INTEGRAL UNIVERSITY, LUCKNOW

DIRECTORATE OF DISTANCE EDUCATION

BCA-306/BCS-305 Paper Code: VP/B

VISUAL PROGRAMMING

CONTENTS

UNIT: 1-PROBLEM SOLVING

Contents

Program Development Cycle

Hardware refers to the machinery in a computer system (such as the monitor, keyboard, and CPU) and software refers to a collection of instructions, called a **program** (or **project**), that directs the hardware. Programs are written to solve problems or perform tasks on a computer.

Programmers translate the solutions or tasks into a language the computer can understand.

As we write programs, we must keep in mind that the computer will only do what we instruct it to do. Because of this, we must be very careful and thorough with our instructions.

Performing a Task on the Computer

The first step in writing instructions to carry out a task is to determine what the **output** should be—that is, exactly what the task should produce. The second step is to identify the data, or **input**, necessary to obtain the output. The last step is to determine how to **process** the input to obtain the desired output, that is, to determine what formulas or ways of doing things can be used to obtain the output.

This problem-solving approach is the same as that used to solve word problems in an algebra class. For example, consider the following algebra problem:

How fast is a car traveling if it goes 50 miles in 2 hours?

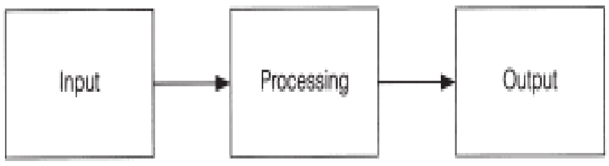
The first step is to determine the type of answer requested. The answer should be a number giving the rate of speed in miles per hour (the output). The information needed to obtain the answer is the distance and time the car has traveled (the input). The formula

rate distance / time

is used to process the distance traveled and the time elapsed in order to determine the rate of speed. That is,

rate = 50miles / 2
= 25 miles / hour

A pictorial representation of this problem-solving process is



We determine what we want as output, get the needed input, and process the input to produce the desired output.
In the following chapters we discuss how to write programs to carry out the preceding operations. But first we look at the general process of writing programs.

Program Planning

A recipe provides a good example of a plan. The ingredients and the amounts are determined by what is to be baked. That is, the output determines the input and the processing. The recipe, or plan, reduces the number of mistakes you might make if you tried to bake with no plan at all. Although it's difficult to imagine an architect building a bridge or a factory without a detailed plan, many programmers (particularly students in their first programming course) frequently try to write programs without first making a careful plan. The more complicated the problem, the more complex the plan must be. You will spend much less time working on a program if you devise a carefully thought out step-by-step plan and test it before actually writing the program.

Many programmers plan their programs using a sequence of steps, referred to as the **program development cycle**. The following step-by-step process will enable you to use your time efficiently and help you design error-free programs that produce the desired output.

1. **Analyze:** Define the problem.

Be sure you understand what the program should do, that is, what the output should be. Have a clear idea of what data (or input) are given and the relationship between the input and the desired output.

2. **Design:** Plan the solution to the problem.

Find a logical sequence of precise steps that solve the problem. Such a sequence of steps is called an **algorithm**. Every detail, including obvious steps, should appear in the algorithm. In the next section, we discuss three popular methods used to develop the logic plan: flowcharts, pseudo code, and top-down charts. These tools help the programmer break a problem into a sequence of small tasks the computer can perform to solve the problem.

Planning also involves using representative data to test the logic of the algorithm by hand to ensure that it is correct.

3. **Choose the interface:** Select the objects (text boxes, command buttons, etc.).

Determine how the input will be obtained and how the output will be displayed. Then create objects to receive the input and display the output. Also, create appropriate command buttons to allow the user to control the program.

4. **Code:** Translate the algorithm into a programming language.

Coding is the technical word for writing the program. During this stage, the program is written in Visual Basic and entered into the computer. The programmer uses the algorithm devised in Step 2 along with knowledge of Visual Basic.

5. **Test and debug:** Locate and remove any errors in the program.

