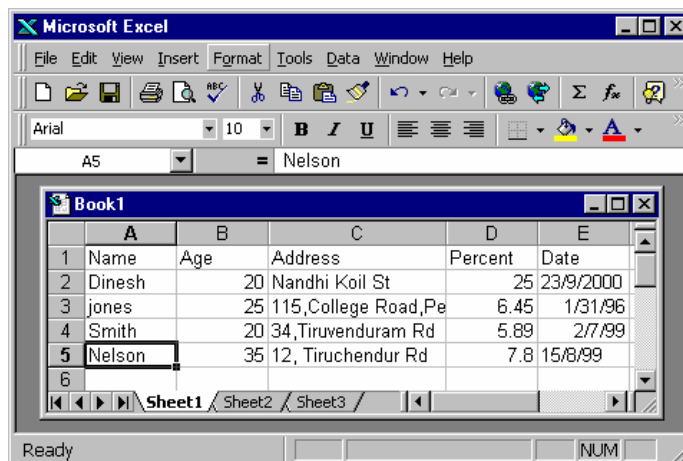


Figure 17.1 In word the Excel sheet is inserted

OLE Automation

The method allows you to programmatically manipulate objects exposed by another application from within your Visual Basic applications. It's also a standard that defines how code is shared between applications and how applications can be controlled from within other applications. For example, when you copy a range of cells from an Excel spreadsheet onto a Word document, you embed the range of cells. With OLE Automation, your application can request that Excel perform some calculations and return the result to Word. For example, you can pass a table to Excel and request that Excel manipulate the numeric data in ways that are not possible with Word's tools and then return the processed data to Word.



17.6 Creating An Activex Control

The simplest way to create the user interface of an ActiveX control is to use controls that already exist. This is known as creating from constituent controls. To illustrate the techniques for creating a control, we are going to create a simple Security control that allows the user to enter a userID and password. If you handle logins for your applications, or have to pass user information to a database, you at some time have had to ask for a userID and password. Typically, you place a couple of labels and text boxes on a form and do a little bit of text processing. This is a simple enough task, but wouldn't it be nice if all you had to do was draw a single control on the form, then access its properties to get all the needed information? This is what the Security control does

1. Choose File ➤ New Project to open the New Project window
2. Click the ActiveX Control icon. Visual Basic creates a new project that contains a UserControl, named UserControl1 as shown in Fig-17.2 . This is the control's Form on which the visible interface will be built. (Once the project is created, you will need to set the Name property of the control otherwise your control will go through life with the generic name User Control1 (or 2 and so on).
3. Click the control from the toolbox and place it on the UserControl1.
4. Set the basic properties of the control (name or the caption property for a label or command button

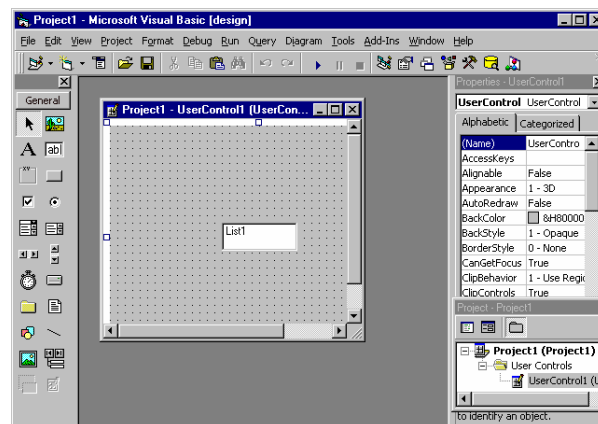


Figure 17.2 UserControl1

Using The Activex Control

After you have finished developing and testing your control, you can use the control in many programs. You can easily use the control in any other VB program that you write. You can also use the control in any application that supports ActiveX controls.

These applications include Microsoft Office, other development tools such as FoxPro, and even Internet applications or web pages that support ActiveX. With this type of usage available, you can see why the capability to create ActiveX control is such a powerful feature in VB.

Compiling the control

The first step to being able to use your control in other application is to complete it. When you compile a control, you are creating an OCX file that can be accessed by other programs. Compiling your control involves the following steps:

1. Select the project in your project group that contains the control.
2. Select the Make OCX item from Visual Basic's File menu.
3. From the Make Project dialog box, click the Options button to set any necessary compilation options.
4. In the Make Project dialog box set the name and path for the OCX file.
5. Click the OK button to start the compilation.
6. When the process is finished, you will have an OCX file that others can use to access your control.

Working with the Control in Visual Basic

To use your ActiveX control in other Visual Basic programs, you have two options:

- You can use the OCX that contains the control.
- You can add the CTL control definition file to the project.

Using the OCX for the control is the preferred method of accessing the control unless you are going to modify the control. Even then, you are better off modifying the original control project and recompiling the control than making modifications in a new project. To add your OCX control to your Visual Basic toolbox, open the Components dialog box by pressing Ctrl+T. Then click the Browse button in the dialog to bring up the Open dialog that allows you to specify the name and path of the control. When you select the control, it will appear in the Components dialog and will be marked to be added to your toolbox. After closing the Components dialog, your control will appear in the Toolbox and will be accessible to the current project, like any other control.

Using the Control in a Web page

Internet Explorer 3.0 and 4.0, as well as some other Internet programs, are capable of displaying and using Active X controls. The key to using the control is the use of the

<object> tag in the HTML code for the web page. The <object> tag specifies the following information about the control:

- The controls's ID
- A name for the instance of the control
- The ClassID of the control

This information identifies the control to your web browser and allows the control to be displayed if the browser supports ActiveX. The Following code shows one instance of the HTML code needed to embed an ActiveX control in a page:

```
<object id = "Test1" name = "Test1"  
classid = "clsid:21F43727-A8FD-11D1-BCCD-0000C051F6F9"  
border = "0"  
width= "188" height= "54"></object>
```

Web page designers make it easy to add ActiveX controls to your pages. These designers handle all the object information for you so you don't have to remember all the details about ClassIDs and the other information.

Testing the Active X control

Once you have created the Active X control, you need to test it to make sure that all the procedures work as you think they should. You can use all the standard debugging techniques that are available in VB to debug your control. There are only a few different steps involved in setting up a test for your ActiveX control.

1. Add a standard project to your development environment to create a project group, which is the mechanism used for testing ActiveX controls and servers
2. Close the User Control window of your control project. If the User - Control (form) window is open, Visual Basic assumes that you are still working on the design of the control and will not allow you to create an instance of the control in another form
3. Create an instance of the control on your test form. Then you can test the properties of the control and write code to test its methods

17.7 short summary

☞ Object linking and embedding is a technology that enables a Windows programmer to create an application that can display data from many different

applications and enables the user to edit that data from within the application in which it was created.

- ☞ The OLE control allows adding objects from other applications and it can have only one object at a time.
- ☞ OLE automation is an industry standard technology that applications use to expose their OLE objects to development tools, macro languages, and other applications that support OLE Automation.

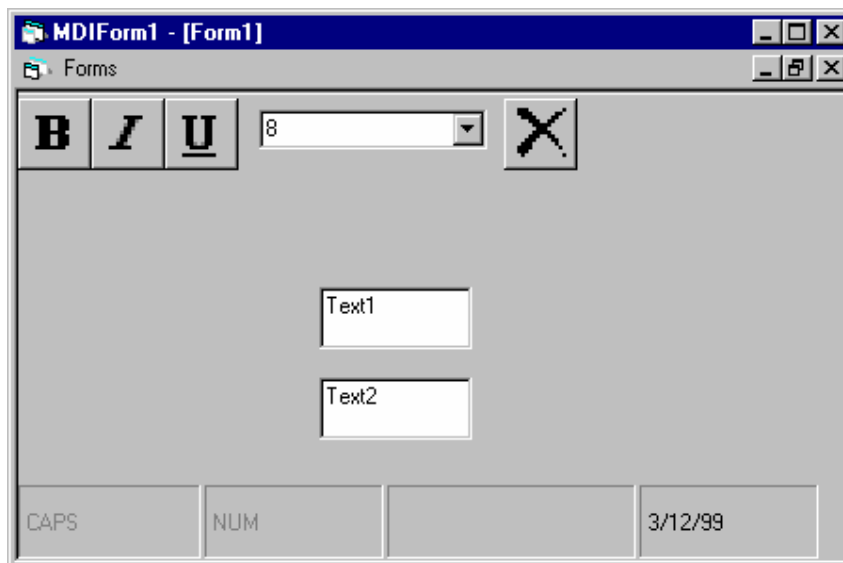
17.8 Brain Storm

1. What is OLE automation?
2. Is it possible to create an OLE control using Visual Basic ? State reasons.
3. What is the difference between DDE and OLE.

Lab Unit - 17 (2 Real Time Hrs)

1. MDI Form

Open a new project. Add a MDI form along with two forms. Set the child Property of forms and provide a way (MENU) to access the two forms from the MDI form.



2. OLE Drag Drop Property

Define two controls that support OLEdragdrop. Set OLEdrag/drop mode to automatic, drag and drop data from one textbox to another.

3. Drag and drop Property

Define two containers (picture box, frame) and three controls (command button, Checkbox, picture box). Set the drag drop mode to automatic. Drag and drop the controls to different containers. Provide drag icons. Set the Dragover Effect when a control is dragged over a container.

4. Image List:

Place a Image List on the form and insert some pictures to the Image List Control. Give appropriate key value from the pictures.

5. Status Bar

Place a status bar control in the MDI FORM. Add panels to the status bar. The status bar should display the time, date in separate panels. Also display the message in any of the panels whenever a text is dragged from one text box and Dropped into another text box. When a new form is opened display its name in any one of the panels of the status bar.

6. Tool Bar:

- Place a toolbar control on the form2. Add buttons to the toolbar as shown below. Set the Image index for each and every button.
- When the first button is pressed the text in the “Text1” must be displayed in Bold format
- When the second button is pressed the text in the “Text1” must be displayed in “Italic” format
- When the third button is pressed the text in the “Text1” must be Underlined
- When a number is selected from the combo1 the font size of the text box must changed according to the value selected
- When the last button is selected, the form “Form1” must be closed.

Lecture - 18

ActiveX Controls

Objectives

In this lecture you will learn the following

- ❖ About ActiveX Controls
- ❖ Designing ActiveX Controls
- ❖ Compiling ActiveX Controls
- ❖ ActiveX document Migration wizards

Lecture Unit - 18

- 18.1 Snap Shot - ActiveX Documents
- 18.2 Designing and Compiling the ActiveX Controls
- 18.3 ActiveX Document Migration Wizards
- 18.4 Short Summary
- 18.5 Brain Storm

Lab Unit 1 (2 Real Time Hrs)

18.1 Snap Shot - ActiveX Documents

ActiveX documents are designed the same way as the Visual Basic Forms are designed. That is, they can have intrinsic controls and ActiveX controls as the normal Visual Basic Forms have. They can also show message boxes and secondary Forms. But, in contrast to the independent appearance of the Visual Basic Forms, ActiveX documents appear in containers, i.e Internet Browser windows such as Internet Explorer, Netscape Navigator, etc. With Internet Explorer 3.0 or later, ActiveX document can be saved and read by writing to and reading from the document's data file.

The base object of an ActiveX document is the **UserDocument** object that resembles a Standard Visual Basic **Form** object with some exceptions.

The **UserDocument** object has most, but not all, of the events that are found on a **Form** object. The events present on a **Form** that are not found on the **UserDocument** include:

- ❖ Activate
- ❖ Deactivate
- ❖ Load
- ❖ QueryUnload
- ❖ Unload

Events present on the **UserDocument**, but not found on a **Form** object include:

- ❖ EnterFocus
- ❖ ExitFocus
- ❖ Hide
- ❖ InitProperties
- ❖ ReadProperties
- ❖ Scroll
- ❖ Show
- ❖ WriteProperties

Note : ActiveX documents can be packaged in either in-process or out-of-process components.

Hyperlink Objects

Hyperlink object behaves as a property of UserDocuments. Using the properties and methods of the **Hyperlink** object, the ActiveX document can request a hyperlink-

aware container, such as Microsoft Internet Explorer, to jump to an another ActiveX Document.

This object has three methods:

- ❖ NavigateTo
- ❖ GoBack
- ❖ GoForward

The Navigate to method

This method makes a hyperlink jump to the specified target. For example, the following code presumes an ActiveX document named "axdMyDoc" exists:

```
UserDocument.Hyperlink.NavigateTo "c:\mydocs\axdmydoc.vbd"
```

The above code will cause the appearance of the axdmydoc in the web browser.

Note : Actually ActiveX documents are stored with the extension .dob. But time being, assume that they are stored with the extension .vbd. Explanation about this is given shortly.

The GoBack and GoForward methods

When the ActiveX documents are viewed in a Web browser, the browser maintains the list of viewed documents i.e history of documents. The **GoBack** and **GoForward** methods enable to go backwards or forwards through the list. For example, an ActiveX document ActXDoc1 is viewed first and an another ActiveX Document ActXDoc2 is currently being viewed. Issuing GoBack method here will enable to revisit to the ActXDoc1. Once GoBack method is called, the document from which the GoBack method is called will be added in the Forward list. Issuing GoFoward method will enable to revisit to that document - in this case ActXDoc2. However, if there is no documents in the Backward and Forward list, then issuing either GoBack or GoForward will cause errors. So error-trapping can be enabled as given in the following example:

Example:

```
Private Sub cmdGoForward_Click()  
    On Error GoTo noDocInHistory  
  
    UserDocument.Hyperlink.GoForward  
    --- or ---  
    UserDocument.Hyperlink.GoBack
```

Exit Sub

noDocInHistory:

Resume Next

End Sub

18.2 Designing And Compiling ActiveX Document

This section creates an ActiveX Document project called ActXDoc by demonstrating how to:

- ❖ Add User Documents
- ❖ Add Forms
- ❖ Use hyperlink object
- ❖ Add menus to the User Document

To create ActXDoc project

- ❖ Choose File | New Project to open the New Project dialog box.
- ❖ Double-click the **ActiveX Document Dll** icon. This will create an ActiveX Document with default User Document named UserDocument1.



ActiveX Document Dll Icon

Note : The ActiveX Document Dll project type will package the ActiveX documents as *in-process* component. To package it as *out-of-process* component use ActiveX Document EXE project type.

- ❖ Choose Project | Add User Document to add a second User Document.
- ❖ Repeat the above step to add third User Document.
- ❖ Name them ActXDoc1, ActXDoc2 and ActXDoc3 respectively.
- ❖ Choose Project | Add Form to add a Form and name it as frmAddress.
- ❖ Create objects and set their properties in the User Document ActXDoc1 as given below:

Object	Property	Value
Label	Caption	"ActiveX Document 1 (ActXDoc1)"
Command Button	Name	cmdNavigateTo
	Caption	"Navigate To"
Text Box	Name	txtAddress
	Text	""
Command Button	Name	cmdShowForm
	Caption	"Show Form"

The designed ActXDoc1 should resemble the Figure 18.1.

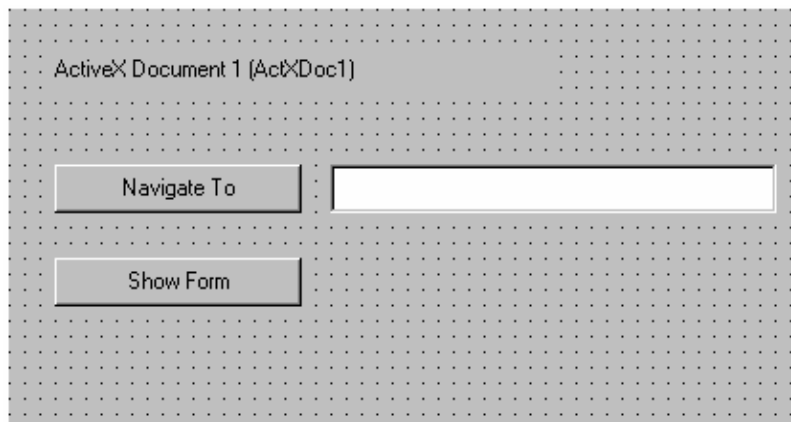


Figure 18.1 Designed ActXDoc1 User Document.

The text box in the ActXDoc1 will accept a name of a User Document with full path such as ..\ActXDoc\ActXDoc2.vbd. Clicking **Navigate To** button will cause a jump to the document specified in the text box. Clicking the **Show Form** will display the frmAddress Form, which will also accept address to Jump To. When the Form is closed, the address in the txtAddress text box will be replaced by the address entered in the text box of the Form frmAddress.

Of course the Form frmAddress is not necessary. For demonstration purpose it is included. The Show Form button contains the following line to active the Form frmAddress.

```
frmAddress.Show vbModal
```

It is necessary to activate the Form in modal mode. Because some web browsers such as Internet Explorer, don't support the showing of modeless forms called from within a DLL. However, if the documents are compiled as *out-of-process* component, Internet Explorer is capable of showing both modal and modeless forms.

Note: A project started as ActiveX DLL can be converted to ActiveX EXE and can be packaged as *out-of-process* component by simply changing the Project type from ActiveX DLL to ActiveX EXE in the Project Properties window.

- ❖ Create objects and set their properties in the Form frmAddress as given below:

Object	Property	Value
Form	Caption	"Address Form"
	Border Style	Fixed Single
	Control Box	False
Label	Caption	"Comfortable Text box to enter the address to Navigate To"
Textbox	Name	txtcomAddress
Command Button	Name	cmdClose
	Caption	Close

The designed Form should resemble the Figure 18.2.

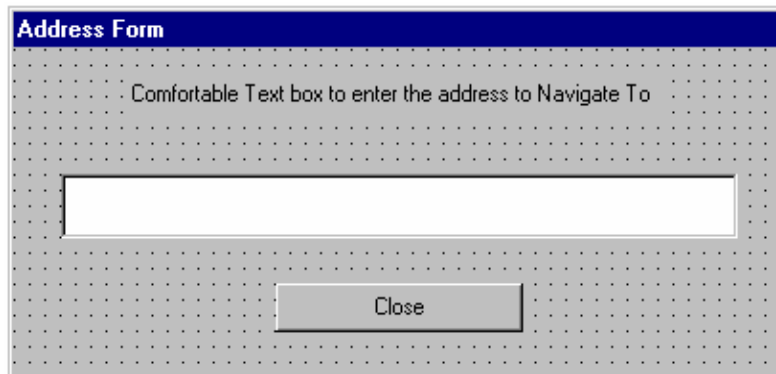


Figure 18.2 Designed frmAddress Form.

The reason to change the Control Box property of the frmAddress Form is that the Form can be closed only by clicking the Close button. The Close button's click event procedure has the following line:

```
frmAddress.Hide
```

That is, the Close button simply hides the Form from the vision. It doesn't unload it from the memory so that the ActXDoc1 can replace the text of the txtAddress text box with the newly entered text in the text box of the Form using the following line.

```
txtAddress.Text = frmAddress.txtcomAddress.Text
```

If the Form is closed through the control box, then it will be removed from the memory and the ActXDoc1 can't access the text entered in the txtcomAddress text box. So only, the Control Box property is set to False.

- ❖ Create objects and set their properties in the User Document ActXDoc2 as given below:

Object	Property	Value
Label	Caption	"ActiveX Document 2 (ActXDoc2)"
Command Button	Name	cmdGoBack
	Caption	"Go Back"
Command Button	Name	cmdGoForward
	Caption	"Go Forward"
Command Button	Name	cmdGoNext
	Caption	"Go Next"

The designed ActXDoc2 should resemble the Figure 18.3.

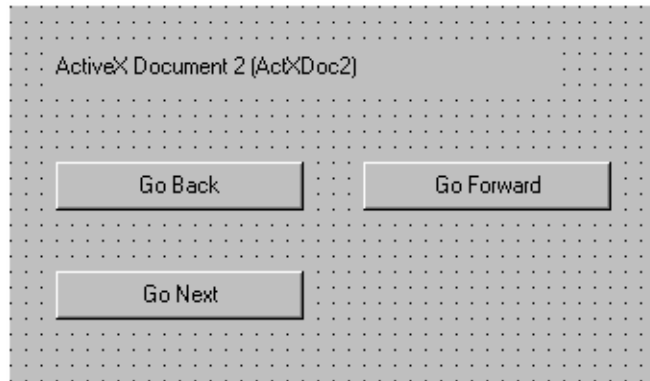


Figure 18.3 Designed ActXDoc2 User Document.

Clicking **Go Back** button will enable to revisit to the previously visited Document, say ActXDoc1. Clicking the **Go Next** button will display the next document i.e., ActXDoc3. If the ActXDoc2 is visited using **Go Back** from another document, then alone clicking **Go Forward** will display that document. Otherwise it won't work.

- ❖ Create objects and set their properties in the User Document ActXDoc3 as given in the following:

Object	Property	Value
Label	Caption	"ActiveX Document 3 (ActXDoc3)"
Label	Caption	"Your Name Please"
Textbox	Name	txtName
	Text	""

Command Button	Name	cmdGoBack
	Caption	"Go Back"

The designed ActXDoc3 should resemble the Figure 18.4.

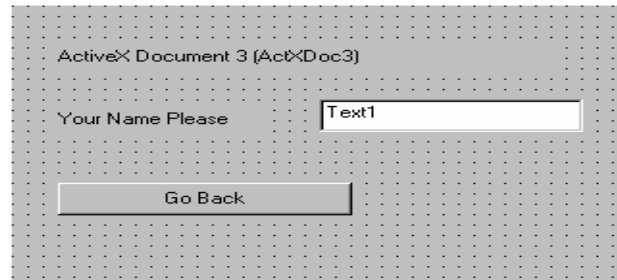


Figure 18.4 Designed ActXDoc3 User Document.

