



CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)

An Autonomous Institution Accredited with 'A' Grade by NAAC



E-Content

On

“Visual Basic Programming”

Prepared By: Prof. Himanshu Dehariya

Department of Computer Science & Electronics



CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)

An Autonomous Institution Accredited with 'A' Grade by NAAC



Graphical User Interface

A GUI is a type of computer human interface on a computer. It solves the blank screen problem that confronted early computer users. These early users sat down in front of a computer and faced a blank screen, with only a prompt. The computer gave the user no indication what the user was to do next. GUI is an attempt to solve this blank screen problem. In computing, a graphical user interface (GUI) is a type of user interface that allows users to interact with electronic devices with images rather than text commands. *GUIs* can be used in computers, hand-held devices such as MP3 players, portable media players or gaming devices, household appliances and office equipment. A *GUI* represents the information and actions available to a user through graphical icons and visual indicators such as secondary notation, as opposed to text-based interfaces, typed command labels or text navigation. The actions are usually performed through direct manipulation of the graphical elements.

Elements of Graphical User Interface: There are two categories of GUI Elements

1) Structural Elements

Window- A window is an area on the screen that displays information, with its contents being displayed independently from the rest of the screen.

Menus- Menus allow the user to execute commands by selecting from a list of choices. Options are selected with a mouse or other pointing device within a GUI. A keyboard may also be used

Icons- An icon is a small picture that represents objects such as a file, program, web page, or command. They are a quick way to execute commands, open documents, and run programs.

Controls- Interface element that a computer user interacts with, and is also known as a control or Widget.

Tabs- A tab is typically a rectangular small box which usually contains a text label or graphical icon associated with a view pane. When activated the view pane, or window, displays widgets associated with that tab; groups of tabs allow the user to switch quickly between different widgets.

2) Interaction Elements

Cursor- A cursor is an indicator used to show the position on a computer monitor or other display device that will respond to input from a text input or pointing device.

Pointer- One of the most common components of a GUI on the personal computer is a pointer: a graphical image on a screen that indicates the location of a pointing device, and can be used to select and move objects or commands on the screen. A pointer commonly appears as an angled arrow, but it can vary within different programs or operating systems.

Selection- A selection is a list of items on which user operations will take place. The user typically adds items to the list manually, although the computer may create a selection automatically.

Adjustment Handle- A handle is an indicator of a starting point for a drag and drop operation. Usually the pointer shape changes when placed on the handle, showing an icon that represents the supported drag operation.



CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)

An Autonomous Institution Accredited with 'A' Grade by NAAC



Frontend & Backend

Frontend: It is responsible for collecting input in various form from users & processing it to confirm to specification, the backend can use. The front end is an interface between user & back end. A "front-end" application is one that application users interact with directly.

Ex: VB, Java, Dot Net etc

Backend: A back end is always be a database. It is accessed indirectly through an external application program. A "back-end" application or program serves indirectly in support of the front-end services, usually by being closer to the required resource or having the capability to communicate with the required resource. The back-end application may interact directly with the front-end.

Ex: Oracle, MS-Access, SqlServer etc

In client/server applications, the client part of the program is often called the **front end** & server part is called the **back end**.

Integrated Development Environment (IDE)

An IDE also called instigated design environment or integrated debugging environment is a software application that provides comprehensive facilities to computer programmers for s/w development. Following are the components of an ideal IDE.

- 1) **A source code editor-** It provides the facility to write or edit code. (Automatically)
- 2) **A compiler and/or interpreter-** It translates the high level code into machine instruction understand by the computer.
- 3) **Build automation tools-**It is the act or scripting a wide verity of tasks like compiling, packaging, running test, deployment and documentation.
- 4) **A Debugger-** To debug & trace the error in the application.
- 5) **GUI-** An environment which provides facility for developing application. A GUI represents the information and actions available to a user through graphical icons and visual indicators.

Programming Languages

A programming language is an artificial language designed to express computations that can be performed by a machine, particularly a computer. Programming languages can be used to create programs that control the behavior of a machine and/or to express algorithms precisely. A programming language is a notation for writing programs, which are specifications of a computation or algorithm. A programming language is a computer language programmers use to develop applications, scripts, or other set of instructions for a computer to execute. Example: BASIC, COBOL, PASCAL, VISUAL BASIC, C, C++, JAVA etc



CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)

An Autonomous Institution Accredited with 'A' Grade by NAAC



Procedural Programming Language

Procedural Programming can sometimes be used as a synonym for imperative programming. Procedures, also known as routines, subroutines, methods, or functions simply contain a series of computational steps to be carried out. Any given procedure might be called at any point during a program's execution, including by other procedures or itself. It is a list of instructions telling a computer, step-by-step, what to do, usually having a linear order of execution from the first statement to the second and so forth with occasional loops and branches. A procedural programming language is one where programs are organized into blocks of code called variously "subroutines", "functions", or "procedures", each of which handles one particular task. The main function of the program (often actually called "main") then makes a series of calls to these procedures in order to achieve its goal. Example: C, Pascal, Basic etc.

Object-Oriented Programming Language (OOPs)

Object-Oriented Programming is a programming paradigm using "objects" – data structures consisting of data fields and methods together with their interactions – to design applications and computer programs. OOPs programming techniques may include features such as data abstraction, encapsulation, messaging, modularity, polymorphism, and inheritance. In contrast, the object-oriented approach encourages the programmer to place data where it is not directly accessible by the rest of the program. Instead the data is accessed by calling specially written functions, commonly called methods, which are either bundled in with the data or inherited from "class objects" and act as the intermediaries for retrieving or modifying those data.

The programming construct that combines data with a set of methods for accessing and managing those data is called an object. An object-oriented program may thus be viewed as a collection of interacting objects, as opposed to the conventional model, in which a program is seen as a list of tasks (subroutines) to perform. In OOP, each object is capable of receiving messages, processing data, and sending messages to other objects. An object-oriented program will usually contain different types of objects, each type corresponding to a particular kind of complex data to be managed or perhaps to a real-world object or concept such as a bank account. Example: C++, Java etc

Event-Driven Programming Language

In computer programming, event-driven programming or event-based programming is a programming paradigm in which the flow of the program is determined by events—i.e., sensor outputs or user actions (mouse clicks, key presses) or messages from other programs or threads. Event-driven programming can also be defined as an application architecture technique in which the application has a main loop which is clearly divided down to two sections: the first is event selection (or event detection), and the second is event handling. Event-driven programs can be written in any language, although the task is easier in languages that provide high-level abstractions. Some integrated development environments provide code generation assistants that automate the most repetitive tasks required for event handling.

Event driven programming is a particular type of paradigm that functions as a result of some form of input. This input can be from somebody operating the human computer interface or it can also be influenced by messages and orders that are received from another computer program. This sort of programming is very common in computer operating systems and graphical user interfaces. This means you are likely to come across event driven programming when you are playing a computer game or navigating your computer's user interface. It makes use of input devices such as a mouse or a joystick, as well as giving users the ability to move around and interact with a system with relative ease. Event driven programming is greatly beneficial because of how user friendly it makes computer applications. Event Driven Programming may or may not be Object-Oriented. Example: Visual Basic, Dot Net, PHP etc



CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)

An Autonomous Institution Accredited with 'A' Grade by NAAC



Difference between Standard Module and Class Module

Standard Module:

- 1) Place for variable declarations, procedures etc. and we can access these procedures wherever needed in the project.
- 2) Available only for the existing modules where they are used.

Class Module:

- 1) Unlike standard module, members of class like object's properties, events can be accessed only whenever an instance of class is created.
- 2) We can make use of it whenever an object reference of the particular class is created for number of applications.

How to use VB Compiler to Compile / Debug and Run the Programs

This describes the various debugging resources available in the VBA Editor (VBE) and how to use them. These are basic descriptions of the various diagnostic tools and how to use them. See the on-line help for more details.

Compiling Visual Basic Program

Compiling your Visual Basic applications is a process that happens throughout the development phase. You compile applications when debugging, setting flags to stop code to follow it through the process. When you are ready to distribute the application, the final compilation is processed and an executable is made for users to run on their computers. Compiling Visual Basic is a simple process that can be carried out in following steps.

- Open the project by clicking the "File" menu and selecting "Open Project."
- Click the "F5" button with the project code opened. This compiles the code and starts it in debug mode. If you have stop points set, the code will pause before running these lines of code for you to evaluate variables and code logic.
- Click the "Debug" menu item and select "Start Debugging." This does the same as the F5 key.
- Hold the "Ctrl" button and press F5. This compiles the project without debug mode, so it is the same process as your users running the application.

Debugging Visual Basic Program: The toolbar buttons are described below.

Immediate Window

The Immediate Window is a window in the VBE in which you can enter commands and view and change the contents of variables while you code is in Break mode or when no macro code is executing. (Break mode is the state of VBA when code execution is paused at a break point). To display the Immediate Window, press CTRL+G or choose it from the View menu. In the Immediate Window, you can display the value of a variable by using the ? Command. Simply type ? Followed by the variable name and press Enter. VBA will display the contents of the variable in the Immediate Window. For example,

?ActiveCell.Address

\$A\$10

You can also execute VBA commands in the Immediate Window by omitting the question mark and entering the command followed by the Enter key:

Debug.Print: You can use the **Debug.Print** statement anywhere in your code to display messages or variable values in the Immediate Window. These statements don't require any confirmation or acknowledgement from the user so they won't affect the operation of your code. For example, you can send a message to the Immediate Window when a particular section of code is executed.



CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)

An Autonomous Institution Accredited with 'A' Grade by NAAC



```
' some code
```

```
Debug.Print "Starting Code Section 1"
```

The use of **Debug.Print** statements makes it easy to track the execution of your code. **Debug.Print** statements have no effect on the execution of your code and so it is safe to leave them in code projects that are distributed to end users. **Debug.Print** statements send messages to the Immediate Window, so you should have this window open in order to see the messages.

Debug.Assert :

The syntax for **Debug.Assert** is: **Debug.Assert (condition)**

Where condition is some VBA code or expression that returns true (any numeric non-zero value) or False (a zero value). If condition evaluates to False or 0, VBA breaks on that line (see Breakpoints, below). For example, the following code will break on the **Debug.Assert** line because the condition ($X < 100$) is false.

```
Dim X As Long
X = 123
Debug.Assert (X < 100)
```

Debug.Assert is a useful way to pause code execution when special or unexpected conditions occur. It may seem backwards that **Debug.Assert** breaks execution when **condition** is False rather than True, but this peculiarity traces its roots back to early C-language compilers.