Testing is the process of finding errors in a program, and **debugging** is the process of correcting errors that are found. (An error in a program is called a **bug.**) As the program is typed, Visual Basic points out certain types of program errors. Other types of errors will be detected by Visual Basic when the program is executed; however, many errors due to typing mistakes, flaws in the algorithm, or incorrect usages of the Visual Basic language rules only can be uncovered and corrected by careful detective work. An example of such an error would be using addition when multiplication was the proper operation.

6. **Complete the documentation:** Organize all the material that describes the program.

Documentation is intended to allow another person or the programmer at a later date, to understand the program. Internal documentation consists of statements in the program that are not executed, but point out the purposes of various parts of the program. Documentation might also consist of a detailed description of what the program does and how to use the program (for instance, what type of input is expected). For commercial programs, documentation includes an instruction manual. Other types of documentation are the flowchart, pseudo code, and top-down chart that were used to construct the program.

Although documentation is listed as the last step in the program development cycle, it should take place as the program is being coded.

Programming Tools

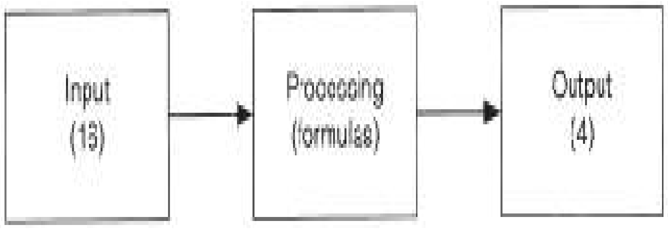
This section discusses some specific algorithms and develops three tools used to convert algorithms into computer programs: flowcharts, pseudo code, and hierarchy charts.

You use algorithms every day to make decisions and perform tasks. For instance, whenever you mail a letter, you must decide how much postage to put on the envelope. One rule of thumb is to use one stamp for every five sheets of paper or fraction thereof. Suppose a friend asks you to determine the number of stamps to place on an envelope. The following algorithm will accomplish the task.

```
1. Request the number of sheets of paper; call it Sheets.          (Input)
2. Divide Sheets by 5.                                        (Processing)
3. Round the quotient up to the next highest whole number; call it Stamps.
                                                              (Processing)
4. Reply with the number Stamps.                                  (Output)
```

The preceding algorithm takes the number of sheets (Sheets) as input, processes the data, and produces the number of stamps needed

(Stamps) as output. We can test the algorithm for a letter with 16 sheets of paper.
1. Request the number of sheets of paper; Sheets = 16.
2. Dividing 5 into 16 gives 3.2.
3. Rounding 3.2 up to 4 gives Stamps = 4.
4. Reply with the answer, 4 stamps.
This problem-solving example can be pictured by

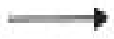
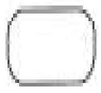
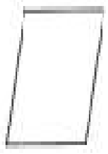
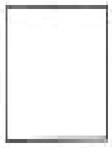
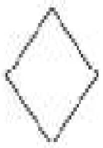
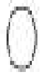
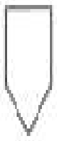
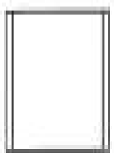
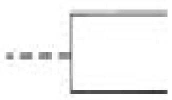