If user enters his name in the **Name** text box when this document appears in the Web browser, visits to some other document and again revisits to this document, the value entered by him won't appear. There should be some mechanism to store the values entered by him. For that, this document declares a private variable `m_Name` and defines Property Get/Let procedures as in the following:

```
' Declaring local copy to store the value entered in the
' Name text box
```

```
Private m_Name As String
```

```
' Returning the value of the local copy
Public Property Get Name() As String
    Name = m_Name
End Property
```

```
' Setting new value for the local copy
Public Property Let Name(ByVal NewValue As String)
    m_Name = NewValue
End Property
```

The Name text box change event procedure has the PropertyChanged statement as in below:

```
Private Sub txtName_Change()
    PropertyChanged
End Sub
```

This causes the Web Browser to prompt the user to save changes. If the user responds positively, the code writes the property value to the PropertyBag using WriteProperties event, whose code is given below.

```
Private Sub UserDocument_WriteProperties(PropBag As PropertyBag)
    PropBag.WriteProperty "Name", txtName.Text, ""
End Sub
```

The WriteProperties behaves in the same manner as it behaved in ActiveX Controls. The only difference is that it writes values in the .frm files when it is used for ActiveX controls, and here it will write the values in the .vbd file. When this document is revisited the ReadProperties event will occur to read from property bag and to display them. Here is the code of ReadProperties event for this User Document.

```
Private Sub UserDocument_ReadProperties(PropBag As PropertyBag)
    txtName.Text = PropBag.ReadProperty("Name", "")
End Sub
```

- ❖ Add the following code to the **frmAddress** form.

```
Private Sub cmdClose_Click()
    Me.Hide
End Sub
```

- ❖ Add the following code to the **ActXDoc1** User Document.

```
Option Explicit
Private Sub cmdNavigate_Click()

    On Error GoTo InvalidAddress ' Enabling Error Handler
    ' Navigating to the sepcified document
    Hyperlink.NavigateTo txtAddress.Text
Exit Sub
```

InvalidAddress:

```
    MsgBox "Invalid Address - Try again with a valid_
        address"
    Resume Next
End Sub
```

```
Private Sub cmdShowFrm_Click()

    ' Assigning the text of txtAddress to the text
```

```
' box of the Form frmAddress
frmAddress.txtcomAddress.Text = txtAddress.Text
```

```
' Displaying the Form in Modal type
frmAddress.Show vbModal
' Replacing the text of txtAddress with the newly
  ' entered Address in the text box of the Form
txtAddress.Text = frmAddress.txtcomAddress.Text
```

```
End Sub
```

- ❖ Add the following code to **the ActXDoc2** User Document.

```
Option Explicit
Private Sub cmdGoBack_Click()
  On Error GoTo noDocInHistory
```

```
' Going Backward
Hyperlink.GoBack
Exit Sub
```

```
noDocInHistory:
```

```
MsgBox "No document in History - Can't go back"
Resume Next
```

```
End Sub
```

```
Private Sub cmdGoForward_Click()
  On Error GoTo noDocInHistory
```

```
' Going Forward
Hyperlink.GoForward
Exit Sub
```

```
noDocInHistory:
```

```
MsgBox "No document in History - Can't go Forward"
Resume Next
```

```
End Sub
```

```
Private Sub cmdGoNext_Click()
  'Jumping to the ActXDoc3
  Hyperlink.NavigateTo_
    "C:\Raguram\VBASIC\ActXDoc\ActXDoc3.vbd"
```


End Sub

- ❖ Add the following code to the ActXDoc3 User Document

Option Explicit

' Declaring local for the Name property

Private m_Name As String

Private Sub cmdGoBack_Click()

 On Error GoTo noDocInHistory

 ' Going Backward

 Hyperlink.GoBack

 Exit Sub

noDocInHistory:

 MsgBox "No document in History - Can't go Back"

 Resume Next

End Sub

Private Sub txtName_Change()

 ' Indicating that property value is changed

 PropertyChanged

End Sub

Private Sub UserDocument_ReadProperties(PropBag As_
 PropertyBag)

 ' Reading the Name property from the PropertyBag

 ' And assigning it to the txtName text box

 txtName.Text = PropBag.ReadProperty("Name", "")

End Sub

Private Sub UserDocument_WriteProperties(PropBag As_
 PropertyBag)

 ' Writing the value of the txtName text box for the

```
' Name property through proeprty bag
PropBag.WriteProperty "Name", txtName.Text, ""
End Sub
```

```
' Property Get / Let Procedures for Name Property
```

```
Public Property Get Name() As String
    Name = m_Name
End Property
```

```
Public Property Let Name(ByVal NewValue As String)
    m_Name = NewValue
End Property
```

- ❖ Choose File | Save Project to save the project files. Name them as given below. Visual Basic will provide the indicated extensions automatically. (Save them in a separate folder, say ActXDoc).

File	File Name	Extension
ActXDoc1	ActXDoc1	.dob
ActXDoc2	ActXDoc2	.dob
ActXDoc3	ActXDoc3	.dob
frmAddress	frmAddress	.frm
Project	ActXDoc	.prj

- ❖ Choose File | Make ActXDoc.Dll to build the ActiveX Document as in-process component.

Compiling your Documents

After you have finished testing and debugging your document, you are ready to compile the document for distribution. To start the compilation process, select the Make Register.exe menu option from the File menu of VB. This will then open the Make project dialog box, prompting you to supply a name and location for the EXE or DLL file. The name of the .vbd file is based on the Name property of the UserDocument object. This file is placed in the same folder that you specified for the EXE file. After you have compiled the ActiveX Document, you can view it in any container program that handles ActiveX documents. Internet Explorer and Office Binder are two such programs. At the time this book is being written, Netscape Navigator 4.0 does not support ActiveX documents. It is already mentioned that ActiveX Documents can be built as either in-process or out-of-process component. In

both cases, when the project is compiled, in addition to creating an .exe. or a .dll file, Visual Basic creates a Visual Basic Document file (Figure 18.5), which has the extension .vbd and it is placed in the same directory as the compiled component (.exe or .dll).

The .vbd file is actually an OLE structured storage – this basically means that data in the file can be accessed and manipulated via standard OLE interfaces. The Internet Explorer uses this interface. So only instead of specifying .dob files .vbd files are mentioned while calling the **NavigateTo** method of hyperlink object to open another document.

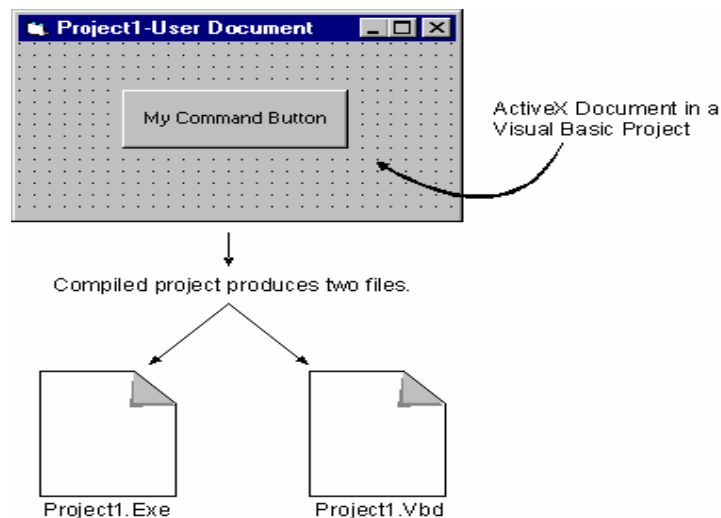


Figure 18.5 Compiling User Documents produces .exe (or .dll) and .vbd files.

18.3 ActiveX Document Migration Wizard

The ActiveX Document Migration Wizard must be available to the Visual Basic IDE. If you have not already done so, you can make the Wizard available by selecting it from the Add-In Manager dialog box, which is accessible by choosing the Add-In Manager Item from the Add-In menu. After you have added the Wizard to your desktop, you can run it by choosing the ActiveX Document Migration Wizard item from the add-Ins menu.

To begin the conversion process, the project whose forms you want to convert must be open. After you have opened the project, launch the ActiveX Document Migration Wizard. At this point you will be presented with a series of screens that will guide you through the conversion process. The first of these screens is introductory only, so click the Next button to continue.

The second screen of the Wizard, shown in Fig 18.6 includes a list box from which you can select the forms that you would like to convert. Any forms that have been

defined for this project will be displayed in this list box. To select forms, simply click in the check box next to the name of the form. When you have made all your selections, click the Next button to continue.

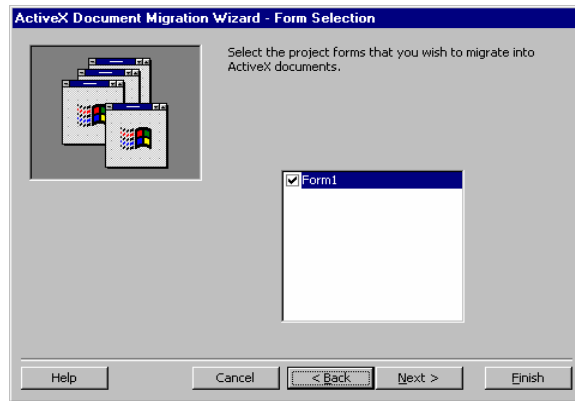
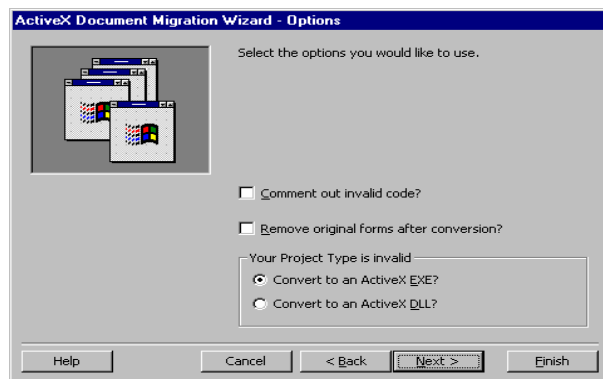


Figure 18.6 - You can select multiple forms in the list box

The Options page of the ActiveX Document Migration Wizard, shown in Figure 18.7, lets you control how the Wizard will process the forms you have selected. The three options enable you to do the



following:

Figure 18.7- Select the options that are appropriate to your needs

- Choose to comment out invalid code. This option will comment out statements such as Load,Unload,or End that are not supported by ActiveX documents.
- Remove original forms after conversion. This option will remove the forms from the current project after the conversion is made. Typically, you will not want to check this option because you will want your original project intact.
- Choose whether to convert your project to an ActiveX EXE or ActiveX DLL project. The option defaults to ActiveX EXE. (ActiveX DLL are used for creating shared components rather than applications)

After making your choices, click the Next button to continue. The last page of the Wizard allows you to view a summary report that describes the Wizard's actions. After making your selection, click the Finish button to begin the conversion. The summary report is a valuable development tool because it provides you with information regarding the steps you need to complete. Figure 18.8 shows the summary report.

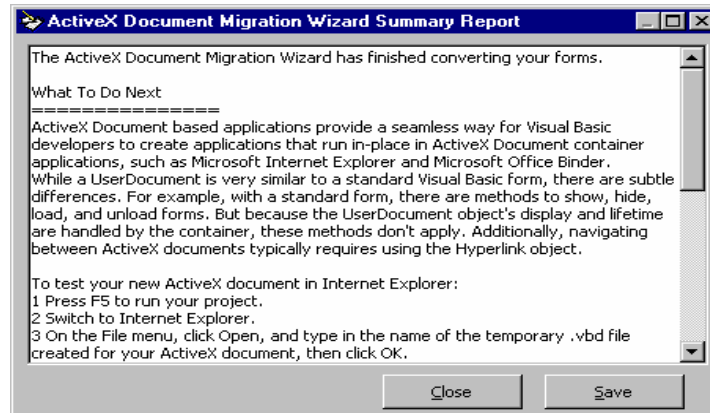


Figure 18.8 – The ActiveX Document Migration Wizard uses a summary report

Viewing the Wizard's work

When the Active X Document Migration Wizard has finished its work, it places the newly created UserDocument objects in the same project as the original forms. The document source files are stored in the same folder as the original form files and are given similar names, with the appropriate extension. For example, a form stored in the file frmTest1.frm creates a UserDocument stored in the file docTest1.dob. As previously stated, the controls of the form are copied to the UserDocument and their relative positions are preserved.

Also, as stated previously, most of the code from your original form is copied over to the UserDocument. Invalid code is commented out and identified by the ActiveX Document Migration Wizard. (This assumes that you choose to comment out invalid code on the Option page of the Wizard.)

18.4 Short Summary

- ☞ Visual Basic can be used to compile class based projects as ActiveX components.
- ☞ The servers can be implemented as ActiveX DLL or ActiveX EXE components. The difference between the two is how the server is executed.
- ☞ ActiveX components are Objects Servers, designed to be used with other application.
- ☞ There are three major steps involved in developing an ActiveXExe.

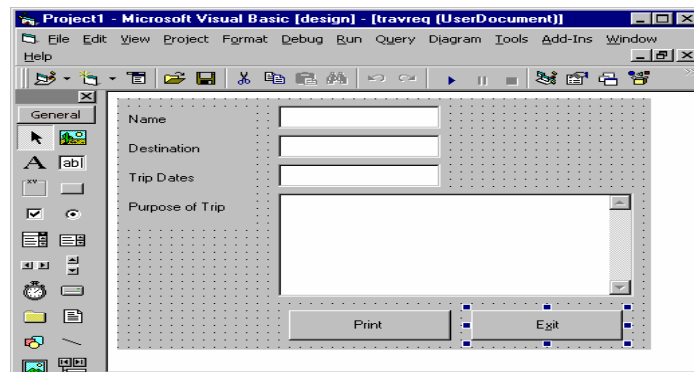
- To create the ActiveX EXE component.
- To compile this component and register.
- To test the component with our client test application.

18.5 Brain Storm

1. What are the uses of ActiveX components. Explain the areas in which it is using.

Lab Units - 18 (2 Real Time Hrs)

1. Open a new project in Visual Basic and select the ActiveX documentEXE
2. Click the UserDocument1 in the Project window and name it as travreq



3. Design the form as shown below
4. On Clicking the Print button this form must be printed.

Lecture - 19

Data Project

Objectives

In this lecture you will learn the following

- ❖ About Data project environment
- ❖ About Creating Data Report
- ❖ About registry

Lecture Unit - 19

- 19.1 Data Project Environment - snap shot
- 19.2 Creating a Data Project
- 19.3 Dragging and Dropping
- 19.4 Registry
- 19.5 API Registry
- 19.6 Short Summary
- 19.7 Brain Storm

Lab Unit 18 (2 Real Time Hours)

19.1 Data Project Environment - snap shot

In Microsoft Visual Basic, database projects are not used. You can use the Data Project template to create a new project, which adds default controls and objects, including the data environment. To use the Visual Database Tools in Visual Basic, you can either add a data connection to the data environment, or make a data connection directly from the Data View window. Also, in Visual Basic, all data connections are made through OLE DB, although you can still use ODBC data source names (DSNs) to define your connections.

In Microsoft Visual Basic you can either add a data connection to the data environment, or make a data connection directly from the Data View window. In Visual Basic, all data connections are made through OLE DB, although you can still use ODBC data source names (DSNs) to define your connections.

19.2 Creating a Data Project

To see the new visual database tools in action, start a new project, and in the Project type dialog box, select Data Project. A Data project is not a special project type. It's basically a Standard EXE project, but Visual Basic loads all the database tools into the project's Toolbox.

In the Project Explorer window, Visual Basic display a new form, as usual, and two ActiveX Designers. A Designer is a special add -in that simplifies the design of components. Database applications rely on two basic components:

- One or more Data Environment component(s)
- One or more DataReport component(s)

The DataEnvironment components lets you design a connection to a database and retrieve the desired records. The DataReport component lets you design reports and use them from within your applications. Both components are based on visual database tools, and they don't require programming. Of course, you can't go far without programming, but the programming model

The Data project's Toolbox also contains a number of ActiveX controls:

- The ADODC (ADO Data control), which is equivalent to the Data control
- The DataList and DataCombo controls, which are data-bound versions of the ListBox and ComboBox controls; they work with the ADO Data control (but not with the Data control)

- The MSFlexGrid control, which is a hierarchical Grid control that also works with the ADO Data control. The MSFlexGrid is very similar to the MSFlex Grid Control, but it's meant to be populated automatically by the DataEnvironment Designer.

To access a database with ADO (or any other data tool), you need two types of objects:

- One or more Connection objects
- One or more Command objects

The Connection object connects your application to a database, and the Command object retrieves records from the database. The command object can also update the database, but I won't discuss this topic in much detail here. As you will see, the Command Object accepts a SQL statement and executes it against the database. If it's a query statement, the Command object returns a RecordSet with the matching records. If it's an action query, the Command object executes it against the database, where one or more records will be updated(optionally, the Command object can return the records that were affected by the query).

Another just as common and just as important operation in database application design is the generation of reports. Retrieving the required data from the database is not difficult, but printing the report over multiple pages with running totals is quite a task. Most developers use third-party tools to generate reports, especially since printing them on different printers is a real challenge. The DataReport Designer allows you to create the outline of a report with point-and-click operations. You first specify the data to be included in the report with the DataEnvironment object, which works in tandem with the DataReport ActiveX Designer to allow you to implement advanced database operations (queries and reports) without any programming.

19.3 Dragging and Dropping

A combination of features that allows the user to drag an item and drop it into another item using the mouse. An item can be a source (the item the user drags) or a target (the item on which the user drops a source).

A feature that enables you to drag one object onto another to perform an action. To drag an object, click the object and, while holding down the mouse button, move it to the new location. When you are satisfied with the location, release the mouse button to drop the object.

When you design Visual Basic applications, you often drag controls around on the form. The drag-and-drop features in Visual Basic allow you to extend this ability to the user at run time. The action of holding a mouse button down and moving a control is called dragging, and the action of releasing the button is called dropping.

Note Dragging a control at run time doesn't automatically change its location – you must program the relocation yourself, as described in "Changing the Position of a Control." Often, dragging is used only to indicate that some action should be performed; the control retains its original position after the user releases the mouse button.

Using the following drag-and-drop properties, events, and method, you can specify both the meaning of a drag operation and how dragging can be initiated (if at all) for a given control.

CategoryItemDescriptionPropertiesDragModeEnables automatic or manual dragging of a control.DragIconSpecifies what icon is displayed when the control is dragged.EventsDragDropRecognizes when a control is dropped onto the object.DragOverRecognizes when a control is dragged over the object.MethodsDragStarts or stops manual dragging.

All controls except menus, timers, lines, and shapes support the DragMode and DragIcon properties and the Drag method. Forms recognize the DragDrop and DragOver events, but they don't support the DragMode and DragIcon properties or the Drag method.

Note : Controls can only be dragged when they do not have the focus. To prevent a control from getting the focus, set its TabStop property to False.

19.4 Registry

A central hierarchical database used to store information necessary to configure the system for applications and hardware devices. The registry contains information – such as the applications installed on the computer and the types of documents each can create, property sheet settings for folders and application icons, what hardware exists on the system, and which ports are being used – which the operating system continually references during operation.

A database maintained by Windows that stores configuration information about the operating system, all Windows applications, ActiveX, OLE, and optional components such as ODBC. For example, the registry is where Windows stores the associations between file name extensions and applications, and where Visual FoxPro stores its application-specific configuration information.

Registry Properties

Registry Object Properties Collections

PropertyDescription

Application

Application

Write: NoThe Application object.AutostartLicensing

Boolean

Write: YesDetermines if license logging is started automatically when SQL Server starts. If True, licensing is started automatically when SQL Server starts. If False, licensing must be started manually.AutostartMail

Boolean

Write: YesDetermines if Mail is started when SQL Server starts. If True, Mail is started automatically when SQL Server starts. If False, Mail must be started manually.AutostartServer

Boolean

Write: YesDetermines if SQL Server is started when Windows NT starts. If True, SQL Server is started automatically when Windows NT starts. If False, SQL Server must be started manually.CaseSensitive

Boolean

Write: NoIndicates if SQL Server is case-sensitive. If True, SQL Server is using a case-sensitive sort order. If False, SQL Server is using a case-insensitive sort order.CharacterSet

String

Write: NoDescription of the character set and code page used by SQL Server.ErrorLogPath

String

Write: YesThe full path and file name of the SQL Server error log.MailAccountName

String

Write: YesThe Mail account name.MailPassword

String

Write: YesThe password for the Mail account specified by the MailAccountName property.MasterDBPath

String

Write: YesThe full path and file name of the database device that contains the MASTER database.NTEventLogging

Boolean

Write: NoIndicates if Windows NT event logging is enabled. If True, SQL Server sends all events to the Windows NT event log as well as the SQL Server error log. If False, SQL Server sends events only to the SQL Server error log.NumberOfProcessors

Long

Write: NoThe number of processors on the SQL Server computer.Parent
SQLServer
Write: NoThe SQLServer object.PerfMonMode
SQLOLE_PERFMON_TYPE

Write: YesThe mode of the performance monitor.PhysicalMemory
Long

Write: NoThe total physical memory in megabytes on the SQL Server
computer.RegisteredOrganization
String

Write: NoThe name of the organization provided when SQL Server was installed.
RegisteredOwner
String

Write: NoThe name of the owner provided when SQL Server was installed.SortOrder
String

Write: NoDescription of the sort order used by SQL Server.SQLRootPath
String

Write: YesThe full path where SQL Server was installed.TapeLoadWaitTime
Long

Write: YesThe number of minutes SQL Server will wait when trying to read a tape
that has not been loaded in the drive. A value of -1 means no wait timeout; SQL
Server will not sop waiting. A value of 0 means SQL Server will only try once to
access a tape in the drive.. TypeOf
SQLOLE_OBJECT_TYPE

Write: NoThe type of object.UserData
Long

Write: YesTemporary storage space contained in this object for private use by the
application. SQL-DMO does not use this property.

19.5 API Registry

Before diving into using API procedures this section discusses the fundamental elements of API procedures.

API procedures are contained in DLLs. So applications can link and use them at run time rather than to link statically at compile time. This means that the libraries can be updated independently of the application, and many applications can share a single DLL. Microsoft Windows itself is comprised of DLLs, and other applications call the procedures within these libraries to display windows and graphics, manage memory, or perform other tasks.

The four functional categories of the Windows API procedures (as these procedures are used in Windows 32 bit environment, they can be abbreviated as Win32 API) are as follows:

- ❖ Windows Management (User)
- ❖ Graphics Device Interface (GDI)
- ❖ System Services
- ❖ Multimedia

Windows Management provides essential procedures to build and manage applications. All the basic input and output of the system goes through this layer of the Win32 API, including mouse and keyboard input and all processing of message sent to the application.

Graphics Device Interface provides procedures to manage graphical devices such as monitor and printer etc. This interface also provides the procedures to define fonts, pens, and brushes, and for line and circle operations.

System Services provides procedures to access the resources of the computer such as disk, and the operating system.

Multimedia provides procedures to play waveform audio, MIDI music, and digital video.

Accessing the Microsoft Windows API

You can gain access to the Windows API (or other outside DLLs) by declaring the external procedures within your Visual Basic application. After you declare a procedure, you can use it like any other language feature in the product.

The most commonly used set of external procedures are those that make up Microsoft Windows itself. The Windows API contains thousands of functions, subs, types, and constants that you can declare and use in your projects. These procedures are written in the C language, however, so they must be declared before you can use them with Visual Basic. The declarations for DLL procedures can become fairly complex. While you can translate these yourself, the easiest way to access the Windows API is by using the predefined declares included with Visual Basic.

The file Win32api.txt, located in the \Winapi subdirectory of the main Visual Basic directory, contains declarations for many of the Windows API procedures commonly used in Visual Basic. To use a function, type, or other feature from this file, simply copy it to your Visual Basic module. You can view and copy procedures from

Win32api.txt by using the API Viewer application, or by loading the file in any text editor.