Break Points

A break point is a setting on a line of code that tells VBA to pause execution immediately before that line of code is executed. Code execution is placed in what is called **break mode**. When VBA is in break mode, you can enter commands in to the Immediate Window to display or change the values of variables. To put a break point on a line of code, place the cursor on that line and press F9 or choose "Toggle Breakpoint" from the Debug menu. To remove a break point, place the cursor on the line with the break point and press F9 or choose "Toggle Breakpoint" from the Debug menu. When a line contains a break point, it is displayed with a brick colored background. Immediately before this line of code is executed, it will appear with a yellow background. Remember, when a break point is encountered, code execution is paused but that line of code has not yet executed. You cannot place break points on blank lines, comment lines, or variable declaration lines (lines with **Dim** statements). After a break point is encountered, you can resume normal code execution by pressing F5 or choosing "Continue" from the Run menu, or stepping through the code line by line (see below). Note that break points are not saved in the workbook file. If you close the file, all break points are removed. Breakpoints are preserved as long as the file is open.

Watch Window

The Watch Window allows you to "watch" a specific variable or expression and cause code execution to pause and enter break mode when the value of that variable or expression is True (non-zero) or whenever that variable is changed. (Note, this is not to be confused with the Watch object and the Watches collection). To display the Watch Window, choose it from the View menu. To create a new watch, choose Add Watch from the Debug menu. This will display the Add Watch window, shown below. There are three types of watches, shown in the Watch Type group box. "Watch Expression" causes that watch to work much like the Locals Window display. It simply displays the value of a variable or expression as the code is executed. "Break When Value Is True" causes

Department of Computer Science & Electronics



CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)

An Autonomous Institution Accredited with 'A' Grade by NAAC



VBA to enter break mode when the watch variable or expression is True (not equal to zero). "Break When Value Changes" causes VBA to enter break mode when the value of the variable or expression changes value. You can have many watches active in your project, and all watches are displayed in the Watch Window. This makes it simple to determine when a variable changes value.

Running Visual Basic Program

To run a VB program you have to click on the green arrow button on the tool bar it will compile and run the program, you can run program through the keyboard by pressing F5 function key of the keyboard

Modular Environment

It is the environment that supports modular programming. Modular programming is a software design technique that increases the extent to which software is composed of separate, interchangeable components, called **modules** by breaking down program functions into modules, each of which accomplishes one function and contains everything necessary to accomplish this. Conceptually, modules represent a separation of concerns, and improve maintainability by enforcing logical boundaries between components. Modules are typically incorporated into the program through interfaces. A module interface expresses the elements that are provided and required by the module. The elements defined in the interface are detectable by other modules. The implementation contains the working code that corresponds to the elements declared in the interface.

Object Orientated Programming in VB

Even if Visual Basic isn't a full-fledged object-oriented programming language, you can still use its classes to better organize your code into truly reusable modules and design your applications entirely using concepts derived from the Object-Oriented Design discipline. Class modules can immensely improve your productivity, help you solve many common and intricate programming problems, and even permit you to perform tasks that would be extremely difficult, if not impossible, otherwise. Most important, objects are the base on which almost every feature of Visual Basic is implemented. For example, without objects you can't do serious database programming, you can't deliver Internet applications. In short, you can do little or nothing without a firm grasp on what objects are and how you can take advantage of them.

A **Class** is a portion of the program (a source code file, in Visual Basic) that defines the properties, methods, and events. In a word, behavior of one or more objects that will be created during execution.

An **Object** is an entity created at run time, which requires memory and possibly other system resources, and is then destroyed when it's no longer needed or when the application ends.

Note: Also read about Polymorphism, Inheritance, and Encapsulation.

Class Module

Class modules (.CLS extension) are foundation of object oriented programming in VB. We can write code in class modules to create new objects. These new objects can include our own customized properties and methods; although custom objects cannot have their own events. We can also use the keyword, New to create multiple copies of our objects. Class Module is a user defined data type. It has data members (methods and variables). It can be accessed by creating the instance called object of that class. For class modules the extension is **.CLS**. Creating a class module in Visual Basic is straightforward: just issue an Add Class Module command from the Project menu. A new code editor window appears on an empty listing. By default, the first class module is named *Class1*, so the very first thing you should do is change this into a more appropriate name.



CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)
An Autonomous Institution Accredited with 'A' Grade by NAAC



Creating a new Class, and Object

Creating a class in Visual Basic is very simple: just select the **Add Class Module** command from the **Project** menu. A new code editor window appears on an empty listing. Visual Basic automatically add a class module named **Class1**, so the very first thing you should do is change the Class name in the Project Properties window in a more appropriate name. The first version of our class includes only a few properties. These properties are exposed as **Public** members of the class module itself.

To create a Class Module (or Class)

1. Start Visual Basic.
2. In the New Project dialog box, select Standard EXE, and then click OK.
3. On the Project menu, click Add Class Module.
4. In the Add Class Module dialog box, select Class Module, and then click Open.
5. In the Properties window, set the Name property for the class module to Student.

Now write the following code in the declaration section of the Student class module

```
Public RNo As Integer
Public SName As String
Public DOB As Date
```

This is a very simple class, which consists of **Public** properties, Once you've created a class, you can create an instance of that class then you can use the properties of that class. The following example creates an instance of the **Student** class, and sets and retrieves its properties:

To create Object of the Class

1. Add a new Form then place a command button on **Form1**.
2. In the **Click event** for the command button, type the following:

```
Dim S As New Student (Creating the object of Student class)
```

```
Private Sub Command1_Click()
    S.RNo = 111
    S.SName = "HIMANSHU"
    S.DOB = "12/07/2011"
    MsgBox "ROLLNO=" & S.RNo & " " & "NAME=" & S.SName & " " & "DATE OF BIRTH=" &
    S.DOB
End Sub
```

The New keyword: The *New* keyword (when used in a *Set* command) tells Visual Basic to create a brand-new instance of a given class. The keyword then returns the address of the instance data area just allocated.

The Set command: The *Set* command simply copies what it finds to the right of the equal sign into the object variable that appears to the left of it. This value can be, for example, the result of a *New* keyword, the contents of another variable that already exists, or the result of an expression that evaluates to an object. The only other tasks that the *Set* command performs are incrementing the reference counter of the corresponding instance data area and decrementing the reference counter of the object originally pointed to by the left-hand variable (if the variable didn't contain the Nothing value):



CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)
An Autonomous Institution Accredited with 'A' Grade by NAAC



Visual Basic Environment

To start visual basic go to start->programs->Microsoft visual studio 6.0->Microsoft visual basic 6.0(Click). Then following visual basic environment will appear on the screen

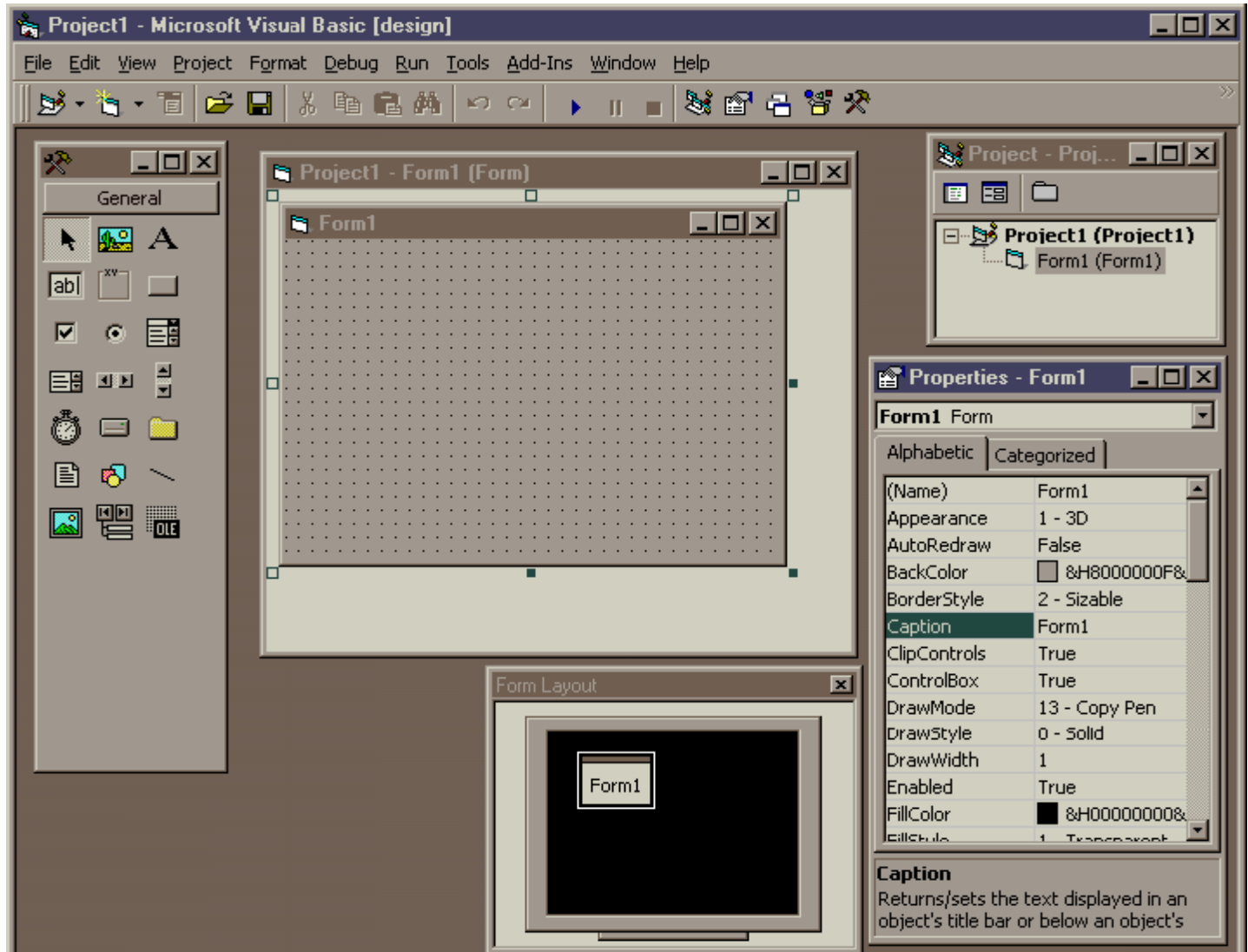


Figure -New project environment.