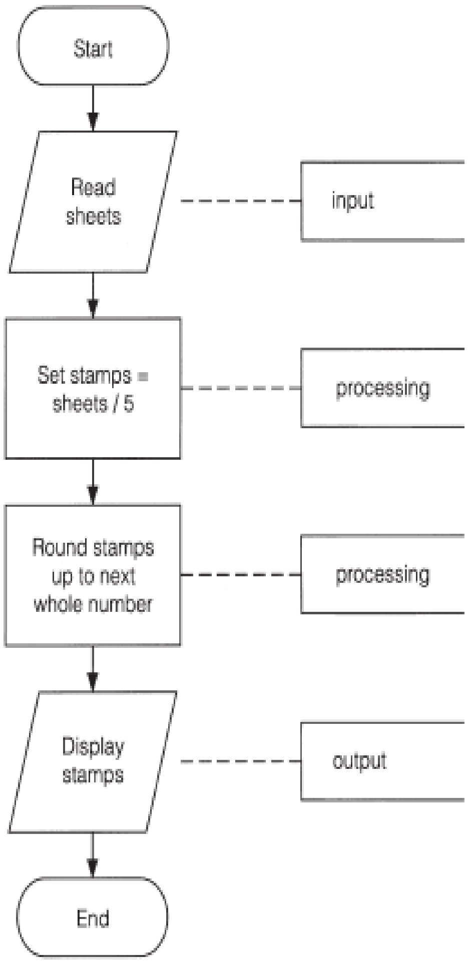
Of the program design tool available, the three most popular are the following:
Flowcharts:
Graphically depict the logical steps to carry out a task and show how the steps relate to each other.
Pseudocode:
Uses English-like phrases with some Visual Basic terms to outline the task.
Hierarchy charts:
Show how the different parts of a program relate to each other.

Flow Charts

A flowchart consists of special geometric symbols connected by arrows. Within each symbol is a phrase presenting the activity at that step. The shape of the symbol indicates the type of operation that is to occur. For instance, the parallelogram denotes input or output. The arrows connecting the symbols, called **flow lines**, show the progression in which the steps take place.

Flowcharts should "flow" from the top of the page to the bottom. Although the symbols used in flowcharts are standardized, no standards exist for the amount of detail required within each symbol.
The main advantage of using a flowchart to plan a task is that it provides a pictorial representation of the task, which makes the logic easier to follow. We can clearly see every step and how each step is connected to the next. The major disadvantage with flowcharts is that when a program is very large, the flowcharts may continue for many pages, making them difficult to follow and modify.

Symbol	Name	Meaning
	*Flowline*	Used to connect symbols and indicate the flow of logic.
	*Terminal*	Used to represent the beginning (Start) or the end (End) of a task.
	*Input/Output*	Used for input and output operations, such as reading and printing. The data to be read or printed are described inside.
	*Processing*	Used for arithmetic and data-manipulation operations. The instructions are listed inside the symbol.
	*Decision*	Used for any logic or comparison operations. Unlike the input/output and processing symbols, which have one entry and one exit flowline, the decision symbol has one entry and two exit paths. The path chosen depends on whether the answer to a question is "yes" or "no."
	*Connector*	Used to join different flowlines.
	*Offpage Connector*	Used to indicate that the flowchart continuesto a second page.
	*Predefined Process*	Used to represent a group of statements that perform one processing task.
	*Annotation*	Used to provide additional information about another flowchart symbol.



PSEUDOCODE

Pseudocode is an abbreviated version of actual computer code (hence, pseudocode). The geometric symbols used in flowcharts are replaced by English-like statements that outline the process. As a result, pseudocode looks more like computer code than does a flowchart.

Pseudocode allows the programmer to focus on the steps required to solve a problem rather than on how to use the computer language. The programmer can describe the algorithm in Visual Basic-like form without being restricted by the rules of Visual Basic. When the pseudocode is completed, it can be easily translated into the Visual Basic language.

The following is pseudocode for the postage stamp problem:

Program: Determine the proper number of stamps for a letter

```
Read Sheets                                                   (input)
Set the number of stamps to Sheets / 5
          (processing)
Round the number of stamps up to the next whole number
          (processing)
```

Display the number of stamps
(output)

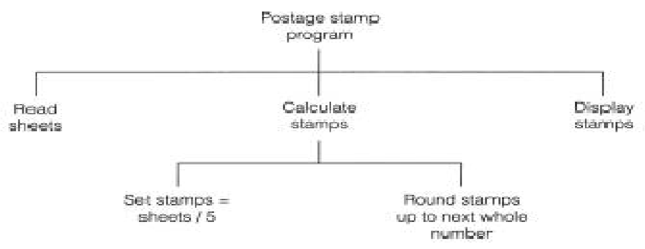
Pseudocode has several advantages. It is compact and probably will not extend for many pages as flowcharts commonly do. Also, the plan looks like the code to be written and so is preferred by many programmers.