Note The Windows API contains a vast amount of code. To find reference information on the procedures and other details included in this API set, refer to the Win32 SDK, included on the Microsoft Developer Network Library CD.

Using the API Viewer Application

The API Viewer application enables you to browse through the declares, constants, and types included in any text file or Microsoft Jet database. After you find the procedure you want, you can copy the code to the Clipboard and paste it into your Visual Basic application. You can add as many procedures as you want to your application.

The API Viewer application

To view an API file

1. From the Add-Ins menu, open the Add-In Manager and load API Viewer.
2. Click API Viewer from the Add-Ins menu.
3. Open the text or database file you want to view.
 - To load a text file into the viewer, click File \ Load Text File and choose the file you want to view.
 - To load a database file, click File \ Load Database File.
4. Select the type of item you want to view from the API Types list.

Note You can have the API Viewer automatically display the last file you viewed in it, when it is opened, by selecting View \ Load Last File.

To add procedures to your Visual Basic code

1. Click the procedure you want to copy in the Available Items list.
2. Click Add. The item appears in the Selected Items list.
3. Indicate the scope of the item by clicking Public or Private in the Declare Scope group.
4. To remove an entry from the Selected Items list box, click the item and click Remove.
5. To remove all entries from the Selected Items list box, click Clear.

To copy the selected items to the clipboard

1. Click Copy. All of the items in the Selected Items list will be copied.
2. Open your Visual Basic project and go to the module in which you want to place the API information.
3. Position the insertion point where you want to paste the declarations, constants, and/or types, and then choose Edit \ Paste.

Converting Text Files to Jet Database Files

To optimize speed, you can convert the Win32api.txt file into a Jet database file, because it is much faster to display the list when opening a database than when opening a text file.

To convert a text file to a jet database file

1. Start the API Viewer application.
2. Click File \ Load Text File and open the .txt file you want to convert.
3. Click File \ Convert Text to Database.
4. Choose the file name and location for your database file, then click OK.

Loading an API File Automatically from the Command Line

You can specify a text or database file on the command line for Apilod32.exe so that the file is automatically loaded when you start API Viewer. Use the following syntax to load the file you choose when you start the API Viewer application:

```
Apilod32.exe {/T | /D} filename
```

ArgumentDescription/TAPI Viewer will load the file as a text file. /T must be uppercase./DAPI Viewer will load the file as a database file. /D must be uppercase.FilenameThe path of the file you want to open.

There must be a space between /T or /D and the filename argument. An error message will be displayed if the file is not found. If you specify a file that is not a database or text file, an error message will be displayed when you try to load the file.

Tip You can view a prompt that shows the parameters of the command line syntax by using a DOS window to navigate to the directory in which the API viewer application is installed, then typing apiload /?.

Viewing the Win32api.txt file with a Text Editor

You can also load the Win32api.txt file in a text editor, such as Microsoft Word or WordPad, to locate the procedures you want to use. Again, you just copy the procedures from the file to a Visual Basic module to use them in your application.







Tip Don't load the Win32api.txt file into a module. This is a large file, and it will consume a lot of memory in your application. You will generally use only a handful of declarations in your code, so selectively copying the declarations you need is much more efficient.

Using Procedures from Other Sources

If you are attempting to call a procedure in a DLL that is not part of the operating system, you must determine the proper declaration for it. The topic "Declaring a DLL Procedure" explains the syntax of the Declare statement in detail.

Note If you use Visual C++ (or a similar tool) to create DLLs that will be called by Visual Basic, use the `__stdcall` calling convention. Do not use the default calling convention (`_cdecl`).

19.5 Short Summary

-  In Visual Basic, all data connections are made through OLE DB, although you can still use ODBC data source names (DSNs) to define your connections
-  Dragging a control at run time doesn't automatically change its location --- you must program the relocation yourself, as described in "Changing the position of a control". Often,
-  Dragging is used only to indicate that some action should be performed; the control retains its original position after the user releases the mouse button.
-  **Note** You can have the API Viewer automatically display the last file you viewed in it, when it is opened, by selecting View \ Load Last File.
-  Controls can only be dragged when they do not have the focus. To prevent a control from getting the focus, set its TabStop property to False.
-  By specifying a text or database file on the command line for Apilod32.exe so that the file is automatically loaded when you start API Viewer.

19.6 Brain Storm

1. What is a data report? What are the advantages of Data Report?
2. What is a Registry?
3. What are the advantages of API Registry?

Lab Unit - 19 (2 Real Time Hrs)

Create a Data Project using NWIND (which is in the VB98 folder) database

Lecture - 20

Dynamic Link Library

Objectives

In this lecture you will learn the following

- ❖ About Dynamic link library
- ❖ About Calling Convention
- ❖ About Microsoft Transaction Server
- ❖ Registering VB DLL with MTS

Lecture Unit - 20

- 20.1 Snap Shot - DLL
- 20.2 DLL calling convention
- 20.3 Understanding DLL Calling Convention
- 20.4 MicroSoft Transaction Server
- 20.5 Introduction to MTS
- 20.6 Three Tired Computing
- 20.7 Registering VB DLL with Transaction Server.
- 20.8 Short Summary
- 20.9 Brain Storm

Lab unit 20 (2 Real Time Hrs)

20.1 Snap Shot - DLL

Dynamic-link libraries (DLL) are modules that contain functions and data. A DLL is loaded at runtime by its calling modules (.EXE or DLL). When a DLL is loaded, it is mapped into the address space of the calling process. (DLL) A file with the file name extension .dll that contains one or more functions compiled, linked, and stored separately from the computing processes that use them.

DLLs can define two kinds of functions: exported and internal. The exported functions can be called by other modules. Internal functions can only be called from within the DLL where they are defined. Although DLLs can export data, its data is usually only used by its functions.

DLLs provide a way to modularize applications so that functionality can be updated and reused more easily. They also help reduce memory overhead when several applications use the same functionality at the same time, because although each application gets its own copy of the data, they can share the code.

The Microsoft® Win32® application programming interface (API) is implemented as a set of dynamic-link libraries, so any process that uses the Win32 API uses dynamic linking.

A Windows term for a set of routines that can be called from procedures and are loaded and linked into your application at run time. DLLs include utility routines or specific functions not intrinsic to Microsoft Windows. `_DLLDefined` when `/MD` or `/MDd` (Multithread DLL) is specified. `/MD`, `/ML`, `/MT`, `/LD` (Use Run-Time Library)

With these options, you can select either single-threaded or multithreaded run-time routines, indicate that a multithreaded module is a dynamic-link library (DLL), and select the retail or debug version of the library.

Note : Having more than one copy of the run-time libraries in a process can cause problems, because static data in one copy is not shared with the other copy. To ensure that your process contains only one copy, avoid mixing static and dynamic versions of the run-time libraries. The linker will prevent you from linking with both static and dynamic versions within one .EXE file, but you can still end up with two (or more) copies of the run-time libraries. For example, a dynamic-link library linked with the static (non-DLL) versions of the run-time libraries can cause problems when used with an .EXE file that was linked with the dynamic (DLL) version of the run-time libraries. (You should also avoid mixing the debug and non-debug versions of the libraries in one process.)

Command LineProject SettingsDescription/MDMultithreaded DLLDefines `_MT` and `_DLL` so that both multithread- and DLL-specific versions of the run-time routines

are selected from the standard .H files. This option also causes the compiler to place the library name MSVCRT.LIB into the .OBJ file.

Applications compiled with this option are statically linked to MSVCRT.LIB. This library provides a layer of code that allows the linker to resolve external references. The actual working code is contained in MSVCRT.DLL, which must be available at run time to applications linked

with MSVCRT.LIB. /MDdDebug Multithreaded DLL Defines _DEBUG, _MT, and _DLL so that debug multithread- and DLL-specific versions of the run-time routines are selected from the standard .H files. It also causes the compiler to place the library name MSVCRTD.LIB into the .OBJ file. /MLSingle-Threaded Causes the compiler to place the library name LIBC.LIB into the .OBJ file so that the linker will use LIBC.LIB to resolve external symbols. This is the compiler's default action. LIBC.LIB does not provide multithread support. /MLdDebug Single-Threaded Defines _DEBUG and causes the compiler to place the library name LIBCD.LIB into the .OBJ file so that the linker will use LIBCD.LIB to resolve external symbols. LIBCD.LIB does not provide multithread support. /MTMultithreaded Defines _MT so that multithread-specific versions of the run-time routines are selected from the standard header (.H) files. This option also causes the compiler to place the library name LIBCMT.LIB into the .OBJ file so that the linker will use LIBCMT.LIB to resolve external symbols. Either /MT or /MD (or their debug equivalents /MTd or /MDd) is required to create multithreaded programs. /MTdDebug Multithreaded Defines _DEBUG and _MT. Defining _MT causes multithread-specific versions of the run-time routines to be selected from the standard .H files. This option also causes the compiler to place the library name LIBCMTD.LIB into the .OBJ file so that the linker will use LIBCMTD.LIB to resolve external symbols. Either /MTd or /MDd (or their non-debug equivalents /MT or MD) is required to create multithreaded programs. /LDNot applicableCreates a DLL.Passes the /DLL option to the linker. The linker looks for, but does not require, a DllMain function. If you do not write a DllMain function, the linker inserts a DllMain function that returns TRUE.

Links the DLL startup code.

Creates an import library (.LIB), if an export (.EXP) file is not specified on the command line; you link the import library to applications that call your DLL. Interprets /Fe as naming a DLL rather than an .EXE file; the default program name becomes basename.DLL instead of basename.EXE.

Changes default run-time library support to /MT if you have not explicitly specified one of the /M options/LDdNot applicableCreates a debug DLL. Defines _DEBUG.

The debug options select the debug versions of the library or DLL and define `_DEBUG`. For more information on using the debug versions, see C Run-Time Debug Libraries.

The Advantages of Using DLLs

Dynamic linking has the following advantages:

- Saves memory and reduces swapping. Many processes can use a single DLL simultaneously, sharing a single copy of the DLL in memory. In contrast, Windows must load a copy of the library code into memory for each application that is built with a static link library.
- Saves disk space. Many applications can share a single copy of the DLL on disk. In contrast, each application built with a static link library has the library code linked into its executable image as a separate copy.
- Upgrades to the DLL are easier. When the functions in a DLL change, the applications that use them do not need to be recompiled or relinked as long as the functions' arguments and return values do not change. In contrast, statically linked object code requires that the application be relinked when the functions change.
- Provides after-market support. For example, a display driver DLL can be modified to support a display that was not available when the application was shipped.
- Supports multi-language programs. Programs written in different programming languages can call the same DLL function as long as the programs follow the function's calling convention. The programs and the DLL function must be compatible in the following ways: the order in which the function expects its arguments to be pushed onto the stack, whether the function or the application is responsible for cleaning up the stack, and whether any arguments are passed in registers.
- Provides a mechanism to extend the MFC library classes. You can derive classes from the existing MFC classes and place them in an MFC extension DLL for use by MFC applications.
- Eases the creation of international versions. By placing resources in a DLL, it is much easier to create international versions of an application. You can place the

strings for each language version of your application in a separate resource DLL, and have the different language versions load the appropriate resources.

A potential disadvantage to using DLLs is that the application is not self-contained; it depends on the existence of a separate DLL module.

20.2 DLL Calling Convention

After the function is declared, it can be called just as a standard Visual Basic function. Here, the statement to call SetWindowText is attached the Form Load event:

```
Private Sub Form_Load()  
    SetWindowText Form1.hWnd, "Extending Visual Basic"  
End Sub
```

When this code is run, the function first uses the hWnd property to identify the window whose caption is to be changed (Form1.hWnd), then changes the text of that caption to "Extending Visual Basic "

Note : Visual Basic can't verify the arguments that are passed to a DLL procedure. If incorrect values are passed, the procedure may fail, which may cause Visual Basic application to stop. So, care should be taken while experimenting with DLL procedures.

20.3 Understanding DLL Calling Convention

The only difference between Visual Basic procedures and the Win32 API procedures is that API procedures must be declared before they can be used. Because the Win32 API procedures (DLL procedures) reside in files that are external to Visual Basic application they can't be used without specifying their locations (DLL file name) and their required arguments. The Win32 API procedures are contained mainly in three Windows DLLs. They are:

Library	Description
❖ USER32.DLL	Library for user interface routines.
❖ KERNAL32.DLL	Library for system services.
❖ GDI32.DLL	Library for graphics device interfaces.

Once the Win32 API procedures are declared they can be used just like a native Visual Basic procedure.

Declaring a DLL Procedure

Declare statement is used to declare a Win32 API function or procedure.

Syntax

If a procedure returns a value

```
Declare Function publicname Lib "libname" [Alias "alias"] [[[ByVal] variable [As type]
[ByVal] variable [As type]]...] As Type
```

If a procedure does not return a value

```
Declare Sub publicname Lib "libname" [Alias "alias"] [[[ByVal] variable [As type]
[ByVal] variable [As type]]...] ]
```

Note : DLL procedures declared in standard modules are public by default and can be called from anywhere in application. DLL procedures declared in any other type of module are private to that module and when they are declared they must be preceded with Private keyword.

Example

The following declare statement shows how to declare SetWindowText function in a form module.

```
Private Declare Function SetWindowText Lib "user32" Alias_ "SetWindowTextA"
(ByVal hwnd As Long, _ByVal lpString As String) As Long
```

The function SetWindowText, changes the caption on a form. Of course the form's caption can simply be changed using its caption property. No need to declare and then to call the function SetWindowText to change the form's caption. But the purpose of using SetWindowText function is to demonstrate how to declare and call an API function.

The Lib clause in the declaration statement specifies the name of the .dll file that contains the function (In the above statement the .dll file name is USER32

that contains the `SetWindowText` function). And the function `SetWindowText` expects two arguments `hwnd` and `lpString`. The `hwnd` argument denotes the handle of a window (here window is a Form) and `lpString` specifies the text that to be set as caption of the window.

The function `SetWindowText` is declared with an alias `SetWindowTextA`. The alias is the function that actually resides in the `WIN32.API`. The function call `SetWindowText` is the function call inside Visual Basic.

You can extend the native capabilities of Visual Basic by taking advantage of the facilities of ActiveX controls (.ocx files), ActiveX objects, and dynamic-link libraries (DLLs). External libraries allow you to access not only the capabilities of other programs, but of Windows itself.

For example, you can use an ActiveX control to read and update the Windows registry directly, or it can call system-level functions by linking to one of the DLLs in Windows.

If the functionality you need isn't already available in an external library, you can create your own ActiveX control with the Microsoft Visual Basic® Control Creation Edition version 5.0.

If the functionality you require is available in a DLL, you can link to the library and call its functions. Before calling a DLL function, you must determine its calling protocol, including the name of the function, the number and data types of its parameters, and the data type of its return value.

20.4 Microsoft Transaction Server (MTS)

A component-based transaction processing system for developing, deploying, and managing server applications. MTS defines a programming model for developing distributed, component-based applications. It also provides a run-time infrastructure and a graphical tool for deploying and managing these applications. There has been an evolution of the way multiuser transaction-based applications are developed. First came mainframes : Users with dumb terminals posted transactions directly to the database. Figure show s the history of multiuser transaction applications.

Next came client/server. Client/server allowed users to work with intelligent machines (PCs). Work was broken down into two parts. The client PC handled the GUI, and the server handled the database. When the client needed data it requested

only the data it needed, instead of directly talking to the whole database. This type of application was sometimes referred to as two-tier architecture, because there were two levels to the application: the GUI and the database. Cutting edge applications are written as three-tiered or n-tier applications.

The ability to build n-tier components – ActiveX EXEs and DLLs – is not new to Visual Basic. What's new to Version 6.0 is the ability to tailor your components to run in conjunction with Microsoft Transaction Server (MTS). Until Microsoft Transaction Server, this typically meant writing server applications or using remote procedure calls. There were no Microsoft Foundation Class Wizards to quickly whip up a framework, and to get reasonable performance the server application developer had to worry about thread and connection pooling.

Microsoft Transaction Server is a component-based transaction processing system for developing, deploying, and managing high performance, scalable, and robust server applications. MTS defines a programming model and provides a run-time environment and graphical administration tool for managing enterprise applications.

Using the new `MTSTransactionMode` property of the class module, you can create components that ignore MTS, or support transactions by setting a property. When the component is not running in an MTS environment, the property is ignored.

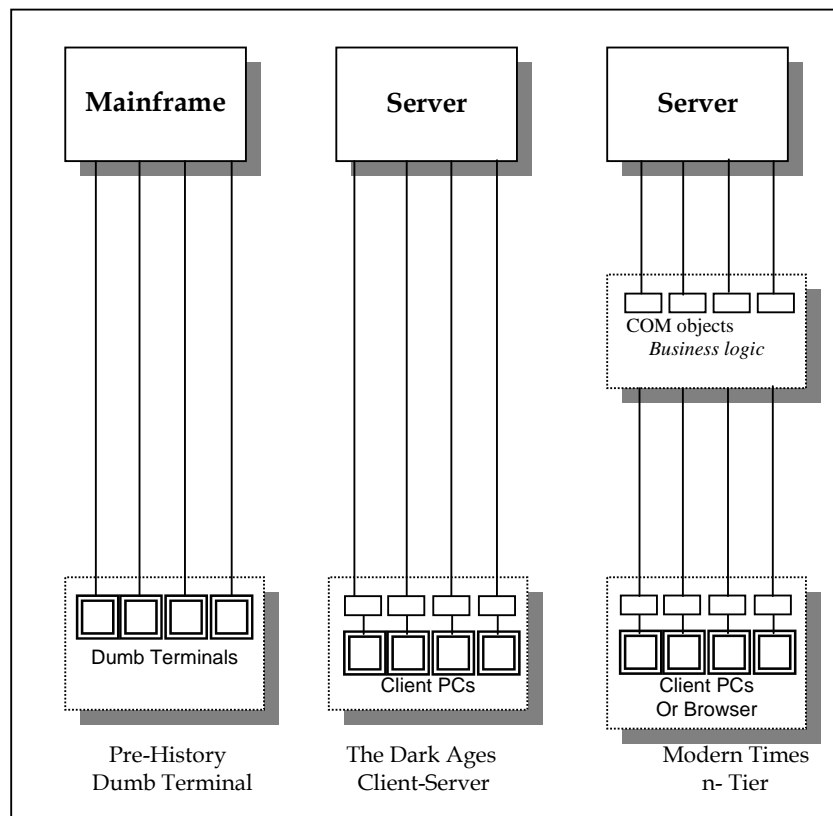


Figure 20.1 History of multiuser transaction applications

20.5 Introduction To Microsoft Transaction Server

Microsoft's Transaction Server is somewhat of a cross between a Transaction Monitor and an ORB, although it tends to lean more towards the ORB set of functionality. If you are running an application on a Windows NT system that has Transaction Server installed and you are using ODBC to access a database, Transaction Server transparently inserts itself between your application and the database to manage that connection (as well as all of the other open connections to the same database). Transaction Server also allows application functionality to be built as a series of ActiveX DLLs and distributed across a network. The following might pique your desire to read the online documentation that comes with MTS.

- Integration with IIS
- Support for Transaction Active Server Pages - Scripts in Active Server Pages can now executed within an MTS-managed transaction.
- Crash Protection for IIS Applications - IIS Web applications can now execute with in their own MTS package, providing process isolation and crash protection for Web application.
- Transactional Events - Developers can embed commands into scripts on Active Server Pages, enabling customization of Web application response based on transaction outcome.
- Microsoft Cluster Server Supports - MTS 2.0 supports Microsoft Cluster Server (MSCS), which enables automatic failover of MTS packages in a cluster
- Support for the XA transaction protocol, including native support for Oracle

20.6 Three - Tiered Computing

A logical application model in which the business application's features are expressed in terms of three categories of services: user services, business services, and data services. This is a conceptual architecture describing the functionality the application is designed to deliver, rather than its implementation in a physical package.

To use Remote Data Service technology, you must understand the three-tiered client/server model. This model separates the various components of a client/server system into three "tiers":

- Client tier—a local computer on which either a Web browser displays a Web page that can display and manipulate data from a remote data source, or (in non-Web-based applications) a stand-alone compiled front-end application.
- Middle tier—a Microsoft® Windows NT® Server computer that hosts components that encapsulate an organization's business rules. Middle-tier components can be either Active Server Page scripts executed on Internet Information Server, or (in non-Web-based applications) compiled executables.
- Data source tier—a computer hosting a database management system (DBMS), such as a Microsoft® SQL Server™ database. (In a two-tier application, the middle tier and data source tier are combined.)


These tiers don't necessarily correspond to physical locations on the network. For example, all three tiers may exist on only two computers. One computer could be a Microsoft® Windows® 95 computer running Microsoft® Internet Explorer 4.0 as its browser. The second computer could be a Windows NT Server computer running both Internet Information Server and Microsoft SQL Server. Designing applications this way gives you greater flexibility when deploying processes and data on the network for maximum performance and ease of maintenance.

20.7 Registering Visual Basic DLLs With Transaction Server

After you create your ActiveX DLL project, you need to register it with Transaction Server before it can be used. You do this through the Transaction Server Explorer. After you have started up the Transaction Server Explorer, you need to make sure that the DTC is running. If the DTC is not running, click the Computer icon for your computer and then choose Tools /MS DTC/Start

- Creating Packages
- Installing Components
- Calling Transaction Server Objects from Visual Basic

20.8 Short Summary

 DLLs can define two kinds of functions: exported and internal. The exported functions can be called by other modules. Internal functions can only be called from within the DLL where they are defined. Although DLLs can export data, its data is usually only used by its functions.

- ☞ The difference between VB procedures and the Win32 API procedures is that API procedures must be declared before they can be used.
- ☞ Visual Basic can't verify the arguments that are passed to a DLL procedure. If incorrect values are passed the procedure may fail, which may cause Visual Basic application to stop. So, care should be taken while experimenting with DLL procedures.
- ☞ Microsoft Transaction Server is a component-based transaction processing system for developing, deploying, and managing high performance, scalable, and robust server applications.
- ☞ We can use an ActiveX control to read and update the Windows registry directly, or it can call system-level functions by linking to one of the DLLs in Windows.

20.9 Brain Storm

1. Define DLL - Explain its advantages.
2. What are the Advantages of ActiveX Controls in Visual Basic?

Lab Unit - 20 (2 Real Time Hrs)

Create a DLL which have to reverse a string

Lecture - 21

Packages and Internet Information Server

Objectives

In this lecture you will learn the following

- ❖ About Creating Packages
- ❖ About DLL calling Convention
- ❖ About Internet Information Server
- ❖ Introduction about IIS & ASP

Lecture Unit - 21

- 21.1 Snap Shot
- 21.2 Creating Packages
- 21.3 Installing Components
- 21.4 Calling Transaction Server Object from VB
- 21.5 Internet Information Server
- 21.6 Introduction to IIS Vs ASP
- 21.7 Short Summary
- 21.8 Brain Storm

Lab Unit - 21 (2 Real Time Hrs)

21.1 Snap Shot

Before you can start registering components in Transaction Server, you must know the concept package into which you are going to install the components. Packages are logical groupings of objects that are generally used as a unit. As a general rule, you will want to create one package for each set of applications that makes use of Transaction Server.