The Menu bar is at the very top. Menu bar contains most of the utilities that are available to you in the Visual Basic environment. Such as File, Edit, View, etc. Click on each of these items to see the available utilities.

File Edit View Project Format Debug Run Tools Add-Ins Window Help

Figure - Visual Basic Environment Menu bar.

Under the Menu bar is the Tool bar. Tool bar contains some of the most often used items from the Menu bar in small icons. Move the mouse pointer on each icon and leave it for a moment and the function of the icon will appear in a small sidebar.











Figure - Visual Basic environment Tool bar.

The Toolbox in the Visual Basic environment contains the controls (also referred to as objects) that are most often used in developing applications. Throughout this textbook you will learn how to use all these controls.



Figure - Visual Basic environment Toolbox.

In the following table the controls in the toolbox will be introduced one at a time. In the following chapters these controls will be used in different applications and their properties will be explored.

Control	Name	Operation	Property	Event
	Pointer	It does not draw a control. Pointer lets you select, resize or move a control already on a form.	-	-
	Picture box	Displays graphics, as a container that receives output from graphics and print methods.	Picture	Picture1_Click()
	Label	Displays text that cannot be changed, like caption under a graphic. It can be changed by program codes.	Caption	Label1_Click()
	Textbox	Holds text that the user can either enter or change, or text generated by application.	Text	Text1_Change()
	Frame	Allows you to create a visual or functional grouping for controls. Draw the Frame first, then draw controls inside the frame, to form a group.	Caption	Frame1_DragDrop()
	Button	Creates a button the user can choose (click) to carry out a sequence of instructions.	Caption	Command1_Click()
	Checkbox	To choose between yes/no, true/false, and include/exclude or multiple choices when more than one item may be chosen.	Caption, Value	Check1_Click()
	Option button	This control is used to select one item from a group of items.	Caption, Value	Option1_Click()



CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)

An Autonomous Institution Accredited with 'A' Grade by NAAC
















	Combo box	Combines the features of list box and text box. The user can either choose an item from the list or enter a value in the text box.	Text, List	Combo1_Change()
	List box	Displays a list of items from which the user can choose one.	Name, List	List1_Click()
	Timer	Used to activate a specific event at set intervals. This control is invisible at run time.	Name, Interval	Timer1_Timer()
	Vertical Scroll bar	Provides a graphical tool for moving through lists or selecting data ranges.	Name, Max, Min	VScroll1_Change()
	Horizontal Scroll bar	Provides a graphical tool for moving through lists or selecting data ranges.	Name, Max, Min	HScroll1_Change()
	Drive List Box	Displays valid disk drives at run time.	Name	Drive1_Change()
	Directory List Box	Displays directories and paths at run time.	Name	Dir1_Change()
	File List Box	Displays a list of files at run time.	Name	File1_Click()
	Shape	Used to draw a variety of shapes such as a rectangle, square, rounded square, oval, or circle, etc.	Name, Shape	-
	Line	Used to draw a variety of line styles on your form at design time.	Name	
	Image	Displays a graphical image from a bitmap, icon, metafile as well as JPG or GIF files. It is decorative and uses fewer resources than a Picture Box.	Name	Image1_Click()
	Data	Provides access to data in databases through bound controls on your form.	Name, DatabaseName, RecordSource	Data1_Validate
	OLE	Allows you to link and embed objects from other applications in your VB application.	Name	OLE1_Updated

Figure-Most frequently used controls in the Visual Basic environment.

There are three other windows that appear in the Visual Basic environment. They are, Properties window and the.

Project window: Project window provides an explorer type view of all the forms and modules in the project, while Properties window provides a list of all the properties available for each control with their corresponding value.

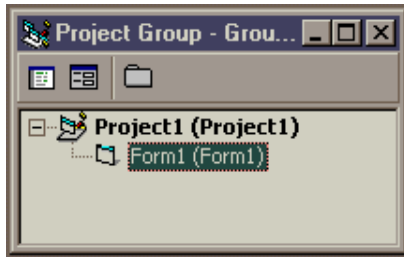
Form Layout window: Layout window shows the relative location of the forms on the screen. As you change the location and size of the forms the changes are reflected in the Layout window. You can change the location of the form by moving its location in the Form Layout window. To change the location of the form in the Form Layout

Property Window: This window contains properties of the controls that drag on the form. You can set the various properties by changing values in the corresponding property field. The control you select on the form its corresponding property will display on the property window

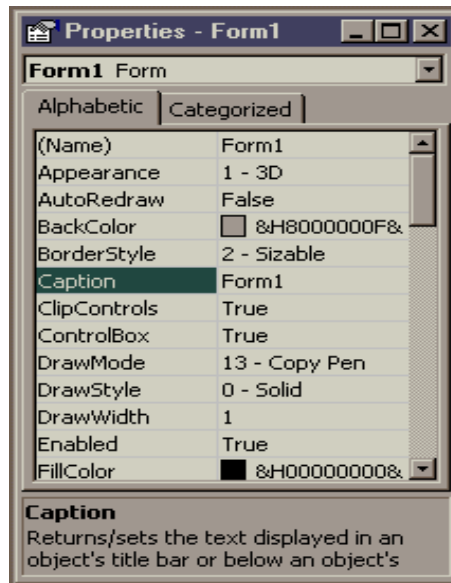


CHRISTIAN EMINENT COLLEGE, INDORE

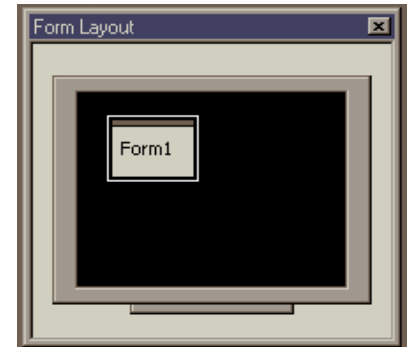
(Academy of Management, Professional Education and Research)
An Autonomous Institution Accredited with 'A' Grade by NAAC



Project Explorer



Property Window



Form Layout windows.

Form Window: Form window, as shown in Figure is the main window in the Visual Basic environment. This is the window within which the application interface is developed. You develop the application interface by placing controls in the Form. You can change the size of the form by clicking on the handles and dragging them.

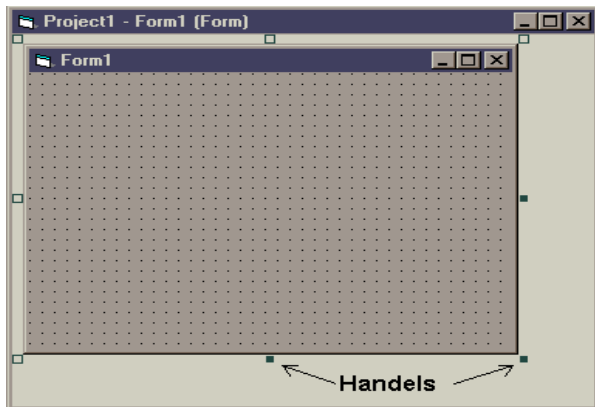


Figure -Form window

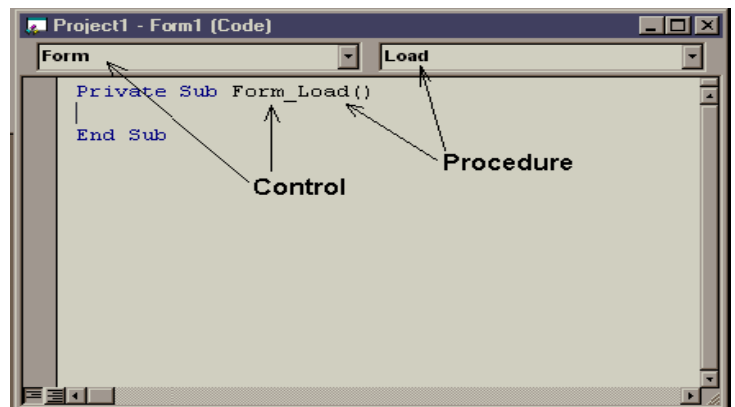


Figure-Code editing window.

Code Editing Window: Once the controls are placed in this window and their properties are assigned in the Properties window it is time to attach codes to each control as needed. In order to write the code for each control you must first bring up the code-editing window for that control. You can do that by double clicking on the control. If you wish to attach code to a Form, you should double click on the Form itself. Once the code edit window appears, the control that the code is being written for is shown in the top left Combo Box and the event procedure is shown in the top right Combo Box. Visual Basic is an event driven programming language. This simply means that the applications that are developed using Visual Basic, much like any other windows applications, will act upon the user's actions such as Click, DoubleClick, DragDrop, MouseDown, etc. Whenever you want the control in your application to respond to an event, you put the instructions in the appropriate event procedure. Of course the user does not initiate all the events. Sometimes events are consequence of other events. For example as you start an application the main Form is loaded in the memory and shows up on the screen. This is a Form_Load event, which simply means that the Load event procedure is activated on the Form control. Using the arrow in the left Combo Box will show all the controls that are used in the application and clicking on the arrow of the right Combo Box shows all the available procedure for the control that is shown on the left Combo Box.



CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)

An Autonomous Institution Accredited with 'A' Grade by NAAC



Visual Basic Data Type

Data Type	Size	Range
Integer	2 bytes	-32,768 Through 32,767
Long Integer	4 bytes	-2,147,483,648 through 2,147,483,648
Single (precision floating point)	4 bytes	-3.042823×10^{38} through 3.042823×10^{38}
Double (precision floating point)	8 bytes	$-1.79769313486232 \times 10^{308}$ through $1.79769313486232 \times 10^{308}$
Currency	8 bytes	-922,337,203,685,477.5808 through 922,337,203,685,277.5808
String	1 byte/character	0 through 65,535 character
Boolean	2 byte	true/false
Date	8 bytes	January 1,100 through December 31,9999
Variant	16 bytes(number)	All data type range ,22 bytes+1 byte/Character(string)

Variant Data type: When you do not specify the type of a variable in a declaration statement it takes the default type which is a variant. As you can see in the above table, variant type allows the variable to take both text (string) and number. Variables with variant data type can store all the data types mentioned above and switch format automatically. It is highly recommended that you use specific data types for specific tasks and variables. For example a counter should always be an integer. Declaring variables at the beginning of each program emphasizes discipline, and saves memory and increases program efficiency. As you can see from the above table using variant data types, although convenient, is not the most efficient way of programming.