21.2 Creating Packages

Before you can start registering components in Transaction Server, You can create a package by following these steps:

1. Select the Packages Installed folder
2. Choose File /New from the main menu
3. On the first screen of the Packages Wizard, choose an Empty Package, as seen in Figure.
4. Type in a name for the package, as shown in Figure. For your catalog sales company package, call it Inventory. Then click the Next button
5. If the objects in the package need to run under a specific login account, select the This User radio button and provide the username and password. Otherwise, leave the default radio button selected as in Figure, which runs all of the objects in the package under the account of the user using the applications that use the objects in this package. (This can affect the availability of resources for the process components, depending on how the access privileges are configured in the system security.)
6. Click the Finish button to complete the process

21.3 Installing Components

After you have a package, you can begin installing components into it. This is where you register the ActiveX DLLs that you have and will be creating with Visual Basic 6. You can install your components by following these steps:

1. Select the Components folder in the package into which you want to install the components, as seen in Figure
2. Click the Install New Components(s) button. As seen in Figure

3. Click the Add Files button and select the ActiveX.Dll that you want to install, as seen in Figure
4. When you return to the Install Components dialog box, the upper list box should show the DLL that you are installing, and the lower list box should show all of the visible classes within the DLL, as in Figure
5. Click the Finish button and the components are installed in Transaction server, as seen in Figure.
6. Select the components you just installed, one at a time, and right-click the mouse. Select Properties from the pop-up menu. On the component Properties Editor, select the Transaction tab, and select the transaction attribute desired for the currently selected component, as in Fig

Whenever you recompile any ActiveX DLL built in VB6, you need to refresh the component information in Transaction Server. This can be easily done by deleting the component from the Transaction Server Explorer and reinstalling the component by following the same steps as were followed to install the component originally. Another way of refreshing the component information in Transaction Server is choosing Tools/Refresh All Components.

21.4 Calling Transaction Server Objects From Visual Basic

Now that you have a component built and registered with Transaction Server, how do you call the method in this object from a Visual Basic application? First, the Visual Basic application has to get a reference to the Transaction Server Object. After the reference has been acquired, the object's methods can be called. There are three ways that a references to a Transaction Server Object can be acquired:

- Using the CreateObject function
- Using the GetObject function
- Using the New Keyword

Note: There is a fourth method for acquiring a reference to a Transaction Server object by another Transaction Server object; that is through the use of the Context Object's CreateInstance method, which creates the object reference in the same transaction context of the current object (depending on the new object's transaction attribute).

From a Visual Basic application, you can create your reference to your Inventory Maintenance object with the following code:

```
Dim obj As Object
```

```
Set obj = CreateObject("InvMaint.InvMnt")
```

From here, you can call the object methods by referencing them via the object you have just created as so:

```
Obj.ChangeInv(iiProductID,CLng(txtCount.Text),iBackOrder,iCurInventory)
```

If you build a simple little applet using the Remote Data Control and the Data Bound Combo Box, you can call your object and see how Transaction Server works. The first thing you need to do is to add the Remote Data Control and the Data Bound List Controls to your Toolbox. You do this choosing Project/Components, as seen in Fig 21.1

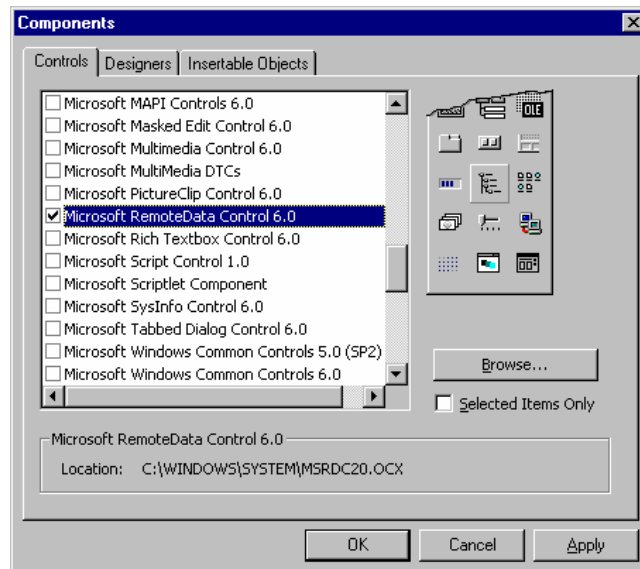


Figure 21.1 - Include Remote Data Control & Data Bound List Controls in VB project

With these controls, you can build a simple form that provides a drop-down list box, containing the product descriptions from the Products table in the database. You accomplish this binding the Remote Data Control to the ODBC configuration you

have set up for your database, providing the username and password that you have configured, and using the following SQL to populate the Remote Data Control:

```
SELECT * FROM Products
```

Next, specify the Remote Data Control as the row source for the Data Bound Combo Box and specify the Prd_Desc column as the ListField, and BoundColumn. Add a text box in which the user can enter the number to add or remove from the inventory and you have the form seen in Figure 21.2

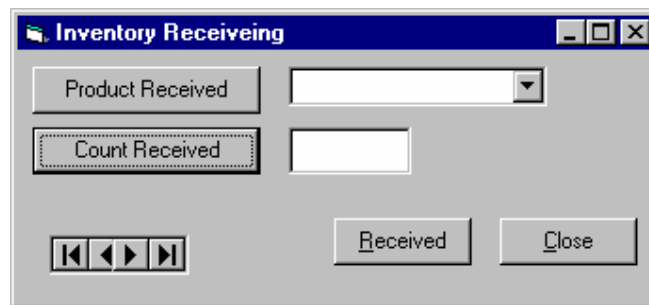


Figure 21.2 – Form to call the method in the object you registered with Transaction Server

Setting the Product ID

When the user selects a product, you need to have a variable into which to place the selected product ID, so declare a variable in the form declarations using the code in the List

Declaring A Variable For Holding The Selected Product ID

Option Explicit

'We always return the same error number

```
Private Const ERROR_NUMBER = vbObjectError + 0
```

'The currently selected product ID

```
Dim iiProductID As Long
```

You set the product ID into this variable whenever the user selects a product from the list by navigating in the Remote Data Control to the currently selected row, and then getting the Prd_ID column from that row, as in the List

Receiving.frm - grabbing the product id from the remote data control when the user selects a product

```
Private Sub dbcProduct_Click(Area As Integer)
    Dim varCurRecord As Variant

    ' What is the currently selected item?
    VarCurRecord = dbcProduct.SelectedItem

    ' Move to the selected record in the result set
    MSRDC1.Resultset.Move 0, varCurRecord

    ' Grab the product ID
    iiProductID = MSRDC1.Resultset!Prd_ID.Value

End Sub
```

Calling the Transaction Server Object

When the user clicks the Received button, you can create a reference to the Transaction Server Object that you created by using the earlier code snippets, and you can display for the user the resulting stock and back-order amounts with the code in the list

Creating a reference to your transaction server object and calling the method that you created in the object

```
Private Sub cmdReceived_Click()
    `Dim iBackOrder As Integer
    Dim iCurInventory As Integer
    Dim ProgID As String
    Dim obj As Object

    ' Set up the error handling
    On Error GoTo ErrorHandler
```

```
' Decide which component to use
ProgID = " InvMaint.InvMnt"

'Create the appropriate object
Set obj = CreateObject(ProgID)
'Were we able to create the object?
If obj Is Nothing Then
    MsgBox "Create object" + progID + "failed."
    Exit Sub
End If

' Call the object method
If obj.ChangeInv(iiProductID, CInt(txtCount.Text), iBackOrder,_
                iCurInventory) = -1 Then
    Err.Raise ERROR_NUMBER
End If

' Release the object
Set obj = Nothing

' Display for the user what the current inventory is
MsgBox "New Inventory received, current backorder count =" _
        + Str$(iBackOrder) + " and current inventory =" _
        + Str$(iCurInventory)

Exit Sub
```

ErrorHandler:

```
' Show the user the error message
MsgBox "Error" + Str$(Err.Number) + " :" + Err.Description
Exit Sub
```

End Sub

Before you run your form, let's change the view in the Transaction Server Explorer to show the status of your object. You do this by selecting the Components folder in the package you have created. Next choose View/Status. The right side of the Explorer now shows the activity status of your object. If you run the form with the Transaction Server Explorer where it can be seen, you can watch as the object that you created earlier is instantiated and executed

21.5 Internet Information Server

Microsoft Internet Information Server is a network file and application server and is included with Windows NT server. IIS supports Hypertext Transfer Protocol(HTTP), File Transfer Protocol(FTP) and gopher information protocols.

An IIS (Internet Information Server) application is a Visual Basic application that lives on a Web server and responds to requests from the browser. An IIS application uses HTML to present its user interface and uses compiled Visual Basic code to process requests and respond to events in the browser.

To the user, an IIS application appears to be made up of a series of HTML pages. To the developer, an IIS application is made up of a special type of object called a webclass, that in turn contains a series of resources called webitems. The webclass acts as the central functional unit of the application, processing data from the browser and sending information to the users. You define a series of procedures that determine how the webclass responds to these requests. The webitems are the HTML pages and other data the webclass can send to the browser in response to a request.

21.6 Introduction to IIS & ASP

As developing DHTML documents are eased in Visual Basic through *DHTML application* project type, developing ASP documents are also eased in Visual Basic through *IIS application* project type. This section explains about IIS Application project type and shows how to use it.

Web Classes

A Web Class is a Visual Basic component that resides on a Web server and responds to input from the browser. When an IIS Application project type is opened, Visual

Basic opens a Web designer that creates the Web Class. Webclasses typically contain WebItems (HTML files) and code that delivers those Webitems to a client.

A webclass in an IIS application has an associated .asp (Active Server Pages) file that Visual Basic generates automatically during the compile process. The .asp file hosts the webclass on the Web server and launches its scripts for execution.

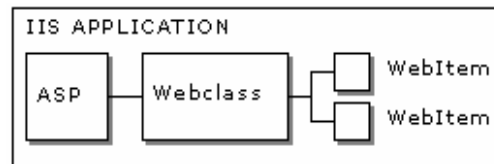


Figure 21.3 Relationship between Webclass and ASP file.

As shown in this figure 21.3, each webclass has its own ASP. In turn, a webclass can have many webitems (HTML files) associated with it.

Advantages of IIS application

- ❖ Reduced cost of deployment per user. End users of an IIS application can run the application using only a browser; no special software needs to be installed on their computers for the application to work.
- ❖ A familiar development environment and model for Visual Basic programmer.
- ❖ Access to a broad audience. IIS applications work with a wide variety of browsers and operating systems, so it can easily reach a wide audience.
- ❖ It can send HTML pages either from Templates or from code to the browser.

Note : Templates are nothing but Webitems that holds designed HTML pages.
A Template can have only one HTML page.

21.7 Short Summary

- ☞ Packages are logical groupings of objects that are generally used as a unit.
- ☞ Visual Basic makes it easy to integrate Web browsing into the application by way of the Web browser control, which is used to add Web-browsing functionality to the Visual Basic application.

- ☞ Templates are nothing but Web items that holds designed HTML pages. A Template can have only one HTML page

21.8 Brain Storm

1. What are the Internet tools available in Visual Basic?
2. Explain Internet Information Server.
3. What are the advantages of IIS application?
4. What are the difference between IIS Vs ASP applications?

Lab Unit - 21 (2 Real Time Hrs)

Create a IIS Application and accept user name and password and check whether the password is "Radiant" or not ?.Raise an appropriate Error Message if Password is wrong.

IIS, ASP and DHTML Application Comparison

Objectives

In this lecture you will learn the following

- ❖ Applications of IIS,ASP, and DHTML
- ❖ About IIS application development process
- ❖ Debugging IIS application

Lecture Unit - 22

- 22.1 Snap Shot
- 22.2 IIS Vs ASP applications
- 22.3 IIS Vs DHTML Applications
- 22.4 IIS Development process
- 22.5 Debugging IIS Applications
- 22.6 Short Summary
- 22.7 Brain Storm

Lab unit 22 (2 Real Time Hrs)

22.1 Snap Shot

In this chapter the discussion will go on application of IIS , ASP and DHTML then Debugging the IIS application and development process.

22.2 IIS Applications VS. ASP Applications

IIS applications bear a superficial resemblance to Active Server Pages applications. Both types of applications present dynamic Web sites and perform their processing on the server rather than the client. However, each has its unique advantages. Active Server Pages are for script developers interested in authoring Web pages, and offer the unique capability of intermingling script with HTML. IIS applications are for Visual Basic developers building Web-based applications, rather than Web pages. IIS applications allow for complicated business processing and easy access from almost any browser or platform.

22.3 IIS Applications VS. DHTML Applications

IIS applications are similar to DHTML applications in one way. Both arrange the contents of Web documents dynamically. However, there are some key differences between the two types of applications:

- ❖ **Dependency** – DHTML applications are dependent on Internet Explorer 4.0 or later. But end users of an IIS application do not need a specific operating system or browser.
- ❖ **Location of processing** – IIS applications are designed to perform most of their processing on the Web server, but DHTML applications perform most of their processing on the browser machine.

Webclass designer

When a project of type IIS Application is opened, Visual Basic presents WebClass designer as in Figure 22.1.

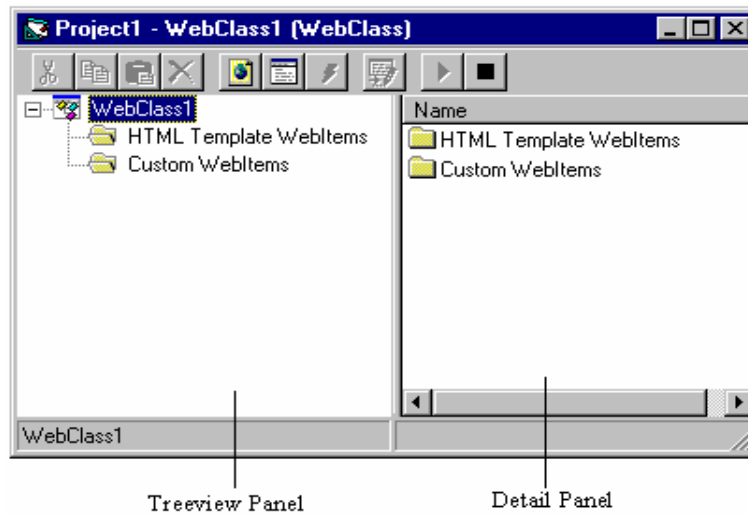


Figure 22.1 Webclass designer.

The **Treeview panel** displays each webitem and connected event in the webclass. There are two kinds of webitems:

- ❖ HTML Template WebItems – HTML files that can be sent to the browser in response to a request
- ❖ Custom WebItems – programmatic resources that creates HTML files on the fly.

The **Detail panel** displays information about the currently selected item in the Treeview panel.

22.4 IIS Application Development Process

The process of creating an IIS application is similar to creating any other project in Visual Basic. The overall process is presented below.

The overall process for creating IIS applications is:

1. Start a new project and select IIS Application as the project type.
2. Save the project.

Note : Unlike forms-based Visual Basic applications, you must save an IIS application before you add HTML template webitems to it.

3. Add as many HTML template webitems and custom webitems to the webclass as needed.
4. Add any custom events to the project.

5. Write code for all standard, template, and custom events in the project.

Note : Make sure you write code for the Start event, or your application will not run unless the user specifies a webitem in the base URL. For more information see "Setting the Start Event".

6. Add other code modules or webclass objects to the project.
7. Test and debug the application by running the project and viewing the application in the browser. It is recommended that you test all browsers you plan to support before releasing the application to end-users.
8. Compile the project.
9. Deploy the application.

Example

This section shows how to create an ASP file using Visual Basic's IIS Application project type. Before creating the IIS Application create an HTML file **Secret.htm** whose code listing is given below.

```
<HTML>
<BODY>
<CENTER>
<H2><U>Now You are in the Secret Site</U><H2>
</CENTER>
</H2>
</BODY>
</HTML>
```

This file will be used as a webitem in the IIS Application project.

Creating a new IIS project

- ❖ Choose File | New Project to open the New Project dialog box.
- ❖ Double-click the IIS Application icon to create an IIS Application project.



IIS Application Icon

- ❖ Name the project as IISPro.
- ❖ In the Project Window double click the Webclass1 to bring its designer front.
- ❖ Set the properties of the Webclass1 as given below:

Property	Value	Description
Name	wclsSecret	Specifies the name of the web class.
NameInURL	keySecret	Specifies the name of the web class as it appears in the address that invokes the webclass. That is, its corresponding ASP page will be created as keySecret.ASP (key to Secret) which is used in the address string.

- ❖ Save the project with the default names. (It is very important to store the project in a specific folder. Then alone the HTML files that are added as webitems can be stored in the Project folder).
- ❖ Right click on the **HTML Template Webitems** in the Treeview panel and choose Add HTML template from the popup menu that appears.
- ❖ In the dialog box that appears browse and select Secret.htm and click open. This will add a template under the **HTML Template Webitems** in the Treeview panel as Template1. Rename it as SecretHtm.
- ❖ Open the code window of the Web class designer.
- ❖ Replace the code of the Web class start event procedure {WebClass_ Start()} with the following code.

```

Private Sub WebClass_Start()
    Dim Param
    Set Param = Request.QueryString

    If Param("Name") = "Indian" Then
        'Rendering the Secret Site
        SecretHtm.WriteTemplate
    Else
        'Writing a reply to the user
        With Response
            .Write "<html>"
        End With
    End If
End Sub

```

```
.Write "<body>"
.Write "You are not an authorised user to go to_
      the Secret Site"
.Write "</body>"
.Write "</html>"
End With
End If
End Sub
```

This event will occur when the keySecret.ASP invokes this web class. This procedure checks whether visitor is an authorized user or not. If he is an authorized user, it sends the contents of the template file to the browser using the WriteTemplate method. Otherwise it creates an HTML page, which indicates that he is not an authorized user, and sends it to the browser.

- ❖ Choose File | Make IISPro.dll to compile the project. Visual Basic will create keySecret.ASP file.
- ❖ Create an HTML **HTM.htm** file whose code listing is given below. This file will call the keySecret.ASP file.

```
<HTML>
<BODY>
Type your name and click Submit<BR>
to go to the Secret Site.
<BR><BR>
<FORM Method="Get"
Action = "http://127.0.0.1/ASPs/KeySecret.ASP">
<INPUT Type=Text Name="Name"><BR><BR>
<INPUT Type=Submit Name="Submit" Value=Submit>
</FROM>
</BODY>
</HTML>
```

The string **/ASPs** specifies the virtual directory which is mapped to the project directory in which the KeySecret.ASP file is stored.

- ❖ Open the Htm.htm in the Internet Explorer, which will resemble the Figure 22.2

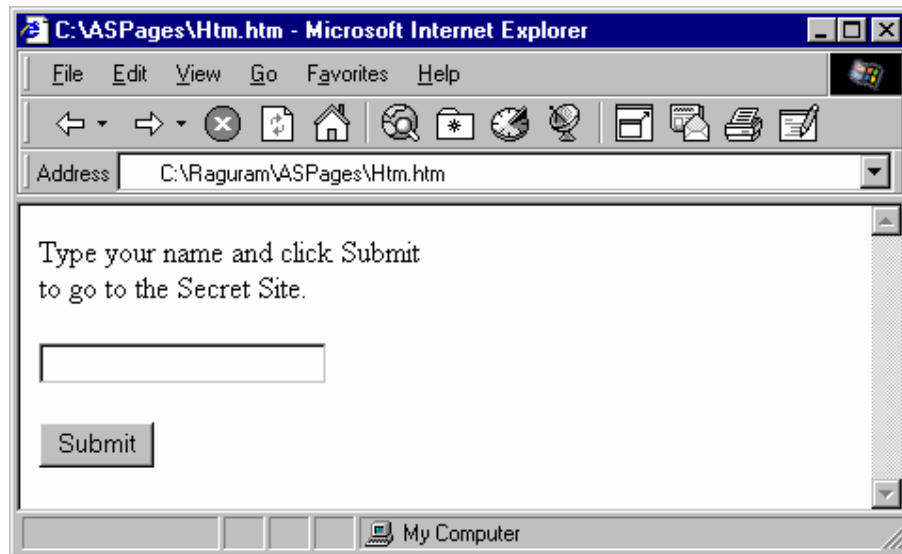


Figure 22.2 HTM.htm file in the Internet Explorer.

- ❖ Type **Indian** in the textfield and click the submit button. This will call the keySecret.ASP file, which invokes the web class. The web class verifies the entered name. As it is Indian (the valid name), it will supply the Secret.htm file, which is stored in the SecretHtm template. The browser receives this file and renders it as in Figure 22.3.

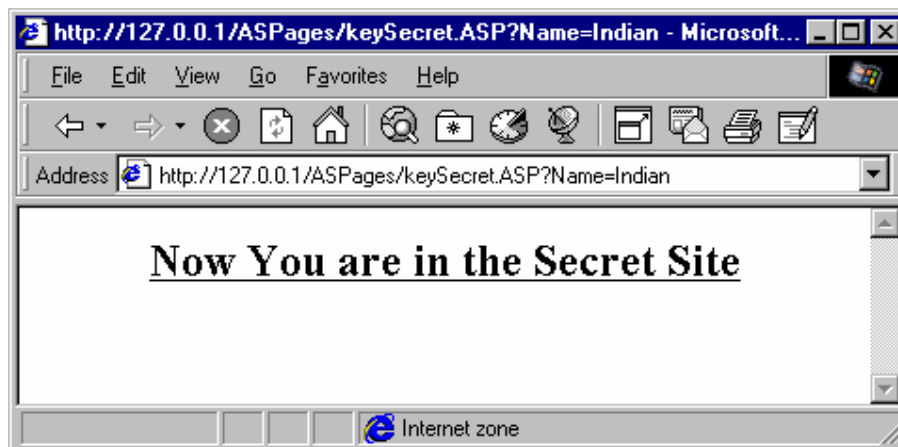


Figure 22.3 HTML page sent by the ASP file when the user is an authorized person.

If some other name is typed and the Submit button is clicked, the web class creates HTML page on the fly indicating that the user is not an authorised user and sends it to the browser. The browser renders it as in Figure 22.4.

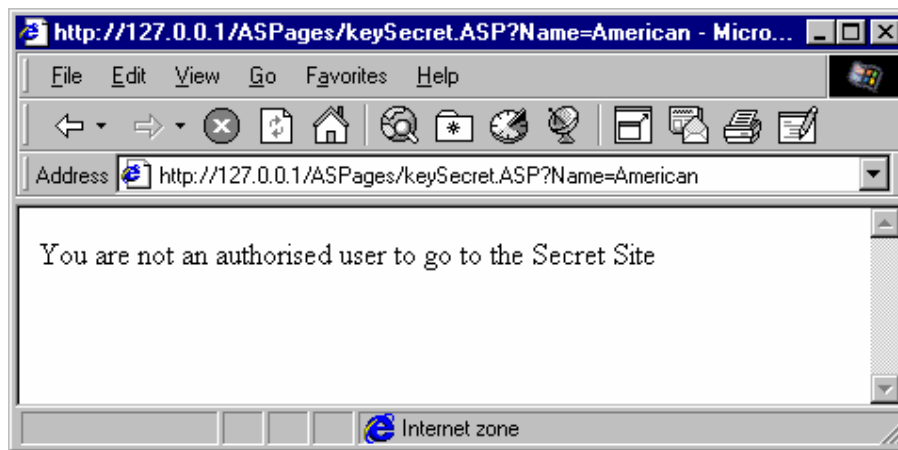


Figure 22.4 - HTML created and page sent by the ASP file when the user is not an authorised person.

22.5 Debugging Your IIS Application

You debug an IIS application in the same way you do any other Visual Basic application – by entering run mode from Visual Basic. Visual Basic loads the webclass run time, creates the virtual root from which to run the .asp file for the application, if necessary, and launches the system's default browser with an HTTP reference to the .asp file. The .asp file, in turn, launches the webclass.

Note Although you can view the .htm files associated with your application in the browser by opening them from the browser's File menu, this is not debugging your application. You must use the Start option from Visual Basic to enter debugging mode.

When you debug, you have the full Visual Basic development environment at your disposal. You can use all the tools available in Visual Basic – breakpoints, watch variables, debug statements, and so on – to debug your project.

Visual Basic prompts you when you debug that it is going to create a virtual directory for your project. A virtual directory is a directory outside your Web server's home directory that appears to browsers as a subdirectory of the home directory. It allows you to publish contents to the Web from directories outside the home directory structure. You cannot change the location of the virtual directory Visual Basic creates for the webclass.

The Project Properties dialog box's Debugging panel settings determine whether the system waits for you to tell it what to do when you go into run mode or automatically starts the webclass you specify. When you choose to automatically

start the webclass, Visual Basic launches Internet Explorer, navigates to the URL for your application, and fires the webclass's BeginRequest event.

Visual Basic deletes all temporary files when it comes out of run mode. In addition, it destroys the instance of the Webclass Designer and restarts the designer in design mode.

Errors in Webclasses

You can use Visual Basic's error-handling features in your IIS applications to trap errors and take corrective action. When an error occurs, Visual Basic sets the various properties of the error object, Err, such as an error number or a description. You can use the Err object and its properties in an error-handling routine so that your application can respond intelligently to an error situation.

In addition to standard error handling, IIS applications allow you to use two special features to handle errors:

- You can use the Trace method to debug your application on the production computer.
- You can use the FatalErrorResponse event to respond to serious run-time errors.

Using the Trace Method

You can use the Trace method to help identify errors during the debug process and to track performance and statistical data. The Trace method sends a specified string to the Win32 OutputDebugString API. The string can then be captured to a suitable debugging tool such as DBMON.

Using the Trace method can allow you to debug on your production server computer and record useful information such as information about the execution of the application, error messages that occur, and any other information you need.

Handling Fatal Errors

A fatal error on a webclass is one from which the application cannot recover or restore the appropriate webitem. For example, a fatal error might be an unhandled error within a webclass event, a structural error, or an unexpected error within the run-time DLL. Following such an error, the webclass run time fires the FatalErrorResponse event. The application is terminated and the instance of the webclass is destroyed.

When a fatal error occurs, the application can write a message to the Response object in the handler for the FatalErrorResponse event. This message can be one that you write, or it can be the default message for the .asp file associated with the webclass. To write your own message, use the Response object, then set the senddefault argument of the FatalErrorResponse event to False. To use the default error message, leave the senddefault argument set to True.

Note :- The webclass run time provides an Error property that is only available from within the FatalErrorResponse event. This property returns an object that describes the error that caused the webclass to terminate.

The webclass run time also logs fatal errors to the NT event log. On Windows 95 systems, the run-time DLL creates a log file in the Windows directory and logs the error there.

22.6 Short Summary

- ☞ A fatal error on a webclass is one from which the application cannot recover or restore the appropriate web item
- ☞ Visual Basic deletes all temporary files when it comes out of run mode. In addition, it destroys the instance of the Webclass Designer and restarts the designer in design mode.
- ☞ DHTML applications are dependent on Internet Explorer 4.0 or later. But end users of an IIS application do not need a specific operating system or browser.
- ☞ IIS applications are for Visual Basic developers building Web-based applications, rather than Web pages.

26.7 Brain Storm

1. What are all the IIS application development process?
2. How an IIS application development can be debugged

Lab Unit - 22 (2 Real Time Hrs)

<<<< Test to be Conducted >>>>

Lecture - 23

Internet concepts

Objectives

In this lecture you will learn the following

- ❖ About Internet
- ❖ About Intranet
- ❖ About world wide web
- ❖ About Internet Explorer

Lecture Unit - 23

- 23.1 Snap Shot
- 23.2 Internet concept
- 23.3 Intranet
- 23.4 World wide web
- 23.5 Internet explorer
- 23.6 Working in the Internet
- 23.7 Short Summary
- 23.8 Brain Storm

Lab Unit - 23 (2 Real Time Hrs)

23.1 Snap Shot

If there is one technology that caught up literally overnight and has affected more users than any others, it is the Web. The World Wide Web (WWW) is the set of all Web sites and the documents they can provide to clients (users). Visual Basic 6 has evolved to help the programmers to build Web applications. There are two different types of Web applications – Dynamic HTML (DHTML) applications and Microsoft Internet Information Server (IIS) applications.

23.2 Internet Concepts

This section briefs essential Internet concepts as follows.

Internet

The Internet is a global network of computers that communicate using a common language. It is similar to the international telephone system – no one owns or controls the whole thing, but it is connected in a way that makes it work like one big network. The Internet use a common protocol to communicate – TCP/IP (Transmission Control Protocol/Internet Protocol). TCP/IP is a simple protocol because it had to be implemented consistently on all computers and operating systems. Indeed, TCP/IP is a truly universal protocol, it's there when you need it and allows your computer to connect to any other computer on the Internet.

Each computer on the Internet has a unique address, for example, 193.25.84.100. Each number is a value in the range 0 through 255, which means the Internet have more than $256*256*256*256$, or approximately 4,000,000,000 computers. To accommodate a large number of users, Internet service provider use a pool of addresses(since not all users connect at once, 256 addresses may accommodate 100 users or more. It would be nice if we all had a unique IP address, like an e-mail address, but this is not possible. If we did, we could build wide area networks that span the globe easily. However, every time you connect to your Internet service provider you get a different IP address

23.3 Intranet

Unlike Internet, which is a global network, it is a private network. But it uses the Internet communication standards and tools to provide information to the restricted users. For example, a company may setup a Web site that is accessible only to its employees who are geographically separated. On an intranet, you can exploit the web model to simply operations, without the security issues you face on the Internet or the limitations imposed by connecting computers with modems.

An intranet is also a network of computers operating on the TCP/IP protocol, but it is not global. Intranet are restricted to the users of a corporation, a university, or

some other organization, and they are not accessible by the outside world. Unlike the World Wide Web, intranets don't have more than one server. This machine supplies all the documents requested by the clients. Many corporations use intranets to provide information to their employees, and they run another Web site for external users. The reason for building corporate intranets is to exploit the technology that made the Web so popular. So the main characteristic of a corporate intranet is not that it uses TCP/IP, as much as HTTP (HyperText Transfer Protocol)

23.4 World Wide Web

The World Wide Web (WWW or simply the Web) gives a graphical, easy-to-navigate interface for looking at documents on the Internet. These documents, as well as the links between them, comprise a "Web" of information.

Files or pages on the Web are interconnected. This connection is made by means of special text or graphics called *hyperlinks*.

Pages can contain text, images, movies, and sounds – just about anything. These pages can be located on computers anywhere in the World. When a connection to the Internet is made, it means equal access is available to information worldwide.

The computers that host Web sites are called *servers*; their service is to provide the documents that the clients request. *Clients* are the seemingly endless numbers of personal computers connected to the Internet. Web browser, such as Internet Explorer (explained in the next section), helps to exploit the Web.

23.5 Internet Explorer

Microsoft Internet Explorer is a Web browser. Just as Microsoft Word is a tool to create and format documents, or Microsoft Excel is a tool to create spreadsheets and perform calculations, Internet Explorer is a tool to navigate and access, or "browse," information on the Web.

The Internet Explorer toolbar provides a range of detailed functions and commands for managing the browser. The addressbar below the toolbar displays the address of the current Web page. Typing an address and hitting Enter in it will enable to visit to the specified page. Clicking a hyperlink will also enable to jump to the new page.

With Netscape dominating the Web browser and server market, Microsoft, in the early 1996 introduced a new Web browser and server to the Internet community. Microsoft Internet Explorer is destined to be one of the most popular Web browser for several reasons. Not only is the browser easy to use and is supported by the world's largest software company, it also supports several other specific HTML extensions. These include the following

- ◆ Background sounds played automatically when the Web page is loaded

- ◆ Inline animations of AVI files instead of graphic images
- ◆ Marquees that scroll across the browser window

23.6 Working of the Internet

For a computer to be a server on the Internet, it must have two things: an address by which other computers can locate it and the capability to understand and process the various protocols. A server is assigned a unique numeric ID called an IP (Internet Protocol) address. As the numeric address is very difficult to remember their corresponding *domain name* is used, which are in friendlier format such as <http://www.microsoft.com>.

The software used on the physical computer to make it a server that can speak the protocols of the Internet and respond accordingly is called *Internet Server Software* or *Web Server* such as Microsoft Internet Information Server.

For a client computer to be able to communicate with a server on the Internet, it must have a connection to the Internet. Then, when connected, it must have a way to contact and receive data from Internet servers through the various protocols. The connection is accomplished via an Internet Service Provider (ISP), such as VSNL, Satyam Online, etc. The tool to communicate to the server and receive the data returned by the Internet Server is handled by the Internet browser, such as Internet Explorer. The Figure 5.1 depicts the working of the Internet.

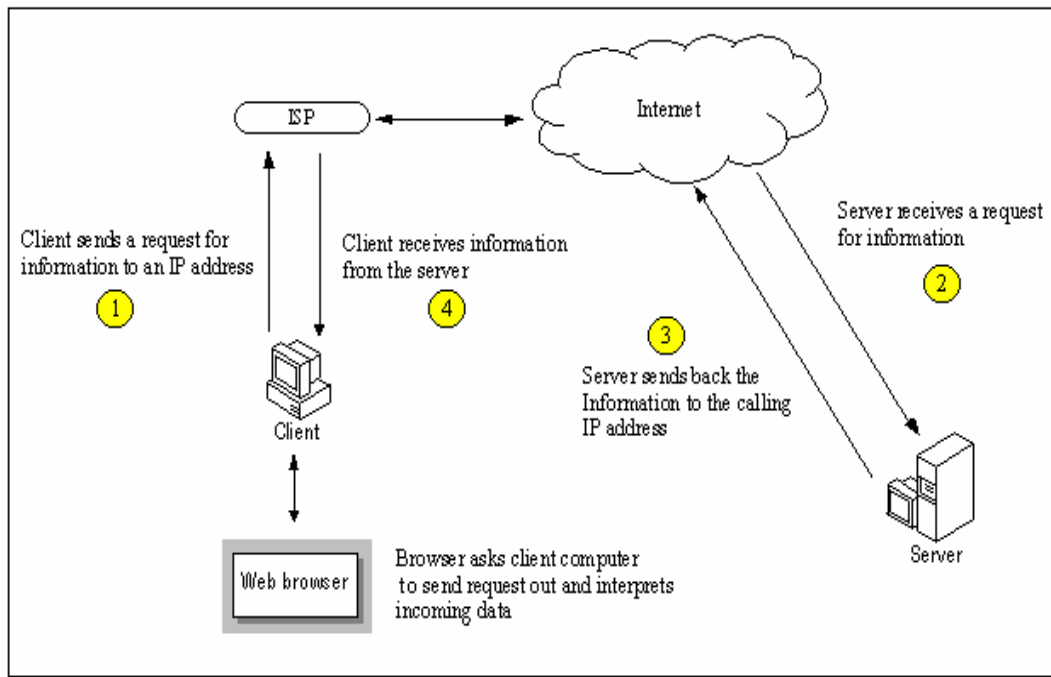


Figure 23.1 Working of Internet.

23.6 Short Summary

- ☞ The Internet is a global network of computers that communicate using a common language. It is similar to the international telephone system.
- ☞ An Intranet is also a network of computers operating on the TCP/IP protocol, but it is not global.
- ☞ The World Wide Web (WWW or simply the Web) gives a graphical, easy-to-navigate interface for looking at documents on the Internet. These documents, as well as the links between them, comprise a “Web” of information.
- ☞ The Internet Explorer toolbar provides a range of detailed functions and commands for managing the browser. The address bar below the toolbar displays the address of the current Web page.

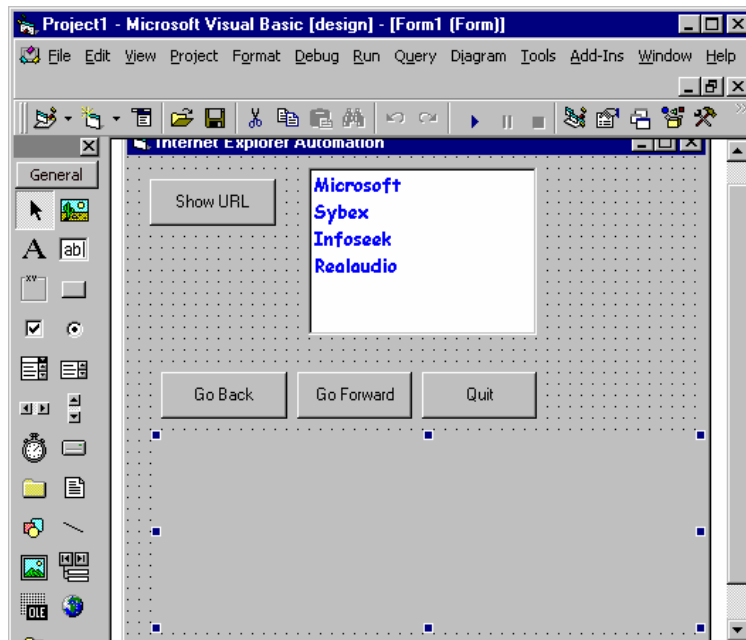
23.7 Brain Storm

1. Discuss briefly Internet concepts, working of Internet, Intranet and Internet Explorer.

Lab Unit - 23 (2 Real Time Hrs)

Create an Internet Explore project and make it online

Design your form as shown below



Lecture - 24

DHTML

Objectives

In this lecture you will learn the following

- ❖ About DHTML
- ❖ About Server
- ❖ About Client / Server computing
- ❖ Client / Server Model

Lecture Unit - 24

- 24.1 Snap Shot
- 24.2 Developing a DHTML application
- 24.3 DHTML application in VB
- 24.4 Structure of DHTML application
- 24.5 Advantages of DHTML application
- 24.6 Short Summary
- 24.7 Brain storm

Lab Unit - 24 (2 Real Time Hrs)

24.1 Snap Shot

This chapter is discuss on HTML developing , its structure and its applications.

24.2 Developing an Dynamic HTML Document

This section develops a simple DHTML document, which has 5 major segments (designed using DIV tags). The first segment has one password box and a command button. This segment alone is visible when the document is loaded in browser and all other segments are invisible. When a user enters a password and clicks the command button, the script written for the command button's click event, verifies the validity of the entered password. If it is valid the first segment disappears and second segment appears, which has three option (radio) buttons. The other three segments are designed for each option button and they are positioned at a same location. Because only one option button can be selected, the segment corresponding to that option alone is made visible and other two segments are made invisible. This DHTML document is named as **India.htm**, as it describes about Indian Monuments and its listing is given below.

```
<HTML>
<HEAD>
<TITLE>
Indian Monuments
</TITLE>

<SCRIPT Language = "VBScript">

Sub Opt1_Onclick()
    divForOpt1.Style.Visibility = "Visible"
    divForOpt2.Style.Visibility = "Hidden"
    divForOpt3.Style.Visibility = "Hidden"
End Sub

Sub Opt2_Onclick()
    divForOpt2.Style.Visibility = "Visible"
    divForOpt1.Style.Visibility = "Hidden"
    divForOpt3.Style.Visibility = "Hidden"
End Sub

Sub Opt3_Onclick()
    divForOpt3.Style.Visibility = "Visible"
    divForOpt1.Style.Visibility = "Hidden"
```

```
        divForOpt2.Style.Visibility = "Hidden"
    End Sub

    Sub cmdLogin_onClick()

    If frmLogin.txtPWord.Value = "Indian" then
        MsgBox "You are logged In"
        divLogin.Style.Visibility = "Hidden"
        divOptions.Style.Visibility = "Visible"
        divForOpt1.Style.Visibility = "Visible"
    Else
        MsgBox "Password Incorrect"
    End If

    End Sub

</SCRIPT>
</HEAD>

<BODY background="backgrd.gif">
<CENTER>
<H2>INDIAN MONUMENTS</H2><HR>
</CENTER>

<DIV Id=divLogin>
<FORM Id=frmLogin>
<LABEL>Password
<INPUT Type=Password Id=txtPWord>
</LABEL>
<INPUT Type=Button Id=cmdLogin Value="Login">
</FORM>
</DIV>

<DIV Id=divOptions Style="Position:Absolute;
    Left:20;Top:90;Visibility:Hidden">
    Choose any one of Monuments <BR>
    to view its Description

    <DIV Style="Position:Relative;Top:10;Width:130;Height:80;
        Background-color:Yellow">
    <FORM Id=frmOptions>
    <INPUT Type=Radio Name=Monument Id=Opt1 Checked>Taj
```

```

    Mahal<BR>
    <INPUT Type=Radio Name=Monument Id=Opt2>Qutub Minar<BR>
    <INPUT Type=Radio Name=Monument Id=Opt3>Fatehpur Sikri<BR>
    </DIV>

</FORM>
</DIV>

<DIV Id=divForOpt1 Style="Visibility:Hidden">
<IMG Src="C:\Raguram\Images\TajMahal.Gif"
    Style="Position:Absolute;Left:275;Top:85">

<DIV      Style="Position:Absolute;Left:20;Top:240;Width:450;Background-
color:Yellow">
One among the 7 World wonders.  Built by Shajahan at the riverbank of
Yamuna.
</DIV>
</DIV>

<DIV Id=divForOpt2 Style="Visibility:Hidden">
<IMG          Src="                C:\Raguram\Images\QMinar.Gif"
Style="Position:Absolute;Left:275;Top:85">

<DIV Style="Position:Absolute;Left:20;Top:240;Width:450;
    Background-color:Yellow">
Astonishing tower with height 242 feet.  Started by Quthputheen and successfully
completed by Eldhuthmish.
</DIV>
</DIV>

<DIV Id=divForOpt3 Style="Visibility:Hidden">
<IMG          Src="                C:\Raguram\Images\FSikri.Gif"
Style="Position:Absolute;Left:275;Top:85">

<DIV Style="Position:Absolute;Left:20;Top:240;Width:450;
    Background-color:Yellow">
Instance for sculpting art.  Nick named as love designed in stone.  Built by
great emperor Akbar.
</DIV>
</DIV>
</BODY>
</HTML>

```

When this HTML page is opened in the Internet Explorer it will prompt to enter the password as in Figure 24.1.

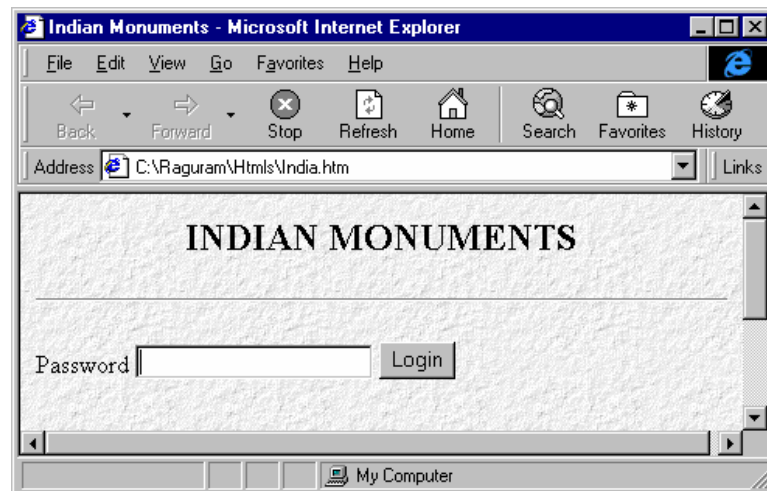


Figure 24.1 India.htm - prompting the visitor to enter the Password.

Enter the password **Indian** and click **Login** button. Now the first segment in this document that contains the textfield, button, will disappear and the other segments that describes about Indian Monuments will appear as in Figure 24.2

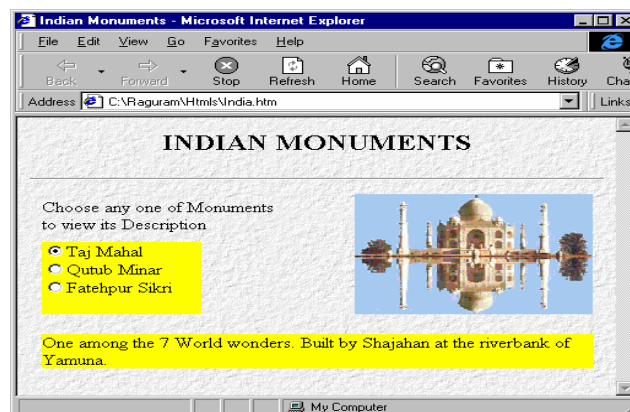


Figure 24.2 India.htm – Describing about Indian Monuments.

- ❖ Click any one of the options among the three to view their description.

24.3 DHTML application in Visual Basic

So far, web pages are created in non-visual environment (Notepad). Creating web pages in this method consumes much time and also positioning the controls and elements of a web page at proper location will cause the developer's eye turn to red. As so much of tags are used even to create simple web pages, there is a chance that few tags may be misspelled. And it is difficult to trace out the errors caused due to this misspelling.

Visual Basic 6.0 includes new project type called DHTML Application, which provides visual environment to rapidly develop the DHTML applications.