Variable Declaration

There are two ways to declare variables in Visual Basic-

Implicit Declaration: "Implicit" declaration is when you start using a variable simply by assigning a specific data type to it.

Explicit Declaration: "Explicit" declaration is when you declare each variable at the beginning of the program. Using "Option Explicit" statement in the general declaration section of a program forces Visual Basic to generate an error message if a variable is not declared. This way you make sure that you are using only explicit form of variable declaration. To declare a variable as a specific data type uses the following format:

Dim <variable name> as <data type>

Ex. Dim *Counter* as Integer
Dim *Velocity* as Single
Dim *Distance* as Double
Dim *LastName* as String

Constant Declaration: Constants are numbers that does not change value throughout your project. In many engineering problems there are constant numbers that are used quite often. In Visual Basic you can declare a constant by using the keyword "Const". One of the more familiar constants is π . To declare π as a constant in Visual Basic uses the following format.

Const pi= 3.14159265

Department of Computer Science & Electronics



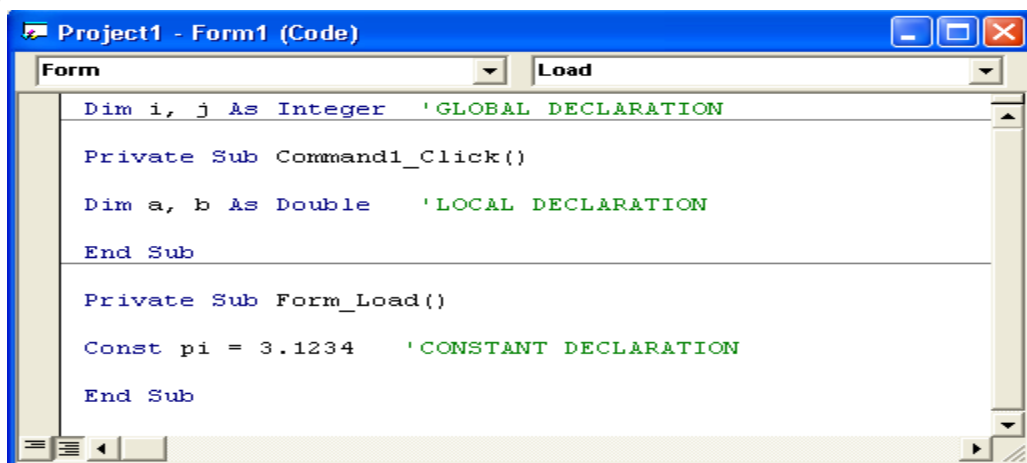
CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)

An Autonomous Institution Accredited with 'A' Grade by NAAC



Local and Global Variables: If a variable is declared in a specific event procedure, i.e. `cmdDisplay_Click()`, then it is a local variable and can only be used in that event procedure with the declared format. If you want to use this variable in another event procedure, then you must declare it again. In order to make a variable available to all the controls and procedures in your program, you need to declare it as a global variable. This is done by declaring the variable in the General (declaration) event procedure. This event procedure is available through the Code Window as the first item in the “Object” drop-down list and first item in the “Proc.” drop-down list as seen in the following figure.



Operators in Visual Basic

Mathematical operations in Visual Basic take place by the use of operators. Operators will help us add; subtract, multiply, divide and exponentiation numerical values. Following is a list of mathematical operations used in VB.

Mathematical Operators			
Operator	Meaning	Example	Precedence
\wedge	Exponentiation	$3 \wedge 3 = 27$	First
$*$	Multiplication	$3 * 3 = 9$	Second
$/$	Division	$3 / 3 = 1$	Second
\backslash	Integer Division	$7 \backslash 2 = 3$	Third
$-$	Subtraction	$3 - 3 = 0$	Fifth
$+$	Addition	$3 + 3 = 6$	Fifth
Mod	Remainder	$9 \text{ Mod } 2 = 1$	Forth
Comparison Operator			
$=$	Equal to	$a = b$	
$<>$	Not Equal to	$a <> b$	
$>$	Greater Than	$a > b$	
$<$	Less Than	$a < b$	
$<=$	Greater Than or Equal to	$a <= b$	
$>=$	Less Than or Equal to	$a >= b$	
Logical Operator			
AND	If both conditional expressions are True, then the result is True.		
OR	If either conditional expression is True, then the result is True.		
NOT	If the conditional expressions are False, then the result is True. If the conditional expressions are True, then the result is False.		
XOR	If one and only one of the conditional expressions is True, Then the result is True. If both are True, or if both are False, then the result is False.		



CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)

An Autonomous Institution Accredited with 'A' Grade by NAAC



Mathematical Function: For more number crunching power, Visual Basic provides a set of mathematical functions. Functions operate on a variable, an expression or a value that is provided as an argument (n). Following is a list of some of the Visual Basic functions.

Function	Purpose
Sin(n)	Returns the sine of the angle n. the angle is expressed in radians
Cos(n)	Returns the cosine of the angle n. the angle is expressed in radians
Tan(n)	Returns the tangent of the angle n. the angle is expressed in radians
Atn(n)	Returns the arctangent of n in radians
Abs(n)	Returns the absolute value of n
Exp(n)	Returns the constant e to the power n
Rnd(n)	Generates a random number between 0 and 1
Sgn(n)	Returns 1 if n is less than zero, 0 if n is zero, and +1 if n is greater than zero
Sqrt(n)	Returns the square root of n
Str(n)	Converts a numeric value to a string
Val(n)	Converts a string value to a number

Control Structure & Loops in Visual Basic

If-Then-Else Statement		
If condition Then statement1 statement2 End If	If condition Then statement1 Else statement2 End If	If Condition1 Then Statement1 Statement2 ElseIf Condition2 Then Statement3 Statement4 ElseIf Condition3 Then Statement5 Statement6 End If
Example: Dim a, b As Integer If a > b Then MsgBox "a is greater" End If	Example: Dim a, b As Integer If a > b Then MsgBox "a is greater" Else MsgBox "b is greater" End If	Example: Dim Temperature As Single If Temperature < 50 Then Text1.Text = "Low Temperature" ElseIf Temperature > 50 and Temperature < 100 Then Text1.Text = Temperature ElseIf Temperature > 100 Then Text1.Text = "High Temperature" End If

Select Case Statement	For Next Loop	Do While Loop
Select Case Variable Case value1 Statement1 Statement2 Case value2 Statement3 Statement4 End Select	For counter = start To end Step value statements Next counter	Do While condition statements Loop

Department of Computer Science & Electronics



CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)
An Autonomous Institution Accredited with 'A' Grade by NAAC



Example: Dim Sem As Integer Select Case Sem Case 1 <i>Label1.Caption="You are a Freshman"</i> Case 3 <i>Label1.Caption="You are a Sophomore"</i> Case 5 <i>Label1.Caption="You are a Junior"</i> Case 7 <i>Label1.Caption="Graduate Soon"</i> End Select	Example: x = 0 For i = 2 To 10 Step 2 x = x + i Next i	Example: m = 0 x = 0 Do While m<20 x = x + i m = m + 1 Loop
While Loop		
While condition statements Wend		
Example: Dim number As Integer number = 1 While number <=100 number = number + 1 Wend		

Array

One Dimension Array:

Two Dimension Array:

Immediate		
11	22	33
44	55	66
77	88	99

```
Dim A(0 To 5) As Integer
```

```
Private Sub Command1_Click()  
    For i = 0 To 4  
        x = InputBox("enter array value")  
        A(i) = x  
    Next i  
End Sub
```

```
Private Sub Command2_Click()  
    For j = 0 To 4  
        List1.AddItem (A(j))  
    Next j  
End Sub
```

```
Dim A(0 To 2, 0 To 2) As Integer
```

```
Private Sub Command1_Click()  
    For i = 0 To 2  
        For j = 0 To 2  
            x = InputBox("enter array value")  
            A(i, j) = x  
        Next j  
    Next i  
End Sub
```

```
Private Sub Command2_Click()  
    For i = 0 To 2  
        For j = 0 To 2  
            Debug.Print A(i, j) & vbTab;  
        Next j  
        Debug.Print  
    Next i  
End Sub
```



CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)

An Autonomous Institution Accredited with 'A' Grade by NAAC



Text File

A text file (sometimes spelled "textfile": an old alternate name is "flatfile") is a kind of computer file that is structured as a sequence of lines. A text file exists within a computer file system. The end of a text file is often denoted by placing one or more special characters, known as an end-of-file marker, after the last line in a text file." Text file" refers to a type of container, while plain text refers to a type of content. Text files can contain plain text, but they are not limited to such. A file that holds text. The term *text file* is often used as a synonym for ASCII file, a file in which characters are represented by their ASCII codes. A text file is a computer file that stores a typed document as a series of alphanumeric characters, usually without visual formatting information. The content may be a personal note or list, a journal or newspaper article, a book, or any other text that can be rendered accurately in typewritten form. Text files are similar to word processing files in that the content of both is primarily textual; they differ in that text files usually do not record information such as character style and size, pagination, or other details that would specify the appearance of a finished document. Some computer operating systems make a basic distinction between a text file, which is intended to be translated directly into human-readable text, and a binary file, which is interpreted directly by the computer.

Record: One logical section of a file that holds a related set of data. If the file contains Student information, a record would hold the information on one student: name, address, studentID, etc. If there are 5,000 students registered, the file contains 5,000 records.

Field: Part of a record that defines specific information. In the Student record, FirstName, LastName, StudentID, are fields. The field is the lowest element in the file. Even if the information consists of one character, Sex is M or F, it is still considered a separate field. The field is the equivalent of the variable - we call it a variable when it is used to store data in memory and call it a field when it stores in a file.

I/O: It stands for Input/Output. Whenever you work with a file you have to have ways of reading data from the file (that's Input) and ways of writing data to the file (that's Output). I/O operations consist of all those commands that let you read and write files.

Types of Files

Sequential file: This is a file where all the information is written in order from the beginning to the end. To access a given record you have to read all the records stored before it. It is in fact like listening to a tape - you can go forward or back but you can't jump directly to a specific song on the tape. It can even be of use when there is a large amount of data to be stored, provided it all has to be processed at one time, eg: a file of invoices to produce a statement at month-end.

Random file: a file where all records are accessible individually. It is like a CD where you can jump to any track. This is useful when there is a large quantity of data to store and it has to be available quickly: you have to know if a part is in stock for a customer who is on the phone; the program doesn't have time to search through 10,000 records individually to locate the correct one. This method of storage became popular when hard-disk drives were developed. Random files which support "direct access" by record number

Binary file: this is a special, compacted form of the random file. Data is stored at the byte level and you can read and write individual bytes to the file. This makes the file access very fast and efficient. We won't be covering this type of file in these exercises. If you need to find out more about it, go to the VB Reference Manual. Binary files are "unstructured" files which are read from or written to as series of bytes, where it is up to the programmer to specify the format of the file



CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)
An Autonomous Institution Accredited with 'A' Grade by NAAC



Modes of Files

Mode in the Open statement indicates how the file will be used. There are five access modes:

Input: open file for sequential input; the file will be read sequentially starting at the beginning.

Output: open file for sequential output; records will be written sequentially starting at the beginning; if the file does not exist, it is created; if it does exist, it is overwritten.

Random: open file for random read and write; any specific record can be accessed.

Append: sequential output to the end of an existing file; if the file does not exist it is created; it does not overwrite the file.

Binary: open file for binary read and write; access is at byte level.

Opening and Closing Files

Some commands that are common to both Sequential and Random files. The first command to include in a program that needs to work with files is the Open command. Open assigns the file to a numbered file handle, also called a channel, or sometimes a buffer. The format of the command is:

Open "Filename" [For Mode] [AccessRestriction] [LockType] As #FileNumber

For example: Open "MyFile.txt" For Random Read Lock Read As #1

MyFile.txt is the name of the file in the disk directory. For Random means that access to the records can be random; if access is not specified, For random is the default value. Read restricts access to Read-only - the user cannot write or change the records. Lock Read means that only the person reading the record can have access to it at any given time; it is not shared among users. As #1 means the file is assigned file handle #1; for all processing in the program, it will always be referred to as #1, not its Filename. AccessRestriction and LockType are parameters that are used mostly with files in a network environment. You use them when you want the file to be shared or not, and you want to prevent certain users from changing or deleting things that they shouldn't. For Mode in the Open statement indicates how the file will be used. There are five access modes we have discussed above-Input, Output, Random, Append, Binary, Random is used by default. Once processing is finished, you need to Close all the files that have been opened. The format for the Close statement is:

Close #FileNumber1 [, #FileNumber2]...

You can close any number of files with one Close statement. Eg: Close #1, #2, #3. The following statement closes all open files: Close

Writing and Reading a Sequential file

There are two commands that allow you to write data to a sequential file: **Print** and **Write**. They work in almost the same way but, the Print command does not separate the fields in the file in quite the same way which makes the data harder to read afterwards. There is really no valid reason to use Print when creating a sequential file. The format of the Write command is:

Write #FileNumber, OutputList
Print #FileNumber, OutputList

Where FileNumber is the number the file was opened with and OutputList is one or more variables you want to write to the file.

Department of Computer Science & Electronics



CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)

An Autonomous Institution Accredited with 'A' Grade by NAAC



Creating a Sequential File

There are many ways to organize data in a sequential file. The following steps create a new sequential file and write data to it.

1. Choose a file name. A file name can contain up to 255 characters consisting of letters, digits, and a few other assorted characters (including spaces and periods). In this book we use 8.3 format names; that is, each name has a base name of at most 8 characters, and optionally a period followed by a three-letter extension.
2. Choose a number from 1 through 511 to be the reference number of the file. While the file is in use, it will be identified by this number.
3. Execute the statement-**Open “filespec” For Output As #n** where n is the reference number. This process is referred to as opening a file for output. It establishes a communications link between the computer and the disk drive for storing data *onto* the disk.
4. Place data into the file with the **Write #** statement. If a is a string, then the statement **Write #n, “a”** writes the string a surrounded by quotation marks into the file. If c is a number, then the statement **Write #n, c** writes the number c , without any leading or trailing spaces, into file number n . The statement **Write #n, “a”, c** writes a and c as before, but with a comma separating them. Similarly, if the statement **Write #n** is followed by a list of several strings and/or numbers separated by commas, then all the strings and numbers appear as before, separated by commas. After each **Write #** statement is executed, the “carriage return” and “line feed” characters are placed into the file.
5. After all the data have been recorded in the file, execute **Close #n** where n is the reference number. This statement breaks the communications link with the file and dissociates the number n from the file.

Example

```
Private Sub Command1_Click()  
Open "PIONEER.TXT" For Output As #1  
Write #1, "ENIAC"  
Write #1, 1946  
Close #1  
End Sub
```

Adding Items to a Sequential File

Data can be added to the end of an existing sequential file with the following steps.

1. Choose a number from 1 through 511 to be the reference number for the file. It need not be the number that was used when the file was created.
2. Execute the statement-**Open “filespec” For Append As #n** where n is the reference number. This procedure is called **opening a file for append**. It allows data to be output and recorded at the end of the specified file.
3. Place data into the file with **Write #** statements.
4. After all the data have been recorded into the file, close the file with the statement-**Close #n**.

The Append option for opening a file is intended to add data to an existing file. However, it also can be used to create a new file. If the file does not exist, then the Append option acts just like the Output option and create the file. The three options, Output, Input, and Append, are referred to as **modes**. A file should not be open in two modes at the same time. For instance, after a file has been opened for output and data have been written to the file, the file should be closed before being opened for input. An attempt to open a nonexistent file for input terminates the program with the “File not found” error message.

Department of Computer Science & Electronics



CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)

An Autonomous Institution Accredited with 'A' Grade by NAAC



There is a function that tells us whether a certain file has already been created. If the value of **Dir** ("**filespec**") is the empty string "", then the specified file does not exist. (If the file exists, the value will be the file name.) Therefore, prudence often dictates that files be opened for input with code such as

```
If Dir("filespec") <> "" Then
Open "filespec" For Input As #1
Else
message = "Either no file has yet been created or "
message = message & "the file is not where expected."
MsgBox message, , "File Not Found"
End If
```

There is one file-management operation -deleting an item of information from a file. An individual item of a file cannot be changed or deleted directly. A new file must be created by reading each item from the original file and recording it, with the single item changed or deleted, into the new file. The old file is then erased and the new file renamed with the name of the original file. Regarding these last two tasks, the Visual Basic statement **Kill** "**filespec**" removes the specified file from the disk and the statement **Name** "**oldfilespec**" As "**newfilespec**" Changes the filespec of a file. (**Note:** The **Kill** and **Name** statements cannot be used with open files. So doing generates a "File already open" message.)

Sequential File Handling Functions

FreeFile(): Returns an Integer representing the next file number available for use by the **Open** statement.

Syntax: FreeFile[(rangenumbers)]

The optional rangenumbers argument is a Variant that specifies the range from which the next free file number is to be returned. Specify a 0 (default) to return a file number in the range 1 – 255, inclusive. Specify a 1 to return a file number in the range 256 – 511.

EOF(): Returns an Integer containing the Boolean value **True** when the end of a file opened for sequential Input has been reached.

Syntax: EOF(filename)

The required filename argument is an Integer containing any valid file number.

FileLen(): Returns a Long specifying the length of a file in bytes.

Syntax: FileLen(pathname)

The required pathname argument is a string expression that specifies a file.

LOF(): Returns a Long representing the size, in bytes, of a file opened using the **Open** statement.

Syntax: LOF(filename)

The required filename argument is an Integer containing a valid file number.

Seek(): Returns a Long specifying the current read/write position within a file opened using the **Open** statement.

Syntax: Seek(filename)

The required filename argument is an Integer containing a valid file number.



CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)
An Autonomous Institution Accredited with 'A' Grade by NAAC



Working with Random Access Files

The command to write records to the Random file is **Put**. Its format is:

Put #Filename, [RecordNumber], Variable

RecordNumber is optional and, if it's omitted, variable is written in Next record position after last Put or Get statement.

The command to read records from a Random file is: **Get**. Its format is:

Get #FileNumber, [RecordNumber], Variable

If RecordNumber is omitted, next record is read from the file.

Random Access Files

Sequential files do not have any structure. As a matter of fact, the structure is defined by the code that reads the file rather than in the file itself. This feature is advantageous from a readability standpoint but limits the functions that a programmer can use to search through the file. For example, Visual Basic does not contain a function to jump to a specific record in a sequential file because Visual Basic does not know the file's structure. One way to move some structure into the file itself is to store user-defined record types rather than just string data. You can then open them in Random mode and are not limited to sequential access.

Creating a Record Type

Custom record types are really just user-defined data types. You create user-defined data types by using the Type statement. Type declarations are entered in the general declarations section of a Code window. The following code uses the Type statement to declare an Employee record type:

Private Type Employee

EmpID As Integer

LName As String * 30

Fname As String * 20

Title As String * 20

End Type

The custom record type can then be accessed from code by using dot notation, as in the following example:

Dim emp1 As Employee

Emp1.Fname= "Joe"

Emp1.Lname= "Smith"

Emp1.Title= "Chicken Plucker"

Emp1.EmpID = 12345

Notice that accessing fields in a custom data type is similar to accessing an object's properties,

Opening a Random Access File

The main difference between opening a random access file and a sequential file is that you must specify the record length in the Open statement. The record length for a user-defined data type can be obtained by using the Len statement on a variable of that type, as in the following example:

Dim emp1 As Employee

Open "D:\EMPINFO.DAT" for Random As #1 Len = Len(emp1)

Department of Computer Science & Electronics



CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)
An Autonomous Institution Accredited with 'A' Grade by NAAC



The preceding line of code opens the file EMPINFO.DAT for random access. The Len= part of the Open statement tells Visual Basic that subsequent reads and writes to the file will assume that the file contains only records the length of emp1.