Web Pages vs. Forms

DHTML applications are structured differently than forms-based Visual Basic applications. In a DHTML application, the user interface consists of a series of HTML pages rather than forms. An HTML page is like a form in that it contains all the visual elements that makes up the application's user interface. Text box, Image, radio buttons, and check boxes can easily be placed on an HTML page at desired locations.

An HTML page is stored in a .htm file that is analogous to a .frm file. The following table sums up the differences between forms-based applications and Web-based applications:

	Forms-based application	Web-based application
User interface	Visual Basic forms	HTML pages
File format	.frm files	.htm or .html files, or generated from Visual Basic code
Creator	Developer	Web designer or developer
Run time	Visual Basic run-time DLL, msvbvm60.dll	Web browser with msvbvm60.dll

Table 24.1 Differences between Forms-based application and Web-based application.

24.4 Structure of DHTML Applications

DHTML applications are made up of the following pieces:

- ❖ One or more HTML pages.
- ❖ Visual Basic code that handles the events generated from the HTML pages.
- ❖ A project DLL that contains the Visual Basic code and is accessed by the run-time component generated automatically when the application is compiled.

When a DHTML project is compiled, all HTML pages are stored in separate .htm files. The code that handles the events generated from the HTML pages are compiled in to a .dll file. The id of the .dll file is automatically included in the .htm

files. So whenever an event raises from a .htm file, the code for that event in the .dll file is executed by the browser automatically.

24.5 Advantages of VB DHTML applications

Visual Environment HTML pages can be created in the familiar Visual Basic IDE. As it is a Visual environment, controls can be placed in the HTML pages at the desired locations easily. Debugging tools can also be used to fix the bugs in VB scripts.

Code Security When scripts are embedded with an HTML page, anyone can access the page, read the script, and make changes to it. As Visual Basic compiles the code into .dll, the code is not part of the HTML file itself, and so it can't be tampered.

24.6 The DHTML Page designer

When a project of type DHTML is opened, Visual Basic presents DHTML Page designer as in Figure 5.4, which enables to:

- ❖ Create a new HTML page, or edit the contents of an existing page.
- ❖ Determine which elements on the page are programmable and explore all the dynamic elements on the page.
- ❖ Access the Code Editor window to write code for each programmable element on the page.

The following sections brief the different areas of the DHTML Page designer.

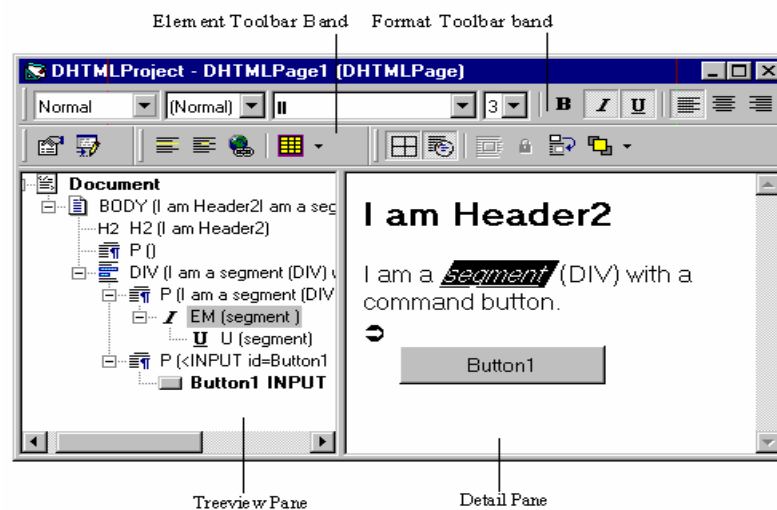


Figure 24.3 DHTML Page Designer.

Tree View Pane

Displays a hierarchical representation of all of the elements within an HTML page. For each element, the page designer lists the ID (if one exists), the type of control, and in some cases the beginning of the content for that element. Elements with IDs are indicated in bold. All elements in the treeview are children of the Body element and the Document object.

Detail Pane

Presents a drawing surface that enables to create a new page or edit the contents of an existing page. It provides a visual environment to add HTML elements to the page, position them, and set properties that control their physical appearance.

Format Tool bar band

Contains buttons that are frequently used when formatting an HTML page in the designer. The items on this toolbar are available whenever a DIV element, or a hyperlink, or a piece of text, or the BODY or DOCUMENT objects in the treeview or detail panes of the designer is selected.

When a toolbar button is clicked to carry out the action represented by that button, Visual Basic inserts the appropriate opening and closing HTML tags or tag attributes to the selection. These tags are not visible within the designer, but the results of formatting are rendered in the detail pane.

Element Toolbar Band

Contains buttons that are frequently used when grouping the elements on the page such as by using DIV tag or working with tables. With the exception of the Table Operations icon, the options on this toolbar band are available only when some elements are selected. The Table Operations icon is always available.

<p>Note : The HTML files created using DHTML Page designer are stored in the files with the extension .dsr. The actual .htm files are created only when the application is compiled.</p>

Creating an DHTML Application using Visual Basic

This section shows how to create a simple DHTML application, which contains three HTML pages.

The first HTML page has a hyperlink that leads to the second HTML page. The second page contains a command whose clicking leads to the third page using code. This is not a full-fledged application. But the purpose of this section is to show how creation of the DHTML applications are simpler than by using HTML tags.

To create the DHTML application

- ❖ Choose File | New Project to open the New Project dialog box.
- ❖ Double-click the DHTML Application icon.



DHTML Application Icon

This will create a new DHTML Application with default DHTML page (DHTMLPage1).

- ❖ Choose Project | Add DHTML Page to add the second page.
- ❖ Repeat the above step to add the third page.
- ❖ Name the Project as Seas, DHTMLPage1 as ArabicSea, DHTMLPage2 as BayOfBengal, and DHTMLPage3 as IndianOcean using Properties Window.
- ❖ In the Project Explorer window double click the ArabicSea to bring its design window front.
- ❖ In the **Format band** select **Header1** from the **HTML Block Elements** combo box and enter "Arabic Sea" in the **Details pane**.
- ❖ Hit enter and type the following:

Click me to go to Bay of Bengal
- ❖ Select the word "me" and click the **Make Selection Into Link** icon in the **Element Toolbar band**.
- ❖ Right click on the word "me" in the popup menu that appears, choose Properties. This will display the Properties dialog box.

- ❖ Enter **BayOfBengal.htm** in the Link text box and **Leads to Bay of Bengal** in the popup text box as in Figure 5.5. This text will appear in the browser window when the mouse pointer moves over the hypertext “me”. When it is clicked 4BayOfBengal.htm page will appear in the browser.

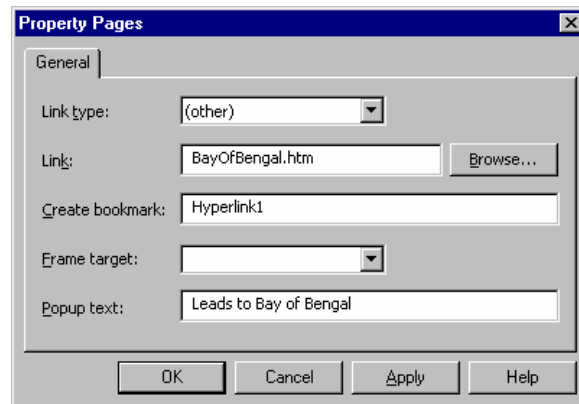


Figure 24.4 Destination (through Link) and Popup text for the hypertext “me”.

The designed DHTML page ArabicSea should resemble the Figure 24.5

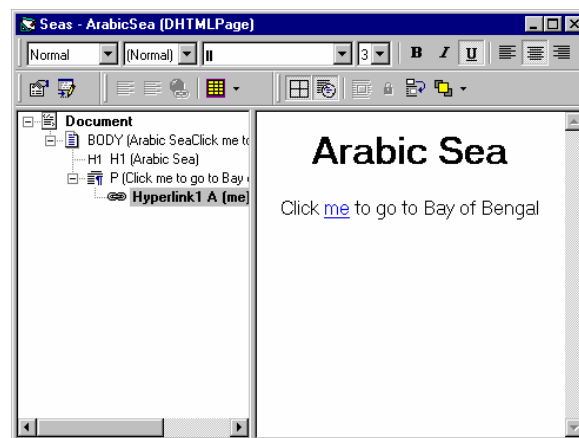


Figure 24.5 Designed DHTML Page ArabicSea.

- ❖ In the Project Explorer window double click the BayOfBengal to bring its design window front.
- ❖ In the **Format band** select **Header1** from the **HTML Block Elements** combo box and enter “Bay of Bengal” in the **Details pane**.
- ❖ Draw a button and set its caption as “Click me to go to Indian Ocean”

The designed DHTML page BayOfBengal should resemble the Figure 24.6.

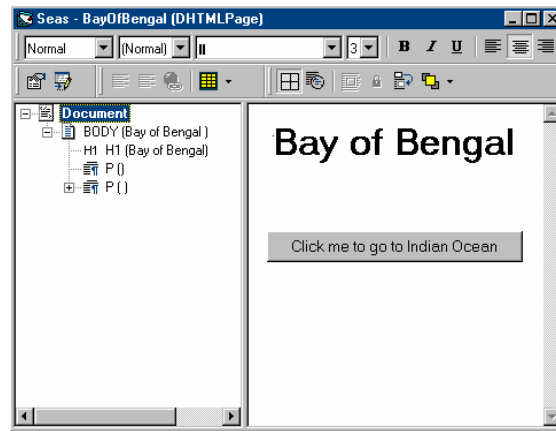


Figure 24.6 Designed DHTML Page BayOfBengal.

- ❖ Write the following lines for the button's (Click me to go...) click event procedure.

```
BaseWindow.location.href = "IndianOcean.htm"
```

The above line assigns the HTML file name "IndianOcean.htm" to the Href (Hyperlink reference) property of the location object of the basewindow to jump to that file.

- ❖ In the Project Explorer window double click the IndianOcean to bring its design window front.
- ❖ In the **Format band** select **Header1** from the **HTML Block Elements** combo box and enter "Indian Ocean" in the **Details pane**.
- ❖ Save the Project and Choose File | Make Seas.dll.
- ❖ When the Visual Basic prompts the .dll file name accept the default file name Seas.dll.
- ❖ When the Visual Basic prompts file name for each DHTML files, enter them as ArabicSea.htm, BayOfBengal.htm, and IndianOcean respectively.
- ❖ Open the Internet Explorer and in its address text box type ArabicSea.htm with full path and hit enter. This will open the ArabicSea.htm file in the Internet Explorer.
- ❖ Test the

- ❖ hyperlink and button to jump to their destinations.

24.6 Short Summary

- ☞ Visual Basic 6.0 includes new project type called DHTML Application, which provides visual environment to rapidly develop the DHTML applications.
- ☞ **Visual Environment** HTML pages can be created in the familiar Visual Basic IDE. As it is a Visual environment, controls can be placed in the HTML pages at the desired locations easily. Debugging tools can also be used to fix the bugs in VB scripts.
- ☞ *Code Security* When scripts are embedded with an HTML page, anyone can access the page, read the script, and make changes to it. As Visual Basic compiles the code into .dll, the code is not part of the HTML file itself, and so it can't be tampered.

24.8 Brain Storm

1. What are all the process involved in developing a DHTML script?
2. What are the advantages of DHTML?

Lab Unit - 24 (2 Real Time Hrs)

Lecture - 25

Data Report

Objectives

In this lecture you will learn the following

- ❖ About Data Report
- ❖ About Report Designer
- ❖ Creating and Printing a Data Report

Lecture Unit - 25

- 25.1 Snap Shot
- 25.2 Report
- 25.3 Introduction to Data Report
- 25.4 Parts of a Data Report
- 25.5 Report Designer
- 25.6 Data Report Controls
- 25.7 Data Report Creation
- 25.8 Short Summary
- 25.9 Brain Storm

Lab unit 25 (2 Real Time Hrs)

25.1 Snap Shot

This Chapter goes on discussion about Data Report, Data report designer, Data Report control and creating a simple Data Report Creation.

25.2 Report

Displays or prints a report under the control of a report definition file created with MODIFY REPORT or CREATE REPORT.

Syntax

```
REPORT FORM FileName1 | ?  
[ENVIRONMENT]  
[Scope] [FOR IExpression1] [WHILE IExpression2]  
[HEADING cHeadingText]  
[NOCONSOLE]  
[NOOPTIMIZE]  
[PLAIN]  
[RANGE nStartPage [, nEndPage]]  
[PREVIEW [[IN] WINDOW WindowName | IN SCREEN]  
[NOWAIT]]  
[TO PRINTER [PROMPT] | TO FILE FileName2 [ASCII]]  
[NAME ObjectName]  
[SUMMARY]
```

Arguments

FileName1

Specifies the name of the report definition file to print.

?

Displays the Open dialog box, from which you can choose a report file.

25.3 Introduction to Data Report

Helps you design a report that displays fields and records from the underlying table or query.

A report is an effective way to present data in a printed format. You can display the information the way you want to see it.

You create the report using graphical objects called Data Report designer controls. Data Report designer controls include: a data-bound TextBox control, a Function control which displays calculated figures, an Image control for inserting graphics, labels that display captions, and a Line and a Shape control that graphically organizes the data.

Note : Although the Data Report designer controls are similar to Visual Basic intrinsic controls, Data Report Designer controls have a limited subset of features. When the Data Report designer is added to a project, the designer's controls are placed in the Visual Basic Toolbox on a new tab named DataReport and can be used only in the Microsoft Data Report.

Creating reports is a main function of any good business application. A system might have useful data, but without a coherent way to present it, the numbers are meaningless. So, Microsoft provides a Data Report designer, which is used to generate eye-catching reports. It is used in conjunction with a data source such as the Data Environment designer.

The Data Report generates reports using records from a database. To use it:

1. Configure a data source, such as the Microsoft Data Environment, to access a database.
2. Set the DataSource property of the DataReport object to the data source.
3. Set the DataMember property of the DataReport object to a data member.
4. Right-click the designer and click Retrieve Structure.
5. Add appropriate controls to the appropriate sections.
6. Set the DataMember and DataField properties for each control.
7. At run time, use the Show method to display the Data Report.

Use the DataReport object to programmatically change the appearance and behavior of the Data Report by changing the layout of each Section object.

The Data Report designer also features the ability to export reports using the ExportReport method. This method allows you to specify an ExportFormat object, from the ExportFormats collection, to use as a template for the report.

25.4 Parts of The Data Report

The Data Report designer consists of the following objects:

DataReport object—Similar to a Visual Basic form, the DataReport object has a visual designer. It is used to create the layout of a report.

Section object— The data report designer contains sections which helps to configure the report more elegantly. The Sections of the Data report designer is explained in the next section.

Data Report Controls—Special controls that only work on the Data Report designer are included with it. These controls are found in the Visual Basic Toolbox, but they are placed on a separate tab named "DataReport."

25.5 Report Designer

Use the Report Designer to create and modify reports. When the Report Designer window is active, Visual Basic displays Report Controls toolbar.

Data Report Designer Features

The Data Report designer has several features:

Drag-and-Drop Functionality for Fields— When fields are dragged from the Microsoft Data Environment designer to the Data Report designer, Visual Basic automatically creates a text box control on the data report and sets the DataMember and DataField properties of the dropped field. A command object can also be dragged from the Data Environment designer to the Data Report designer. In that case, for each of the fields contained by the Command object, a text box control will be created on the data report; the DataMember and DataField property for each text box will be set to the appropriate values.

Toolbox Controls—The Data Report designer features its own set of controls. When a Data Report designer is added to a project, the controls are automatically created on a new Toolbox tab named DataReport. Most of the controls are functionally identical to Visual Basic intrinsic controls, and include a Label, Shape, Image, TextBox, and Line control. The sixth control, the Function control, automatically generates one of four kinds of information: Sum, Average, Minimum, or Maximum.

Print Reports— When a report is generated, it can be printed by clicking the printer icon on the toolbar.

Note: A printer must be installed on the computer to print a report.

File Export - In addition to creating printable reports, it can be exported to text files, by clicking the Export icon and specifying text file name.

Report Designer Shortcut Menu

Contains shortcuts to several commands useful in the Report Designer. These commands appear when you right-click the Report Designer.

Note These shortcut menu options are context sensitive. Not all of the following options appear each time you open the shortcut menu.

Shortcut Menu Options

Paste

Places from the Clipboard cut or copied sections of text at the insertion point. This command corresponds to the Paste command in the Edit menu.

Cut

Removes selected text to the Clipboard. Once you cut something, you can paste it elsewhere in the current application or in other application. This command corresponds to the Cut command in the Edit menu.

Copy

Duplicates selected text and puts it on the Clipboard. Once you copy something to the Clipboard, you can paste it elsewhere in the current application or in other applications. This command corresponds to the Copy command in the Edit menu.

Preview

Shows you the results of your work without printing it and displays the Print Preview toolbar. Use the options on the toolbar to change the preview. You can also click the image of the previewed page to zoom in, and click again to zoom out or return to design mode. This command corresponds to the Preview command in the View menu.

Print

Allows you to print the current report by displaying the Print dialog box. This command corresponds to the Print command in the File menu.

Data Environment

Displays the Data Environment Designer which allows you to visually create and modify the data environment of the report. This command corresponds to the Data Environment command in the View menu.

Data Grouping

Displays the Data Grouping dialog box so you can create data groups and specify their properties. This command corresponds to the Data Grouping command in the Report menu.

Properties

Displays a dialog box that corresponds to the object type selected (a text field, rectangle, an OLE Object field, etc.) which allows you to set the object's position and other parameters. These parameters can include printing instructions, calculations and expressions.

Help

Displays Help on the selected item.

Sections of the Data Report Designer

The Data Report designer contains the following Sections:

Report Header – contains the text that appears at the very beginning of a report, such as the report title, author, or database name.

Page Header – contains information that goes at the top of every page, such as the report's title.

Group Header/Footer – contains a "repeating" section of the data report. Each group header is matched with a group footer. The header and footer pair are associated with a single Command object in the Data Environment designer.

Details – contains the innermost "repeating" part (the records) of the report. The details section is associated with the lowest-level Command object in a Data Environment hierarchy.

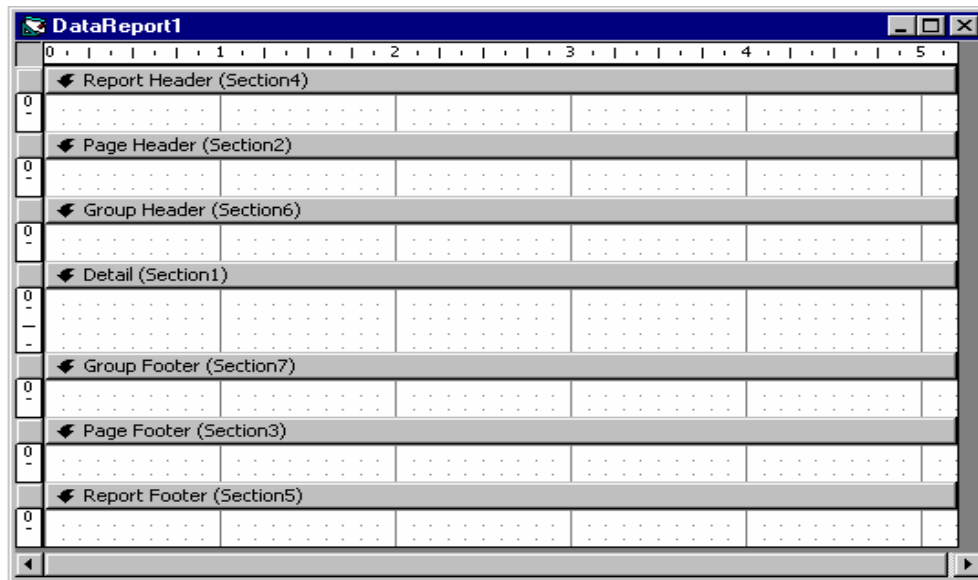


Figure 25.1 Data Report.

Page Footer – contains the information that goes at the bottom of every page, such as the page number.

Report Footer – contains the text that appears at the very end of the report, such as summary information, or an address or contact name.

25.6 Data Report Controls

When a new Data Report designer is added to a project, the following controls are automatically placed in the Toolbox tab named DataReport:

- ❖ **TextBox Control** (RptTextBox) – holds the data that is supplied at runtime.
- ❖ **Label Control** (RptLabel) – used to identify fields or sections.

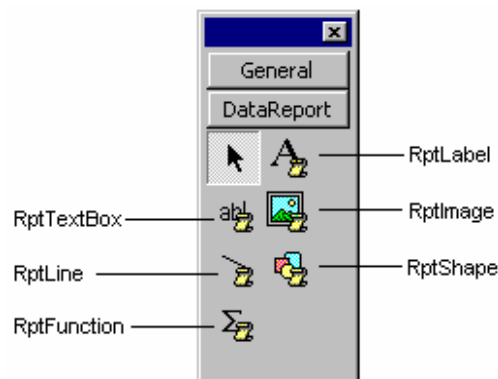


Figure 25.2 Data controls for Data Report on the Toolbox.

- ❖ **Image Control** (RptImage) – enables to place graphics on a report.

Note : This control cannot be bound to a data field.

- ❖ **Line Control** (RptLine) – lets to draw rules on the report to further distinguish sections.
- ❖ **Shape Control** (RptShape) – enables to place rectangles, triangles, or circles (and ovals) on a report.
- ❖ **Function Control** (RptFunction) – a special text box that calculates values as the report is generated.

25.7 Creating a Simple Data Report

This topic creates a simple data report using a Data Environment designer as a data source. The Data Environment designer uses the NorthWind database supplied with Visual Basic to create a simple hierarchical cursor. The cursor contains two tables, Customers and Orders, and uses the CustomerID field to link the two. The finished report resembles the figure 25.3 .

Figure 25.3 Simple Data Report: Order Dates by Customers

Before you begin the step-by-step process, ensure that the Northwind database (Nwind.mdb) is present on your computer. If it is not present, copy the file from your Visual Basic CD onto your hard disk.