Adding Records with *Put* statement

After you open a file for random access, use the Put statement to store records in the file. The syntax of the Put statement is as follows:

Put filename,[recnumber],variablename

The following code uses the Put statement in a For loop to write five records of type Employee to file #1:

```
For i = 1 To 5
emp1.LName = InputBox$("Enter Last Name")
emp1.Fname = InputBox$("Enter First Name")
emp1.Title = InputBox$("Enter Title")
emp1.EmpID = i
Put #1, , emp1
Next i
```

Notice the omission of the recnumber parameter, which specifies the location in the file where the new record should be written. This approach is useful for altering existing records in a file. If you do not specify this parameter, the record is written to the current file pointer location.

Retrieving Records with *Get* statement

To retrieve records from a random file, use the Get statement. The Get statement can be used to read information back into your record type, The syntax of the Get statement is similar to Put statement :

Get filename,[recnumber],variablename

The following example that reads record number 4 in the file into the variable emp1 and then displays the Title field:

```
Get #1, 4, emp1
MsgBox "Employee title is " & emp1.Title
```

If you omit the recnumber parameter, Get behaves like the sequential Line Input statement; that is, each subsequent Get reads the next record.

Random Access with *Seek* statement

To move from record to record, use the Seek statement. The Seek statement has two parameters, the file number and record number, as shown here:

```
Seek #1, 3
```

The preceding line of code causes the next Put or Get to access record number 3.



CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)
An Autonomous Institution Accredited with 'A' Grade by NAAC



File-Manipulation Functions

Most of the file-manipulation commands in Visual Basic are as straightforward as their MS-DOS equivalents, although with a few limitations. These commands are as follows-

Copy a file	-	FileCopy <source>, <dest>
Delete one or more files	-	Kill <path>
Rename a file-		Name <oldname> As <newname>
Create a new folder -		MkDir <pathname>
Remove an empty folder-		Rmdir <pathname>
Change current directory-		ChDir <pathname>
Change current drive-		ChDrive <drive>

Copying Files: The FileCopy command has the limitation that you cannot use wildcards to specify multiple files. FileCopy can copy files locally or over a network, as shown in the following examples: The following line copies a file while changing its name:

FileCopy "D:\My Documents\Hey Now.txt", "C:\DATA\TEST.TXT"

The FileCopy statement automatically overwrites an existing file, unless the file is read-only or locked open by another application.

Deleting Files: Visual Basic also allows you to delete files by using the Kill statement. Kill can use wildcards to specify multiple files, as in the following example:

Kill "D:\NewDir*.doc"

Renaming Files: The Name statement is like MS-DOS's RENAME command but can be used on only one file at a time:

Name "C:\Windows\Desktop\TEST1.TXT" AS "D:\NewDir\TEST2.TXT"

In the preceding example, note the MkDir statement, which you probably have guessed is used to create a new directory. The MkDir and Rmdir statements add and remove directories, like their MD and RD counterparts in MS-DOS.

You also can use Name like the MOVE command in MS-DOS if the specified paths are different: Moves the file to a new directory

MkDir "D:\NewDir"

Setting the Current Directory: By using the ChDir and ChDrive statements, you can set the current working directory on each drive and switch between current drives, eliminating the need to specify the full path for each file operation: 'Change to the desired directory and drive and rename a file

chdir "C:\Windows\Desktop"
chdrive "C:"

--



CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)
An Autonomous Institution Accredited with 'A' Grade by NAAC



File System Objects (FSO)

File System Objects (FSO) completely recast Visual Basic's fundamental powers over disk storage in an object-oriented mold. It makes file access and management considerably easier, even though it isn't quite complete.

Text File Access with FSO

The top in the FSO hierarchy is the **FileSystemObject** class. To read or write a text file, you must first create an instance of this class:

```
Dim myFSO
```

```
Set myFSO = CreateObject ("Scripting.FileSystemObject")
```

The following alternate syntax accomplishes the same thing:

```
Dim myFSO As New Scripting.FileSystemObject
```

The **Scripting** qualifier identifies the library in which the **FileSystemObject** class is defined. To use the class, you do not need to select the Microsoft Scripting Runtime library in the Visual Basic References dialog box, but doing so will give you access to the classes, methods, and properties in the Object Browser. After creating an instance of the **FileSystemObject** class, the next step is to create a **TextStream** object. A **TextStream** object is nothing more than a regular text file enclosed in an FSO wrapper. The **FileSystemObject** has two methods for creating **TextStream** objects:

- **CreateTextFile** creates a new text file. If a file of the same name already exists, it is overwritten.
- **OpenTextFile** opens a text file for reading and/or writing. If the file already exists, new data is appended to existing data.

The syntax for these methods is similar. In these examples, assume that **myFSO** is a **FileSystemObject** and that it has been declared as a type **Variant** or **Object**. The first line of code creates a new file, and the second line opens an existing file:

```
Set ts = myFSO.CreateTextFile (filename [, overwrite [, unicode]])
```

```
Set ts = myFSO.OpenTextFile (filename [, iomode [, create [, format]])
```

Filename is a string expression specifying the name (including path information) of the file. The **overwrite** argument is either **True** or **False**, indicating whether an existing file will be overwritten. If this argument is omitted, the default is **False**. If **overwrite** is **False** and **filename** already exists, an error occurs. You can trap this error to permit the user to verify file overwrites. Set **unicode** to **True** to create a Unicode file, or **False** (the default) for an ASCII file. Set **iomode** to the constants **ForReading** or **ForAppending** to read the file or write data to the end of the file, respectively. A file opened using **ForReading** cannot be written to. Set **create** to **True** or **False** to specify whether a new file will be created if the file named in the **filename** argument does not exist. The default is **False**. The **format** argument is a tristate argument (that is, one that can take on one of three possible values) that determines the format of the file:

- **TriStateTrue** opens the file as Unicode.
- **TriStateFalse** (the default) opens the file as ASCII.
- **TristateUseDefault** uses the system default format setting.



CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)

An Autonomous Institution Accredited with 'A' Grade by NAAC



Properties of the TextStream object.

Property	Description
AtEndOfLine	True if the file pointer is at the end of a line; False otherwise. This property applies only to TextStream files that are open for reading; otherwise, an error occurs.
AtEndOfStream	True if the file pointer is at the end of the file; False otherwise. This property applies only to TextStream files that are open for reading; otherwise, an error occurs.
Column	Returns the column number of the file pointer. The first character on a line is at column 1.
Line	Returns the current line number.

Methods of the TextStream object.

Property	Description
Close	Closes the file associated with the TextStream object. Always execute the Close method when you are done reading/writing the file.
Read(n)	Reads the next n characters from the file and returns the resulting string.
ReadAll	Reads the entire file and returns the resulting string.
ReadLine	Reads an entire line (up to, but not including, the newline character) from a TextStream file and returns the resulting string.
Skip(n)	Skips ahead (moves the file pointer) by n characters.
SkipLine	Skips to the beginning of the next line.
Write(s)	Writes the string s to the file. No extra or newline characters are added.
WriteBlankLines(n)	Writes n blank lines to the file.
WriteLine(s)	Writes the string s to the file followed by a newline character.



CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)

An Autonomous Institution Accredited with 'A' Grade by NAAC



File Management with FSO

The **FileSystemObject** is the “top” object in the FSO hierarchy. As such, it provides access to all of the system’s drives (both local and network), folders, and files. There are several other objects in the hierarchy, and you can see that they correspond to the way in which disk drives are organized:

- **Drive** object, which corresponds to a single disk drive on the system.
- **Folder** object, which corresponds to a single folder on a drive.
- **File** object, which corresponds to a single file in a folder.

The **FileSystemObject** has a **Drives** collection that contains one **Drive** object for each local and network drive. You can query a **Drive** object’s properties to obtain information about the drive, such as its type and the amount of free space. Each **Drive** object contains a single **Folder** object representing the drive’s top-level folder, or root. Each **Folder** object contains a **Folders** collection and a **Files** collection. The **Folders** collection contains a **Folder** object for each subfolder within the folder, and the **Files** collection contains a **File** object for each file in the folder. The **Drives**, **Folders**, and **Files** collections are like any other Visual Basic collection, and they are used the same way. Each **Drive** object has a set of properties that provides information about the physical drive. Except as noted, these properties are all read-only.

Drive objects properties.

Property	Description
AvailableSpace	The amount of space available to a user on the specified drive or network share. Generally the same as the FreeSpace property, but may differ on systems that support quotas.
DriveLetter	The drive letter associated with the drive. Returns a zero-length string for network drives that have not been mapped to a drive letter.
DriveType	A value indicating the type of the drive. Possible values are 0 (unknown), 1 (removable), 2 (fixed), 3 (network), 4 (CD-ROM), and 5 (RAM disk).
FileSystem	The type of file system. Available return types include FAT, NTFS, and CDFS.
FreeSpace	The amount of space available to a user on the specified drive or network share. Generally the same as the AvailableSpace property, but may differ on systems that support quotas.
IsReady	Returns True if the drive is ready, False if not. Used with removable media and CD-ROM drives, returning False if the media has not been inserted.
Path	The path of the drive. This consists of the drive letter followed by a colon.
RootFolder	Returns a Folder object representing the drive’s root path.
SerialNumber	The unique serial number identifying a disk. Use this property to verify that a removable media



CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)

An Autonomous Institution Accredited with 'A' Grade by NAAC



	drive contains the proper media.
ShareName	The share name assigned to a network drive. For non-network drives, a zero-length string.
TotalSize	The total capacity of the drive.
VolumeName	The volume name of the drive. You can write to this property to change a drive's volume name.

The Folder Object

A Folder object represents a single folder, or subdirectory, on a drive. You use the object's methods to copy, move, or delete the folder and you use the object's properties to obtain information about the folder. Perhaps most important, a **Folder** object contains two collections, **Files** and **Subfolders**, that provide access to the files and subfolders within the folder. Table 4 explains the properties of the **Folder** object.

Properties of the Folder object.	
Property	Description
DateCreated	Date and time the folder was created.
DateLastAccessed	Date and time the folder was last accessed.
DateLastModified	Date and time the folder was last modified.
Drive	Drive letter of the drive where the folder resides.
Files	A Files collection containing all the files in the folder.
IsRootFolder	True if the folder is the root, False otherwise.
Name	Sets or returns the name of the folder.
ParentFolder	Returns a Folder object representing the folder's parent folder.
Path	The path of the folder, including drive letter.
ShortName	The short name used by programs that require the old 8.3 naming convention.
ShortPath	The short path used by programs that require the old 8.3 naming convention.
Size	The size, in bytes, of all files and subfolders contained in the folder.
SubFolders	A Folders collection containing one Folder object for each subfolder.



CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)

An Autonomous Institution Accredited with 'A' Grade by NAAC



Because each **Folder** object contains information about its parent folder and its subfolders, you can easily traverse the entire folder structure on a drive. The **Copy** method copies the folder and its contents to a new location. The syntax is as

f.Copy destination[, overwrite]

The **destination** argument specifies the destination where the folder is to be copied to. Set **overwrite** to **True** (the default) to overwrite existing files or folders, or to **False** otherwise.

The **Move** method moves the folder and its contents from one location to another. The syntax is:

f.Move destination

Here, **destination** specifies the destination where the folder is to be moved to.

The **Delete** method deletes a folder and its contents. The syntax is:

f.Delete [force]

The File Object

The **File** object. In the FSO model, each file on a disk is represented by a **File** object. There are two ways to create a **File** object. If you know the name and path of the file, you can use the **FileSystemObject**'s **GetFile** method. Assuming that **fs** is an instance of the **FileSystemObject** class:

Dim f

Set f = fs.GetFile(filespec)

The argument **filespec** contains the filename, including a relative or absolute path. An error occurs if the file called out in **filespec** does not exist. Note that executing **GetFile** does not open the file, it simply returns a **File** object linked to the file. You can also use the object's methods for certain types of file manipulation. The methods and properties are explained as follows

The File object's methods.

Method	Description
Copy dest [, overwrite]	Copies the file to dest . An existing file is overwritten only if overwrite is True (default).
Move dest	Moves the file to dest .
Delete [force]	Deletes the files. Read-only files are deleted only if force is True . The default is False .
OpenAsTextStream	Opens the file and returns a TextStream object.

The File object's properties.

Property	Description
----------	-------------



CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)

An Autonomous Institution Accredited with 'A' Grade by NAAC



Attributes	Returns a value summarizing the file's attributes, as explained in detail below.
DateCreated	Date and time the file was created.
DateLastAccessed	Date and time the file was last accessed.
DateLastModified	Date and time the file was last modified.
Drive	Drive letter of the drive where the file resides.
Name	Sets or returns the name of the file.
ParentFolder	Returns a Folder object representing the file's parent folder.
Path	The path of the file, including drive letter.
ShortName	The short filename used by programs that require the old 8.3 naming convention.
ShortPath	The short path used by programs that require the old 8.3 naming convention.
Size	The size, in bytes, of the file.

VBScript

It is a scripting language developed by Microsoft and supported by Microsoft's Internet Explorer Web browser. VBScript is based on the Visual Basic programming language, but is much simpler. In many ways, it is similar to JavaScript. It enables Web authors to include interactive controls, such as buttons and scrollbars, on their Web pages. VBScript, Microsoft's Visual Basic Scripting Edition, is a scaled down version of Visual Basic. While it doesn't offer the functionality of Visual Basic, it does provide a powerful, easy to learn tool that can be used to add interaction to your web pages. If you are already experienced in either Visual Basic or Visual Basic for Applications, you will find working with VBScript easy and should be immediately productive. Don't be concerned if you haven't worked in another version of Visual Basic. VBScript is easy to learn, even for the novice developer. We can say the it is a-

- VBScript is a scripting language
- A scripting language is a lightweight programming language
- VBScript is a light version of Microsoft's programming language Visual Basic
- VBScript is only supported by Microsoft's browsers (Internet Explorer)

How does it work?

When a VBScript is inserted into an HTML document, Internet Explorer browser will read the HTML and interpret the VBScript. The VBScript can be executed immediately, or at a later event. VBScript only works in Microsoft browsers (Internet Explorer).

Department of Computer Science & Electronics



CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)
An Autonomous Institution Accredited with 'A' Grade by NAAC



VBScript Variables

VBScript variables are used to hold values or expressions. A variable can have a short name, like x, or a more descriptive name, like carname. In VBScript, all variables are of type ***variant*** that can store different types of data. Rules for VBScript variable names:

- Must begin with a letter
- Cannot contain a period (.)
- Cannot exceed 255 characters

Declaring (Creating) VBScript Variables

You can declare VBScript variables with the Dim, Public or the Private statement. Like this:

```
Dim x  
Dim carname
```

Assigning Values to Variables

You assign a value to a variable like this:

```
carname="Volvo"  
x=10
```

Example

```
<html>  
<body>  
<script type="text/vbscript">  
document.write("This is my first  
VBScript!")  
</script>  
</body>  
</html>
```

Example

```
<HTML>  
<HEAD>  
<TITLE>Working with VBScript</TITLE>  
</HEAD>  
<BODY>  
  <FORM NAME="frmExercise2">  
    <TABLE>  
      <TR>  
        <TD><B>Quantity :</B></TD>  
        <TD><INPUT TYPE="Text" NAME="txtQuantity"  
" SIZE=5></TD>  
      </TR>  
      <TR>  
        <TD><B>Unit price :</B></TD>  
        <TD><INPUT TYPE="Text" NAME="txtUnitPric  
e" SIZE=5></TD>  
      </TR>  
    </TABLE>  
    <INPUT TYPE="Button" NAME="cmdCalculate" VAL  
UE="Cost">  
  </FORM>  
</BODY>  
</HTML>
```



CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)
An Autonomous Institution Accredited with 'A' Grade by NAAC



Report Handling

Once you have gone to all the trouble of developing and managing a database, it is nice to have the ability to obtain printed or displayed information from your data. The process of obtaining such information is known as creating a data report. There are two steps to creating a data report.

- 1) We need to create a Data Environment. This is designed within Visual Basic and is used to tell the data report what is in the database.
- 2) We create the Data Report itself. This, too, is done within Visual Basic. The Data Environment and Data Report files then become part of the Visual Basic project developed as a database management system.

Example - Phone Directory - Building a Data Report

We will build a data report that lists all the names and phone numbers in our phone database. We will do this by first creating a Data Environment, then a Data Report. We will then reopen the phone database management project and add data reporting capabilities.

Creating a Data Environment

1. Start a new **Standard EXE** project.
2. On the Project menu, click **Add Data Environment**. If this item is not on the menu, click **Components**. Click the **Designers** tab, and choose **Data Environment** and click **OK** to add the designer to your menu.
3. We need to point to our database. In the **Data Environment** window, right-click the **Connection1** tab and select **Properties**. In the **Data Link Properties** dialog box, choose **Microsoft Jet 3.51 OLE DB Provider**. Click **Next** to get to the **Connection** tab. Click the **ellipsis** button. Find your phone database (mdb) file. Click **OK** to close the dialog box.
4. We now tell the **Data Environment** what is in our database. Right-click the **Connection1** tab and click **Rename**. Change the name of the tab to Phone. Right-click this newly named tab and click **Add Command** to create a **Command1** tab. Right-click this tab and choose **Properties**. Assign the following properties:

Command Name - PhoneList
Connection - Phone
DataBase Object - Table
ObjectName - PhoneList

5. Click **OK**. All this was needed just to connect the environment to our database.
6. Display the properties window and give the data environment a name property of denPhone. Click **File** and **Save** denPhone As. Save the environment in an appropriate folder. We will eventually add this file to our phone database management system. At this point, my data environment window looks like this (I expanded the PhoneList tab by clicking the + sign):





CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)
An Autonomous Institution Accredited with 'A' Grade by NAAC



Creating a Data Report

Once the Data Environment has been created, we can create a Data Report. We will drag things out of the Data Environment onto a form created for the Data Report, so make sure your Data Environment window is still available.

1. On the Project menu, click Add Data Report and one will be added to your project. If this item is not on the menu, click Components. Click the Designers tab, and choose Data Report and click OK to add the designer to your menu.
2. Set the following properties for the report:
Name - rptPhone
Caption - Phone Directory
DataSource - denPhone (your phone data environment - choose, don't type)
DataMember - PhoneList (the table name - choose don't type)
3. Right-click the **Data Report** and click **Retrieve Structure**. This establishes a report format based on the **Data Environment**.
4. Note there are five sections to the data report: a Report Header, a Page Header, a Detail section, a Page Footer, and a Report Footer. The headers and footers contain information you want printed in the report and on each page. To place information in one of these regions, right-click the selected region, click Add Control, then choose the control you wish to place. These controls are called data report controls and properties are established just like you do for usual controls. Try adding some headers.
5. The Detail section is used to layout the information you want printed for each record in your database. We will place two field listings (Name, Phone) there. Click on the Name tab in the Data Environment window and drag it to the Detail section of the Data Report. Two items should appear: a text box Name and a text box Name (PhoneList). The first text box is heading information. Move this text box into the Page Header section. The second text box is the actual value for Name from the PhoneList table. Line this text box up under the Name header. Now, drag the Phone tab from the Data Environment to the Data Report. Adjust the text boxes in the same manner. Our data report will have page headers Name and Phone. Under these headers, these fields for each record in our database will be displayed. When done, the form should look something like this:





CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)
An Autonomous Institution Accredited with 'A' Grade by NAAC



In this form, I've resized the labels a bit and added a Report Header. Also, make sure you close up the Detail section to a single line. Any space left in this section will be inserted after each entry.

6. Click File and Save rptPhone As. Save the environment in an appropriate folder. We will now reopen our phone database manager and attach this and the data environment to that project and add capabilities to display the report.

Accessing the Data Report

1. Reopen the phone directory project. Add a command button named cmdReport and give it a Caption of Show Report. (There may be two tabs in your toolbox, one named General and one named DataReport. Make sure you select from the General tools.)
2. We will now add the data environment and data report files to the project. Click the Project menu item, then click Add File. Choose denPhone and click OK. Also add rptPhone. Look at your Project Window. Those files should be listed under Designers.
3. Use this code in cmdReport_Click:

```
Private Sub cmdReport_Click()  
    rptPhone.Show  
End Sub
```
4. This uses the Show method to display the data report.
5. Save the application and run it. Click the Show Report button and this should appear:



You now have a printable copy of the phone directory. Just click the Printer icon. Notice the relationship with this displayed report and the sections available in the Data Report designer.



CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)

An Autonomous Institution Accredited with 'A' Grade by NAAC



Reports (Using ADODB)

The Data Report. Data Reports enable you to easily display a print preview screen, with print and export buttons, from an ADO data source. All you have to do is provide the data and report layout. In addition to this new reporting tool, Visual Basic also supports Crystal Reports, a popular reporting tool that has been packaged with Visual Basic for quite some time now. Crystal Reports, a product of Seagate software, provides an easy way to graphically design and distribute reports.

Creating a Simple Report

The Data Report is known as an ActiveX Designer, meaning that it is a specialized ActiveX object that integrates into the VB environment. "Using ActiveX Data Objects (ADO)." We'll create a sample Data Report based on a query of the BIBLIO database. To create this report, perform the following steps:

1. Set up the ADO recordset to be used by the report.
2. Add a DataReport object to the Visual Basic project.
3. Design the report by placing fields on the DataReport designer form.
4. Write code to display the report in the program.

Setting Up the Data Source

To begin the sample project, open Visual Basic and start a new Standard EXE project. The next step in creating the sample report has nothing to do with the report itself, but rather the data that goes into the report. Before you can design a report, you have to know what you are reporting. Therefore, you can build on the ADO knowledge and create a simple query that can be used for the DataSource property of the report. if necessary to perform these steps to create the query:

1. Add the reference for Microsoft ActiveX Data Objects to your project.
2. If you have not already set up the BIBLIO ODBC data source, do so now.
3. Declare form-level variables, cn and rs, to represent ADO connection and recordset objects.
4. Draw a command button, cmdFill, on the form and label it Fill Recordset.
5. Add code to the form's Load and Unload events to open and close a connection to the BIBLIO data source.
6. Add code to the button's Click event to populate the recordset with the following SQL query:

"Select * from Titles where [Year Published] >= 1996"

The complete code for the form is as follows-Getting Some Sample Data for the Report-

```
Dim cn As ADODB.Connection
Dim rs As ADODB.Recordset
Private Sub cmdFill_Click()
    Set rs = cn.Execute("Select * from Titles where [Year Published] >= 1996")
    MsgBox "Recordset populated."
End Sub
Private Sub Form_Load()
    Set cn = New ADODB.Connection
    cn.Open "DSN=BIBLIO"
End Sub
Private Sub Form_Unload(Cancel As Integer)
    rs.Close
    cn.Close
End Sub
```



CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)
An Autonomous Institution Accredited with 'A' Grade by NAAC

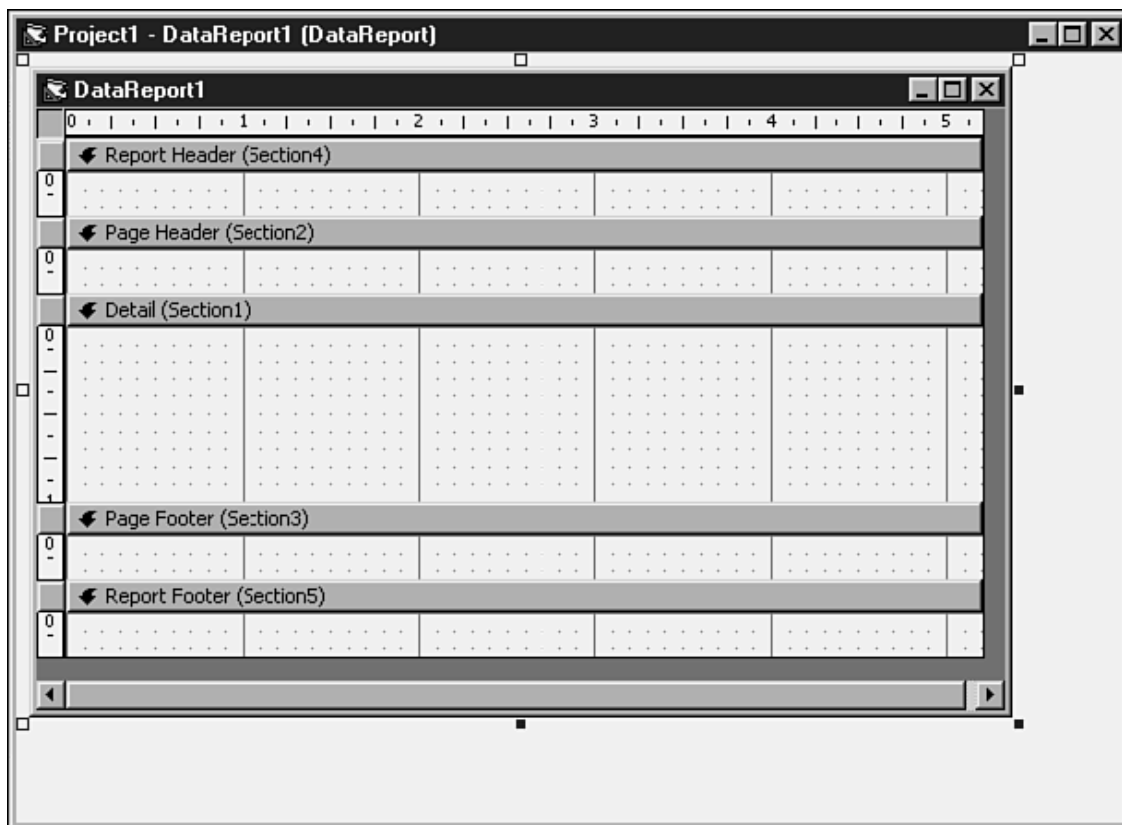


The preceding query creates a recordset containing some records from the Titles table of the BIBLIO sample database. The records were limited to those titles published in or after 1996. When you click the command button, the recordset object rs is populated with the fields from the Titles table. Next, you add a DataReport object to the project and learn how to create a report from the sample recordset.

Adding a Data Report to Your Project

Now that you have data, the next step is to determine how to display that data on a report. First, you need to add a new DataReport object to your Visual Basic project. To do so, choose Project, Add Data Report. You should notice that several things happen immediately:

- A new DataReport object called DataReport1 is added to the Project Explorer window.
- A new DataReport area is added to the Toolbox.
- The Report Designer window appears, as shown in Figure



Notice that a Data Report is divided into several sections, much like reports in Microsoft Access. These sections display the following types of information:

- **Report Header.** Information displayed once, at the top of the report
- **Page Header.** Information displayed at the top of every page
- **Detail Section.** The part of the report that is repeated for each record in the ADO data source
- **Page Footer.** Information displayed at the bottom of every page
- **Report Footer.** Information displayed at the end of the report



CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)
An Autonomous Institution Accredited with 'A' Grade by NAAC



Setting Up the Data Report

While you are in design mode, use the Data Report Designer window to arrange report controls on the report, just as you would arrange custom controls on a form. The types of fields that you can place on a Data Report become available in your Toolbox when the DataReport object has focus. To add information to your report, you first draw the report control in the appropriate section of the report and then set its properties, just as you would with any other control. The controls that can be placed on a Data Report are as follows-

RptLabel -Specifies a label used to display text that is not databound, such as a column name or date printed

RptTextBox - Displays the contents of a database field

RptImage -Contains a picture or other graphic image, such as a company logo

RptLine -Enables you to draw lines on the report—for example, to separate the different sections of the report

RptShape -Enables you to draw a variety of shapes on the control to highlight information & visual effects

RptFunction -Allows you to place a field in the header or footer that contains one of several simple math functions, such as a total of all the values in a particular data field

Displaying the Report

Now that you have determined how you want the report to look, you have to add a few lines of code to display the report. Add a second command button, cmdReport, to the form. Make the caption Display Report. Enter the code from Listing 29.2 in the command button's Click event procedure.

```
Private Sub cmdReport_Click()  
    Dim s As String  
    s = "Prepared by Himanshu"  
    Set DataReport1.DataSource = rs  
    DataReport1.Sections ("rptHeader").Controls ("lblPerson").Caption = s  
    DataReport1.Show  
End Sub
```

The code sets the DataSource property of the DataReport object to the ADO recordset **rs**. Next, the code sets the Caption property of the label control in the report header. Finally, the Show method is invoked, causing the report to be displayed. To test the sample report, run the project and click the Fill Recordset button. After you see the message box stating that the recordset has been filled, click the Display Report button. The Data Report should appear on a separate form in Print Preview mode, as pictured in Figure



CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)

An Autonomous Institution Accredited with 'A' Grade by NAAC



DataReport1

Zoom 100%

Title	Cost	Year Published
AI Agents in Virtual Reality Worlds : Programming Intelligent Vr in C	39.95	1996
The Data Warehouse Challenge : Taming Data Chaos	44.95	1996
Os/2 Goldmine/Book and Cd-Rom	42.95	1996
Visual Basic Algorithms : A Developer's Sourcebook of Ready-To-Run Code	39.95	1996
Visual Basic Internet Programming	39.95	1996
Object-Oriented Software Testing : A Hierarchical Approach	0	1996
Programming With Unix Threads	39.95	1996
Design of Library Automation Systems : File Structures, Data Structures, and Tools	0	1996

Pages: 5



CHRISTIAN EMINENT COLLEGE, INDORE

(Academy of Management, Professional Education and Research)

An Autonomous Institution Accredited with 'A' Grade by NAAC



Event-Driven Programming Language

In computer programming, event-driven programming or event-based programming is a programming paradigm in which the flow of the program is determined by events—i.e., sensor outputs or user actions (mouse clicks, key presses) or messages from other programs or threads. Event-driven programming can also be defined as an application architecture technique in which the application has a main loop which is clearly divided down to two sections: the first is event selection (or event detection), and the second is event handling. Event-driven programs can be written in any language, although the task is easier in languages that provide high-level abstractions. Some integrated development environments provide code generation assistants that automate the most repetitive tasks required for event handling.

Event driven programming is a particular type of paradigm that functions as a result of some form of input. This input can be from somebody operating the human computer interface or it can also be influenced by messages and orders that are received from another computer program. This sort of programming is very common in computer operating systems and graphical user interfaces. This means you are likely to come across event driven programming when you are playing a computer game or navigating your computer's user interface. It makes use of input devices such as a mouse or a joystick, as well as giving users the ability to move around and interact with a system with relative ease. Event driven programming is greatly beneficial because of how user friendly it makes computer applications. Event Driven Programming may or may not be Object-Oriented. Example: Visual Basic, Dot Net, PHP etc