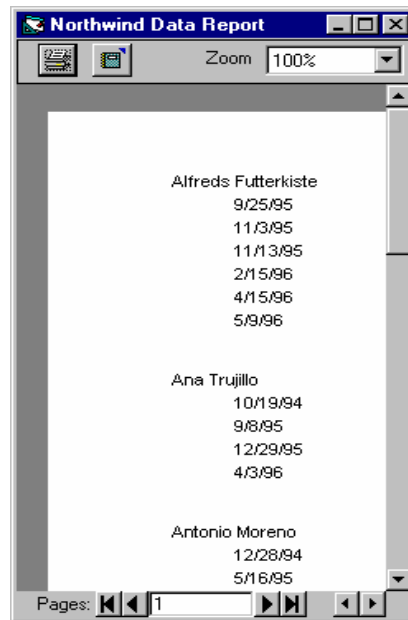
To Create a simple hierarchical cursor in the Data Environment designer

1. Create a new Standard EXE project.
2. On the Project menu, click Add Data Environment to add a designer to your project. If the designer is not listed on the Project menu, click Components. Click the Designers tab, and click Data Environment to add the designer to the menu.
3. On the Data Link Properties dialog box, click Microsoft Jet 3.51 OLE DB Provider. This selects the correct OLE DB provider for accessing a Jet database.
4. Click the Next button to get to the Connection tab.
5. Click the ellipsis button (...) next to the first text box.
6. Use the Select Access Database dialog box to navigate to the nwind.mdb file, which is installed in the Program Files\Microsoft Visual Studio\Vb98 directory.
7. Click OK to close the dialog box.
8. Right-click the Connection1 icon, and click Rename. Change the name of the icon to Northwind.
9. Right-click the Northwind icon, and then click Add Command to display the Command1 dialog box. In the dialog box, set the properties as shown below:

Property	Setting
Command Name	Customers
Connection	Northwind
DataBase Object	Table
Object Name	Customers

10. Click OK to close the dialog box.

11. Right-click the Customers command, and click Add Child Command to display the Command2 dialog box. In the dialog box, set the properties as shown below:



Property	Setting
Command Name	Orders
Connection	Northwind
DataBase Object	Table
Object Name	Orders

12. Click the Relation tab. The Relate to a Parent Command Object check box should be checked. The Parent box should contain Customers; both the Parent Fields and Child Fields/Parameters boxes should contain CustomerID.

When designing relational databases, it's customary for related tables to use the same name for linking fields. In this case, the linking fields are both named CustomerID. The Data Environment designer automatically matches such pairs in the dialog box.

13. Click Add. Click OK to close the dialog box.

Clicking the Add button adds the relation to the Command object. After closing the dialog box, the Data Environment designer reflects the relationship by displaying the two commands as a hierarchy. This hierarchy will be used to create the data report.

14. Set the properties of the project and designer according to the settings below, then save the project:

Object	Property	Setting
Project	Name	PrjNwind
DataEnvironment	Name	deNwind
Form	Name	frmShowReport

Data Report at Run Time

1. On the Project Explorer window, double-click the frmShowReport icon to display the Form designer.
2. On Toolbox, click the General tab.
When you add a Data Report designer to your project, its controls are added to the tab named DataReport. To use the standard Visual Basic controls, you must switch to the General tab.
3. Click the CommandButton icon and draw a CommandButton on the form.
4. Set the properties of the Command1 control according to the table below:

Property	Setting
Name	cmd
Show	Caption
Show	Report

5. In the button's Click event, paste the code below.

```
Private Sub cmdShow_Click()  
    rptNwind.Show  
End Sub
```

6. Save and run the project.
7. Click Show Report to display the report in print preview mode.

Optional—Setting the Data Report as the Startup Object

You can also display the data report with no code at all.

1. On the Project menu, click prjNwind Properties.

2. In the Startup Object box, select rptNwind.
3. Save and run the project.

Note : If you use this method, you can remove the Form object from your project.

Printing a Data Report

Printing a data report can be accomplished in one of two ways. The user can click the Print button that appears on the data report in Print Preview mode (using the Show method), or you can programmatically enable printing using the PrintReport method. If an error occurs during printing, trap it in the Error event.

Choosing to Display a Print Dialog Box

When printing a report programmatically, you have two choices: to print by displaying the Print dialog box, or by printing without displaying the dialog box.

To display the Print dialog box

1. Add a CommandButton to a Form.
2. In the button's Click event, place the following code:

```
DataReport1.PrintReport True
```

The Print dialog box allows the user to select a printer, print to file, select a range of pages to print, and specify the number of copies to print.

Note : Printers must be installed on the computer in order to present a choice of printers.

Printing Without a Dialog Box

In some cases, you may wish to print the report without user intervention. The PrintReport method also gives you the option of selecting a range of pages to print, either all, or a specified range.

To print without displaying the dialog box

1. Add a CommandButton to a Form.

2. In the button's Click event, place the following code:

```
DataReport1.PrintReport False
```

Or, to specify a range of pages to print, use the code below:

```
0DataReport1.PrintReport False, rptRangeFromTo, 1, 2
```

PrintReport Method

At run time, prints the data report created with the Data Report designer.

Syntax

```
object.PrintReport(ShowDialog, Range, PageFrom, PageTo)
```

The PrintReport method syntax has these parts:

PartDescriptionobjectRequired. An object expression that evaluates to an object in the Applies To

list.ShowDialogOptional. A boolean expression that determines if the Print dialog box is shown.RangeOptional. Sets an integer that determines if all the pages in the report will be executed or a range of pages, as shown in Settings.PageFromOptional. An integer that sets the page from which to start printing.PageToOptional. An integer that sets the page at which to stop printing.

Settings

ConstantValueDescriptionrpt RangeAllPages0(Default)

All pages will be printed. rptRangeFromTo1Only the specified range of pages will be exported.

Long

If no arguments are supplied with the method, a dialog box will be displayed prompting the user for appropriate information.

The PrintReport method performs an asynchronous operation. The PrintReport method returns the identifier of the "cookie" that identifies the asynchronous operation.

To extend the data environment

1. On the Data Environment designer, right-click the Orders Command object. Then click Add Child Command.

2. On the Command1 Properties dialog box, set the following properties:

Property	Setting
Command Name	OrderDetails
Connection	Northwind
DataBase Object	Table
Object Name	Order Details

3. Click the Relation tab. The Relate to a Parent Command Object check box should be checked. The Parent box should contain Orders; both the Parent Fields and Child Fields/Parameters boxes should contain OrderID. Click the Add button and then click OK to close the dialog box.
4. Right-click the OrderDetails Command object, and click Add Child Command. Set the properties of the connection as shown below:

Property	Setting
Command Name	Products
Connection	Northwind
DataBase	Object Table
Object Name	Products.

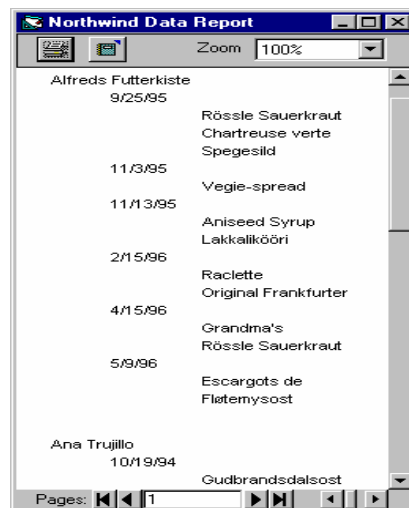


Figure 25.4 Data Report

Click the Relation tab. The Relate to a Parent Command Object check box should be checked. The Parent box should contain OrderDetails; both the Parent Fields and

Child Fields/Parameters boxes should contain ProductID. Click the Add button and then click OK to close the dialog box.

Extending the Data Report

Once the data environment has been extended with new tables, you can extend the data report as well by dragging fields from the Data Environment designer to the Data Report designer.

To Extend the data report

1. Right-click the Data Report designer, and clear Show Page Header/Footer box.

Clearing this option deletes the page header and footer, which are not being used at this point.

2. Right-click the Data Report designer, and click Insert Group Header/Footer. The Insert New Group Header/Footer dialog box will be displayed.

The dialog box allows you to determine if the new header and footer will "bracket" other header/footer pairs. This becomes important as you add more header and footer pairs because the outermost pair of header/footers subordinates all other pairs. Click OK to select the default placement of the new header and footer pair and close the dialog box.

3. Select the new group header, and on the Properties window, change its name from Section1 to Orders_Header. Change the corresponding footer name from Section4 to Orders_Footer.
4. Repeat steps 2 to 3. Name the new group header OrderDetails_Header, and the new group footer OrderDetails_Footer.
5. Click the Detail (Orders_Detail) section to select it. On the Properties window, change the section's name to Products_Detail.
6. Using the mouse, drag the OrderDate field from the Detail (Products_Detail) section to the Orders_Header section.
7. From the Data Environment designer, drag the ProductName field (under the Products command) into the Detail (Products_Detail) section.
8. Delete the Label control named Label1.

9. Resize the group headers, and rearrange the text box controls to resemble the figure 25.5 below.

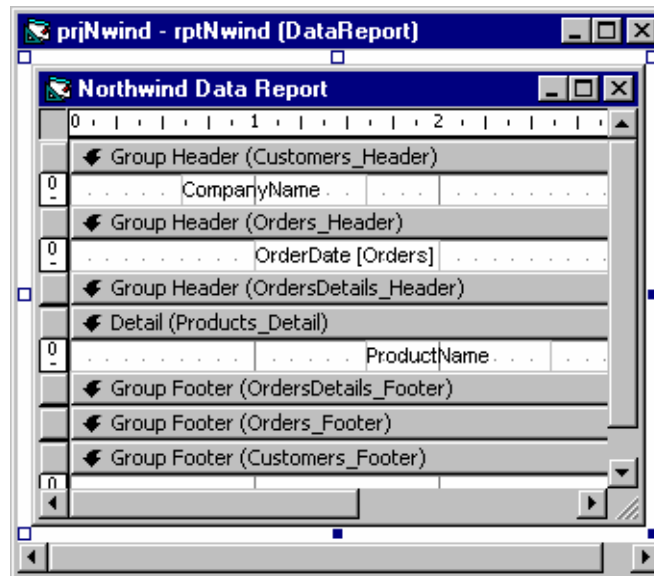


Figure 25.5 Data Report Designer

The figure above requires some explanation. First, the Group footers are all closed in order to take up the least possible space. Like the Details section, any additional space left in any header or footer will be multiplied in the final report. Therefore, if a header or footer doesn't contain any fields, you can close the distance between the headers or footers.

The Group Header named OrderDetails_Header is also closed. If you wonder why no fields are being shown, you must understand that the Order Details table in the Northwind database is a join table – the table contains only the IDs of records from the Orders table joined to IDs of records from the Products table. Thus the Order Details table doesn't contain fields which are actually displayed. Instead, it functions only to join two other tables. In the Data Report designer, the Order Details table therefore functions only to create groups of records – the product names grouped under the order dates.

Finally, the Details section contains only the names of products. The Details section contains the innermost level of repeating records.

10. Save and run the project.

25.8 Short Summary

- The first four kinds of ActiveX designers loaded for a project are listed on the project menu. If more than four designers are loaded, the later ones will be available from the More ActiveX submenu of the Project menu.
- Visual Basic's intrinsic controls, or any ActiveX controls, can not be used on the Data Report Designer.

25.9 Brain Storm

1. What are the parts of Data Report Objects?
2. What are the special features of Data Report Designer?
3. What are all the Options available in Data Report Shortcut Menu?

Lab Unit - 25 (2 Real Time Hrs)

Create the following tables

- **EMP**

Eno
Ename
Basic
HRA
DA
PF
Dno

- **DEPT**

Dno
Dname

Add records to both the tables

Open Data Report

Fetch the tables to the Command of Data Environment

Create a report, which shows all the above listed fields grouped by their **Dno**

Calculate the Net Salary of individuals and display it

Calculate Department-wise Total, Grant Total and display it

Display the page number at the right bottom of the page

Display the title of the report as '**XYZ COMPANY**' and a Logo to the left of it

The Date of preparation should be displayed in the top right corner of the report

After completion of your report, Place a Command Button on to your Form.

When Command Button is clicked, the Report should be displayed.

Active Server Page

Objectives

In this lecture you will learn the following

- ❖ About New Features of Visual Basic6
- ❖ New Object Oriented features
- ❖ Under About Integrated Development Environment
- ❖ How to develop a script in Visual Basic

1

Lecture Unit - 26

- 26.1 Snap Shot
- 26.2 Active Server Pages
- 26.3 Advantages of Active Server Pages
- 26.4 Active Server Pages - Objects
- 26.5 Creating Active Server Pages
- 26.6 Short Summary
- 26.7 Brain Storm

26.Snap Shot

This chapter will discuss on how to display Data Report during the Run Time, Introduction about Active Server Page, Developing Active Server Page and its applications.

26.2 Active Server Pages

Microsoft introduced several methods for developing scripts on the server - some of them quite simple and others not as simple. There was a confusing time when companies did their best to simplify the development for sever side scripts, but none of these methods were particularly easy for Visual Basic developers or even for Web authors.

The situation changed drastically with the introduction of Active Server Pages (ASP), an elegant solution to the problem of scripting. Active Server Pages are basically HTML code that contains VBScript code, which is executed on the server. The HTML code is transmitted to the client, as is. Active Server Pages almost look like HTML document. As the HTML document has the .htm extension, ASP has the extension .asp.

The scripts included on an Active Server Page produce text and HTML tags that are also sent to the client, where they are rendered on the screen. An Active Server Page can produce any output, but only HTML documents can be rendered on the client.

26.3 Advantages of Active Server Pages

Open Architecture

Active Server Pages are not limited to a particular language. Currently, they support VBScript and JScript, but third parties can provide support for other languages such as Perl. In addition, multiple scripting languages can be used interchangeably in the same ASP file.

Ease of Development

Active Server Pages make it easy for HTML authors to activate their Web pages on the server.

Browser Independence

The output produced by an ASP is straight HTML code, which can be viewed with any browser. All application logic needed to generate dynamic content resides on

the server. It may be complicated. But the visitors do not see this complexity, and they aren't required to have a specific browser. Any browser that can handle the latest version of HTML will render the documents produced by the ASP.

Passing Parameters to Server

Active Server Pages provide the client's the requested information. Actually the request is made by the client's browser. It passes visitor-supplied data to the Server. In the Server, the ASP read the data, process them, and pass the results in the form of another HTML page, which is generated on the fly.

In a normal HTML page, when a hyperlink is clicked or some button that lets a jump to another page is clicked, the browser passes the address of the page (which is to be displayed) to the Server and the Server supplies that page to the browser.

But, for the client to interact with the Server there must be a way for the browser to send information to the server, along with the name of the requested page or document. For example, a visitor may like to get information about specific products, say Car. The browser must request the document, say Products.htm, with the user entered data - Car. These data are appended as parameters with the requested document and sent to the server, which is something like this.

`http://www.someserver.com/products.asp?ProductName=Car`

everything proceeding the question mark is the address of the document product.asp. This document is not an HTML page, but the client doesn't care; it will receive an HTML document anyway. The question mark separates the address of the requested document from the parameters. Parameters have a name and a value, and they are specified as,

`parametername=parametervalue`

The parametername parameter is the name, and parametervalue is the value of the parameter. If a Form (defined using <FORM> tag) is used collect information from the visitor, the parameter names are the names of the controls on the Form. Multiple parameters are separated by the ampersand (&) sign, as in the following:

`http://www.someserver.com/products.asp?ProductName=Car&Version=Diesel`

If a value contains spaces, each space is replaced by the plus sign (+) as in the following:

http://www.someserver.com/products.asp?ProductName=Deluxe+Car&Version=Diesel

To append the parameter to an address the <FORM> tag requires two parameters:

- ❖ Action
- ❖ Method

The Action parameter specifies the name of the application that will process the request on the Server. For the purpose of this chapter, this application is an Active Server Page (a file with the extension ASP). The method parameter indicates the type of form-handling protocol that will be used to pass the parameters to the application, and it can have one of the values GET and POST. The application that receives the information usually dictates the method. Active Server Pages recognize both methods, but handle them differently.

The browser, therefore, picks up the address of the application from the <FORM> tag and appends the parameter=value pair to it. Each pair corresponds to a control on the Form: The control's name is the parameter name, and the control's value is the parameter value. The whole thing is done automatically. When the Submit button in the FORM is clicked, the browser submits the request to the Server. This button is defined using <INPUT> tag as in the following:

```
<INPUT Type=Submit Value="Submit">
```

26.4 Objects of Active Server Pages

Previous section explained how the browser sends information to the Server. Active Server Pages, which are, get executed on the Server must have some way to process the information sent by the browser. They use objects to process the incoming information and to give response to the browser. Their major objects are:

Request

It enables to access the parameters passed to the Server by the client, along with the address. Its property **QueryString** returns the entire parameters collection. This collection can be stored in a variable and parameter names can be used to access their values as in the following:

```
' Storing the parameter collection in a variable (Param)
Set Param = Request.QueryString
' Accessing the ProductName parameter
PName = Param("ProductName")
```

Response

It provides methods needed to build the response, which is another HTML document. The Response object represents the output stream, which is directed by the Web server to the client, as if it were another HTML document. The following example shows an ASP builds an HTML document based on the value of the PName variable, in which parameter ProductName's value is stored.

```
<%
Response.Write "<BODY>"
If PName = "Car"
    Response.Write "<P> <B> New Cars </B> </P>"
    Response.Write "Indica <BR>"
    Response.Write "Hyundai <BR>"
Else
    Response.Write "<P> <B> General Products </B> </P>"
    -----
    -----
End if
Response.Write "</BODY>"
%>
```

26.5 Creating an ASP Page

This section first gives a listing of an HTML page, which gets a word from the visitor and calls an ASP page with the entered word as an argument. Then this section shows how to create the ASP page, which gets the entered word by the visitor, generates an HTML page on the fly that prints that word for five times and sends that HTML page to the browser. The following is the listing of the HTML page SendWord.htm.

```
<HTML>
<BODY>
Type a word and click Submit.<BR>
Web Server will print it for five times

<FORM Method="GET"
Action="http://127.0.0.1/ASPages/PrintWord.ASP">

<INPUT Type=Text Name="Word"> <BR><BR>
<INPUT Type=Submit Value="Submit">

</FORM>
</BODY>
```

```
</HTML>
```

The string `http://127.0.0.1` in the Action parameter of the FORM tag specifies the address of the Web Server of the local computer. The string `/ASPAGES` specifies the virtual directory which contains the `PrintWord.ASP` file that is to be executed. Physical directories in the Web Server are mapped into Virtual directories and the Web Server makes use of it. Using virtual directories will prevent the visitors from knowing about the actual location of the Active Server Pages.

The FORM in this HTML page contains two controls - a textfield and a submit button. The Name argument in the INPUT tag that defines the textfield specifies the name of the textfield as "Word". So that when submit button is clicked the parameter=value pair in the address string will appear as in the following:

```
Word=someword
```

The value argument in the INPUT tag that defines the Submit button specifies the caption of the Submit button as "Submit".

The following steps show how to create the ASP file.

- ❖ Open the Notepad.
- ❖ Type the lines given below:

```
<HTML>  
<%  
Set Param = Request.QueryString  
Response.Write "<BODY>"  
Response.Write "<H2> Response from ASP</H2> <HR>"  
For I = 1 to 5  
    Response.Write "<H4>" & Param("Word") & "</H4>"  
Next  
Response.Write "</BODY>"  
>  
</HTML>
```

- ❖ Save it as `PrintWord.asp`

The scripts in this document are placed between the tags `<%` and `>`. So they are not visible to the user and they are replaced by their output (Proved in a Figure shortly). This document first assigns the parameter collection sent by the browser in the variable **Param**. Then it begins to build the HTML document that prints the value of the "Word" parameter for five times using the write method of the Response object. The result of the write method produces an output stream and it is directed by the Server to the browser, and the browser renders the received output.

Open the SendWord.htm in the Internet Explorer, which will resemble the Figure 26.1

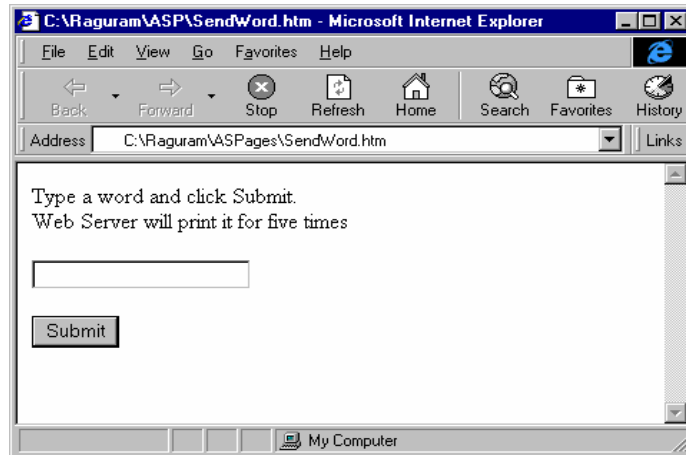


Figure 26.1 SendWord.htm page in Internet Explorer.

- ❖ Type a word in the textfield, say Jaihind, and click the Sumbit button. The browser will send the request to the Server as in the following:

`http://127.0.0.1/ASPAGES/PrintWord.ASP?Word=Jaihind`

When the Server receives the request it will invoke the PrintWord.ASP file with the received parameter "Word". The output produced by the PrintWord.ASP is directed by the Server to the Browser, which renders it as in Figure 26.2

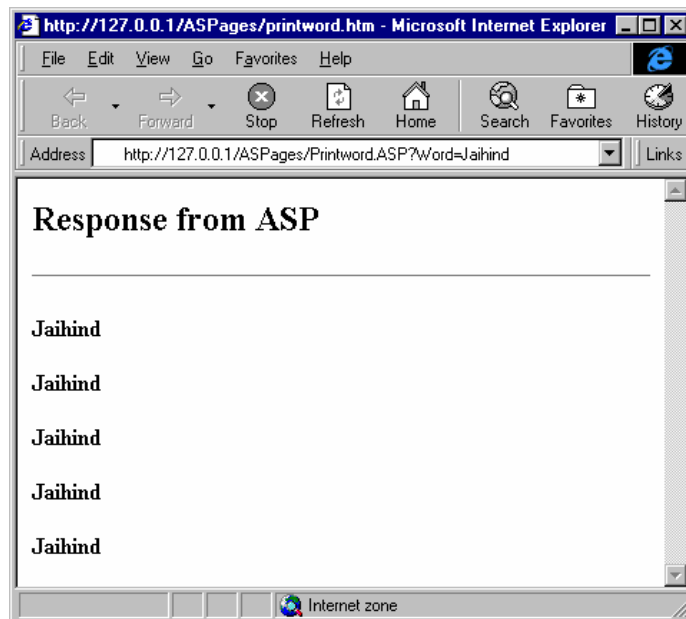


Figure 26.2 Rendered HTML page, which is created by the PrintWord.ASP.

- ❖ Choose View | Source from the Internet Explorer menu. This will display the created HTML document code in the Notepad as in Figure 26.3.

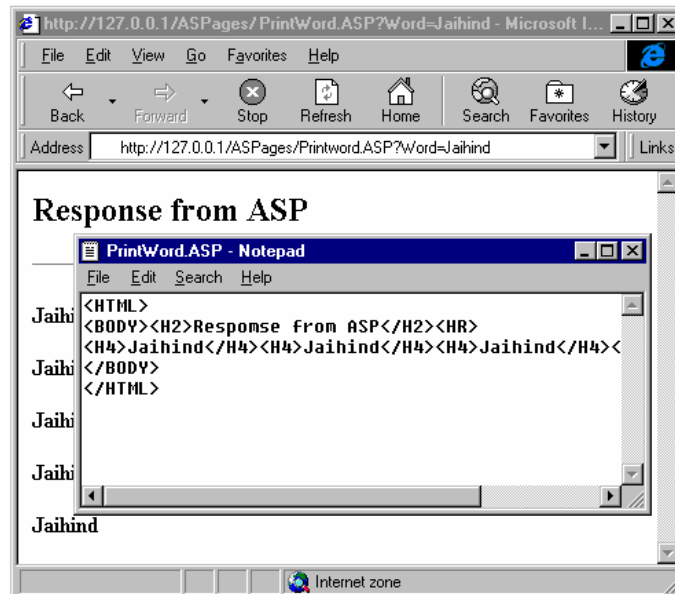


Figure 26.3 Created HTML code by PrintWord.ASP's script in the Notepad.

Note that the scripts in the PrintWord.ASP are replaced by its output.

26.6 Short Summary

- ☞ To call Active Server Pages that live in the local computer, either Internet Information or Personal Web Server must be installed in the local computer.
- ☞ Server side scripts such as Response.Write "...", are enclosed in a pair of <% and %>. These tags enclose all the statements that must be executed on the server. Everything Between these two tags is considered server-side scripts, which is replaced by its output and Seen by the client.
- ☞ Active Server Pages provide the client's the requested information.

26.7 Brain Storm

1. What are all the steps involved do display a Data Report during the Run Time?
2. Write a note on Active Server Pages
3. What are the advantages of Active Server Pages?
4. What are all the ASP Object type?

Lecture - 27

Package and Deployment

Objectives

In this lecture you will learn the following

- ❖ Introduction to Package and Deployment
- ❖ Uses Of Packages and Deployment
- ❖ Working with Packages and Deployment

Lecture Unit - 27

- 27.1 Snap Shot
- 27.2 Package and Deployment Wizard
- 27.3 Overall steps in the Packaging process
- 27.4 Overall steps in the Deployment Process
- 27.5 Short Summary
- 27.6 Brain Storm

27.1 Snap Shot

This Chapter mainly will discuss on about Package and Deployment wizard.

27.2 Package and Deployment Wizard

The package and Deployment Wizard is one of the application in the VB6 program group. You can also use it from within Visual Basic by selecting it in the Add -In Manager.

The Visual Basic Package and Deployment Wizard helps you create .cab files for your application, group them into a unit, or package, that contains all information needed for installation, and deliver those packages to end users. You can use Visual Basic's Package and Deployment Wizard to create packages that are distributed on floppy disks, CDs, a local or network drive, or the Web. The Package and Deployment Wizard automates much of the work involved in creating and deploying these files.

The distribution process of an completed Visual Basic application has the following two steps:

Packaging – The application's files must be packaged into one or more .cab files to deploy them in a desired location. A *.cab file* is a compressed file that is well suited to distribution on either disks or the Internet. A setup program must also be created to install the packaged files in the end user's system.

Deployment – The packaged application must be moved to the location, from which users can install it. This means copying the package to floppy disks or to a local or network drive, or deploying the package to a Web site.

The Package and Deployment wizard

The Visual Basic Package and Deployment Wizard helps automates much of the work involved in creating and deploying the .cab files and setup files.

The Package and Deployment Wizard offers three options:

- ❖ The Package option helps to package a project's files into a .cab file that can then be deployed, and in some cases creates a setup program that installs the .cab files. The wizard determines the files needed to package and leads through all the choices that must be made in order to create one or more .cab files for the application.
- ❖ The Deploy option helps to deliver the packaged applications to the appropriate distribution media, such as floppies, a network share, or a Web site.

- ❖ The Manage Scripts option lets to view and manipulated the scripts that are saved from previous packaging and deployment sessions in the wizard. Each time the wizard is used a script is saved containing all the choices made in the current session. These scripts are useful to speed up the deployment process of the same application for its later versions.

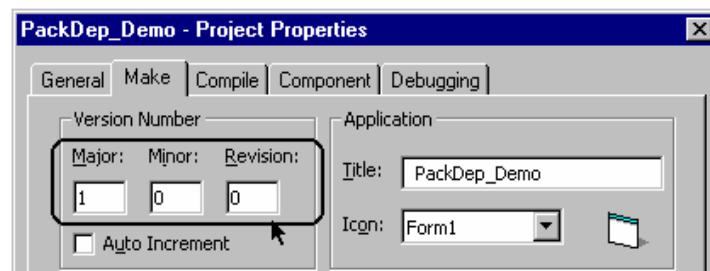
Each of these options is explained in detail in the following sections.

Application Packaging with the Wizard

Application packaging is the act of creating a package that can install an application onto the user's computer. A *package* consists of the .cab file or files that contain the compressed project files and any other necessary files the user needs to install and run the application. These files may include setup programs, secondary .cab files, or other needed files. The additional files vary based on the type of package that is created. There are two kinds of packages:

- ❖ Standard packages – Used to distribute the application on disk, floppy, or via a network share.
- ❖ Internet packages – Used to distribute via an Intranet or Internet site

An important point in creating a package is that, the version number of an application must be set on the make tab (Figure 27.1) of the Project Properties dialog box. It plays vital role when the new version of the existing application is distributed. Without the appropriate change in version numbers, the end user's computer may determine that critical files do not



need to be updated.

Figure 27.1 Portion of Make tab of Project Properties dialog box to set Major and Minor release numbers, and Revision number.

27.3 Overall Steps in the Packaging Process

The following steps are common for any package type. And the Package and Deployment Wizard performs many of these steps automatically.

- 1. Determining the type of package** Either standard package can be chosen for Windows-based programs that will be distributed on disk, on CD, or over a network; or an Internet package for programs that will be distributed (ActiveX programs) on the Web. Creation of only a dependency file can also be chosen.
- 2. Determining the files needed for distribution** The wizard must determine the project files and dependent files for the application before it can create the package. Project files are the files included in the project itself – for example, the .vbp file and its contents. Dependent files are run-time files or components the application requires to run. Dependency information is stored in the vb6dep.ini file, or in various .dep files corresponding to the components in the project.
- 3. Determining where to install files on the user's machine** Program and setup files are usually installed into a subdirectory of the Program Files directory, while system and dependent files are usually installed into the \Windows\System or \Winnt\System32 directory. The setup program must take this into account and determine where to install each file.
- 4. Creating the package** The wizard creates the package and the setup program (setup1.exe) for it, referencing all necessary files. The end result of this step is one or more .cab files and any necessary setup files.

Application Deployment with the Wizard

Application deployment is the act of moving a packaged application to either the distribution media such as CD-ROM or to a Web site from which it can be downloaded. There are two ways to deploy Visual Basic applications:

- ❖ Using Deployment portion of the Package and Deployment Wizard to deploy the applications to floppy disks, a local or network drive, or to a web site.
- ❖ Manually copying files to disks or manually publishing the files to the appropriate Web location.

The Package and Deployment Wizard provides shortcuts and automatically performs some of the same tasks that is performed manually.

27.4 Overall Steps in the Deployment Process

The following steps are common in both ways (by hand -manual and by Wizard) of deploying an application.

- 1. Identifying the package to deploy** The active project might have been packaged using different package types such as Standard package, Internet Package, etc. This step identifies a package for deployment.
- 2. Choosing a deployment method** This step chooses a method to deploy the application to the Internet, to floppy disks, or to a directory on a local or network drive.
- 3. Choosing the files to deploy** If the application is deployed to the Internet, some files may be added or removed from the list of files to be deployed. For example very common .ocx may be available in the end users system. These files can be removed from the list of files to be deployed. This will reduce the download time of the end user's system.
- 4. Determining the destination for deployed files** For Internet deployment, this involves specifying a Web site to which the package is to be deployed. For directory deployment, this means indicating the drive location to which the package is to be deployed. For floppy disk deployment, this means choosing the appropriate floppy drive.

Packaging and Deploying The Packdep_Demo Application

This section shows how to Package and deploy the simple Visual Basic application PackDep_Demo.

- ❖ Start a new project (Standard EXE) and rename the project as PackDep_Demo.
- ❖ Place a command button on Form1 and write the following code in its click event.
Msgbox "India Jindabadh"
- ❖ Save the Form1 and PackDep_Demo project in a separate folder say PDDemo.

Packaging the PackDep_Demo application

This section shows how to package the PackDep_Demo application in the sub folder Setup of the folder PDDemo.

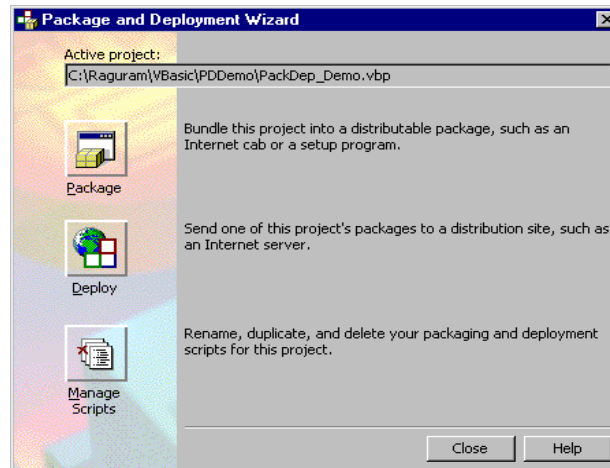
To Package the PackDep_Demo application

- ❖ Choose AddIns | Package and Deployment Wizard.

<p>Note : If Package and Deployment Wizard doesn't appear in the Add-Ins menu, choose Add-Ins Add-Ins Manager, select Package and</p>
--

Deployment Wizard, check Loaded/Unloaded option and click OK to add the Package and Deployment Wizard add-in to the Add-Ins menu.

This will display Package and Deployment Wizard as in Figure 27.2. Figure 27.2 Package and Deployment Wizard in ready mode to Package and Deploy active project PackDep_Demo.



Note: As this Wizard is launched from the Visual Basic IDE the current project is selected automatically for Packaging and Deployment. If this Wizard is launched as a stand-alone component i.e., via start button and Visual Basic sub menu, it will prompt to select a project.

- ❖ Click Package icon. This will immediately go to the next step unless the active project is already compiled. Otherwise the Wizard will ask to compile as shown in the following figure.27.3



Figure 27.3 Wizard asking to compile the active project or to locate the executable file of the active project.

- ❖ Click compile button to continue.
- ❖ After the Wizard compiles the project will prompt to choose the package type as shown in the following figure.

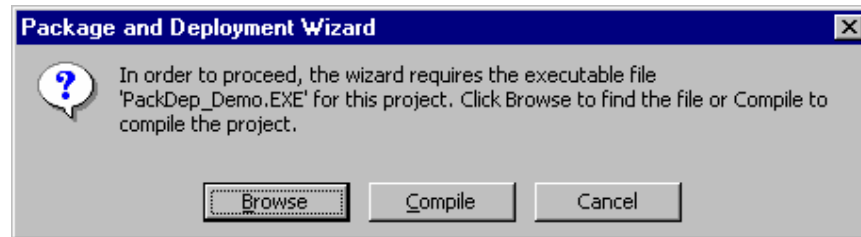


Figure 27.4 Wizard prompting to select the package type.

- ❖ Select Standard Setup Package and click Next.
- ❖ In this step the Wizard will prompt to choose a folder to assemble the package. Click New folder button and enter Setup as name of the new folder that is created as sub folder to the PDDemo folder as shown in the following figure 27.5 and click Next.

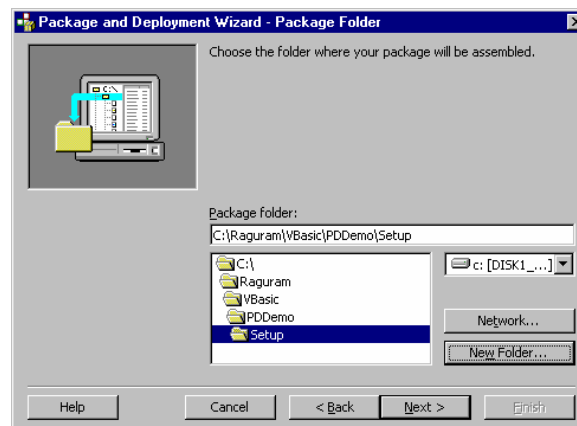


Figure 27.5 Wizard prompting to select a folder to assemble the package.

- ❖ Visual Basic applications consist of more files than just the executable. The next dialog lists the files that need to be installed with the executable. This dialog provides a Add button to include other files such as .mdb (database file), .hlp,(help file) etc. As there are no such files for this case, simply click Next.
- ❖ This step deals with the size of .cab files by providing two options – Single Cab file and Multiple cab files (Figure 5.21). If the package is to be distributed by floppy disks then multiple cab files option must be chosen and cab size no

longer than the floppy must be specified. As floppy disks are not used in this example select Single Cab file option and click Next.

Note: If the package type Internet Package then instead of .cab file .htm file will be created which can be used to download the package using a Web browser such as Internet Explorer.



Figure 27.6 Wizard prompting to select Cab options.

- ❖ In this step Wizard prompts to enter a title that is to be displayed while installing the application. Enter PackDep Demonstration as shown in the following figure 27.7.

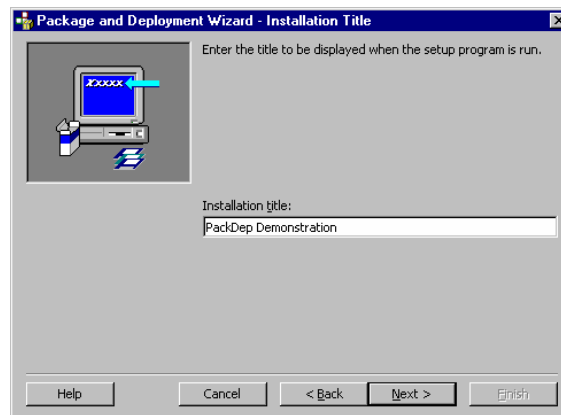


Figure 27.7 Wizard prompting a title to display during installation.

- ❖ In this step the Wizard prompts to specify a location to place an icon to launch the application. By default the Wizard has created a group with the application's name and then an icon as shown in the following figure 27.8.

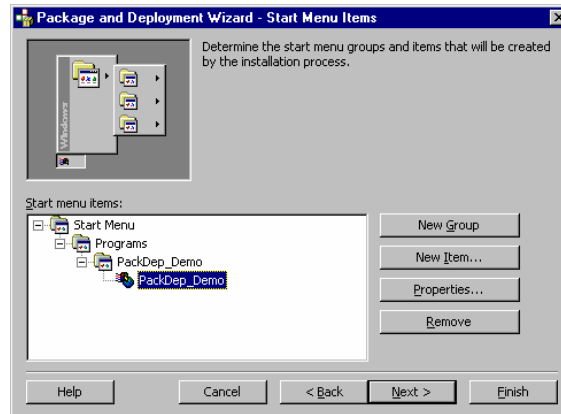


Figure - 27.8 Wizard prompting to specify a location to place the application's icon.

- ❖ As the application has only a single icon, the standard is to create that icon under Programs. Click PackDep_Demo group and click Remove button. And Click **New Item** button and enter PackDep_Demo as the name of the application and click OK in the dialog box that appears (Figure 27.9). And Click Next.

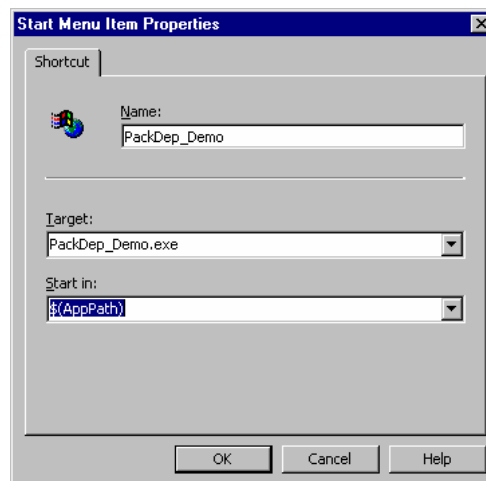


Figure 27.9 Dialog box that gets certain options about new icon.

- ❖ In this step the Wizard enables to change the installation location for each file. This example has only one file. But some other application may require more files such as database (.mdb) file and they have to be installed in some sub folder. Sub folders can be created by adding their name to the macro \$(AppPath) in the Install Location box (shown in the figure 27.10). For example to install database files in a DBs sub folder path can be specified in the Install Location box as \$(AppPath)\DBs.

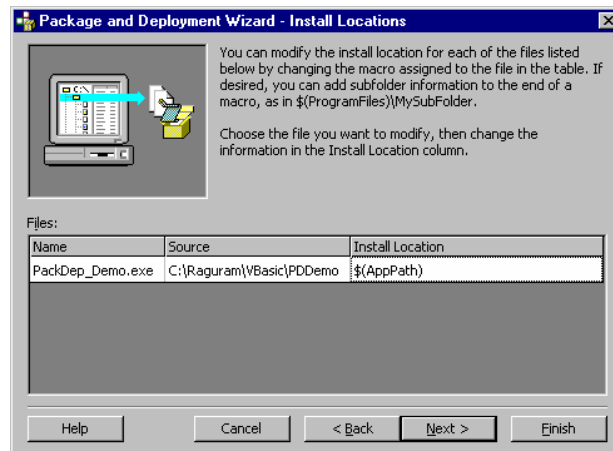


Figure 27.10 Wizard prompting to specify install location for the listed files.

- ❖ As the specified directory \$(AppPath) is sufficient for this example simply click Next.
- ❖ In this step the Wizard presents a dialog box (Figure 27.11) that lets to mark any files as shared. In this example the only listed file is PackDep_Demo.exe. As executable files can't be a shared file simply click Next.

Note : Certain files, such as DLLs and OCXs are considered as shared files. If any these kinds of files are added as part of the installation, they must be marked as shared. So that, when the user uninstall the application, the shared files are verified whether any other application uses these files. These files are removed only if every program, which uses them, is removed.



Figure 27.11 Wizard prompting to mark shared files.

- ❖ In this final step of packaging process the wizard prompts to enter a script (explained shortly) name under which settings of the current session are saved. Simply accept the default script name (shown in the following figure- 27.12) and click Finish.

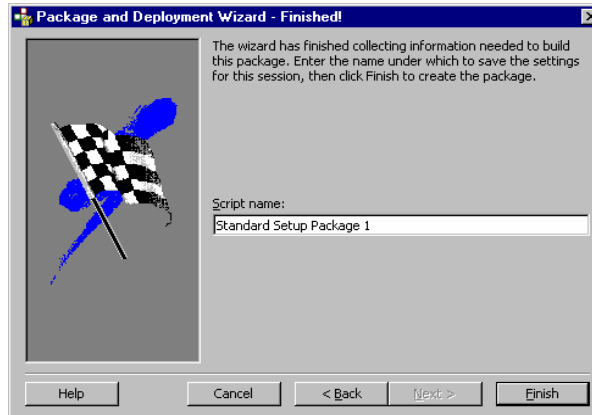


Figure 27.12 Wizard prompting to enter script name.

- ❖ After the wizard finishes building the installation package, it generates a report with some important messages about what was accomplished. Read the report and then click **Close** on the report and the wizard.

Deploying the PackDep_Demo application

In the package method itself, an application can be packaged to a floppy disk and network drive straight away. The main disadvantage in this method is that if another copy is required in a floppy disk then the entire application must be repackaged. So first the package method is used to package the application and the deployment method is used to make multiple copies. Moreover deploying the package in a web site is possible only in the Deployment method. This section shows how to deploy the PackDep_Demo application in a new folder.

To Deploy the PackDep_Demo application

Choose Deploy icon in the Package and Deployment Wizard. This presents a dialog as in Figure 27.13.



Figure 27.13 Wizard prompting to select a package to deploy.

- ❖ If an application has more than one package created with different settings or by different package types then the Package to Deploy combo will list their corresponding script names. From this list desired package can be selected for deployment. For the PackDep_Demo application there is only one package with the script name Standard Setup Package 1 and it is selected by default. So simply click Next.
- ❖ This step presents a dialog (Figure 27.14) to select the deployment method – Folder or Web publishing. If Web publishing option is selected the Wizard will permit to remove files or add more files for deployment. Then the Wizard will prompt to enter an URL (Unified Resource Locator – a Web site address) to deploy the files. As this example is going to deploy the files in a new folder select Folder option and click Next.

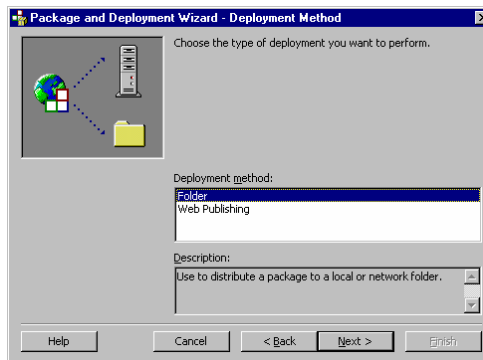


Figure 27.14 Wizard prompting to select a deployment method.

- ❖ In this step the Wizard will prompt (Figure 27.15) to select a folder to deploy the files. Select a path and create a new folder PackDep_App and click Next.

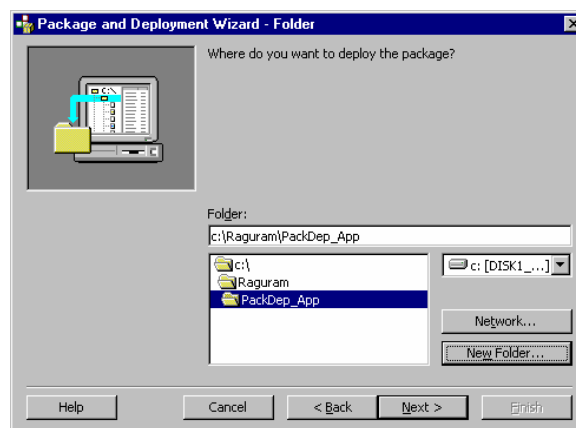


Figure 27.15 Wizard prompting to select folder to deploy the packaged files.

- ❖ In this final step the Wizard will ask a script name to store the current session (deployment session) settings as in the final set of packaging session. Accept the default name and click finish.
- ❖ After the wizard finishes deployment process, it generates a report about what was accomplished. Read the report and then click **Close** on the report and the wizard

27.5 Short Summary

- When creating standard and Internet packages, the Package and Deployment Wizard can also be used to create dependency files. Dependency files list the run time components that must be distributed with the application's project files.
- The Internet Package option will appear in the Package Type list for the ActiveX Projects only
- Standard package and Increment package are two kinds of application packaging wizard.

27.6 Brain Storm

1. Write a note on overall steps in the packaging process?
2. Write a note on overall steps in the deployment process?

Lab Unit - 27 (2 Real Time Hrs)

Create a Setup program to install the following utility which displays date and time In the text box whenever user presses the button.