HIERARCHY CHART

The last programming tool we'll discuss is the **hierarchy chart**, which shows the overall program structure. Hierarchy charts are also called structure charts, HIPO (Hierarchy plus Input- Process-Output) charts, top-down charts, or VTOC (Visual Table of Contents) charts. All these names refer to planning diagrams that are similar to a company's organization chart.

Hierarchy charts depict the organization of a program but omit the specific processing logic. They describe what each part, or **module**, of the program does and they show how the modules relate to each other. The details on how the modules work, however, are omitted.

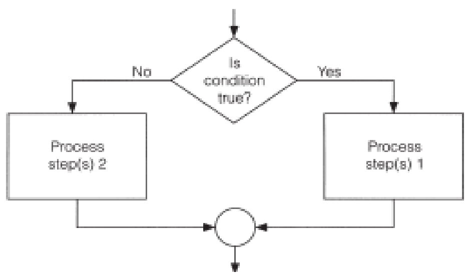
The chart is read from top to bottom and from left to right. Each module may be subdivided into a succession of sub modules that branch out under it. Typically, after the activities in the succession of sub modules are carried out, the module to the right of the original module is considered. A quick glance at the hierarchy chart reveals each task performed in the program and where it is performed.



The main benefit of hierarchy charts is in the initial planning of a program. We break down the major parts of a program so we can see what must be done in general. From this point, we can then refine each module into more detailed plans using flowcharts or pseudocode. This process is called the **divide-and-conquer** method.

The postage stamp problem was solved by a series of instructions to read data, perform calculations, and display results. Each step was in a sequence; that is, we moved from one line to the next without skipping over any lines. This kind of structure is called a **sequence structure**. Many problems, however, require a decision to determine whether a series of instructions should be executed. If the answer to a question is "Yes," then one group of instructions is executed. If the answer is "No," then another is executed. This structure is called a **decision structure**.

```
If condition is true Then
    Process step(s) 1
  Else
    Process step(s) 2
End If
```



CLASS AVERAGE ALGORITHM

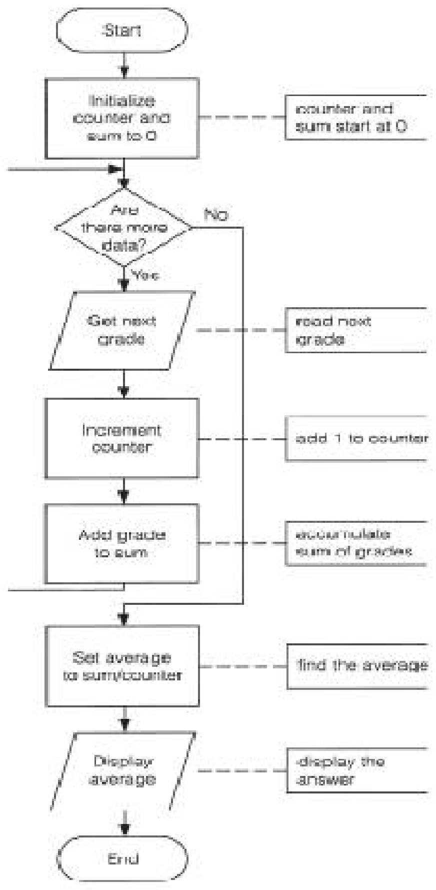
Problem: Calculate and report the grade-point average for a class.

Discussion: The average grade equals the sum of all grades divided by the number of students. We need a loop to read and then add (accumulate) the grades for each student in the class. Inside the loop, we also need to total (count) the number of students in the class.

Input: Student grades

Processing: Find the sum of the grades; count the number of students; calculate average grade
= sum of grades / number of students.

Output: Average grade



Review & Self Assessment Question

Q1- What is program Development Life cycle?
Q2-What do you mean by program Planning?
Q3-What is Flow Chart?
Q4-What is Pseudocode?
Q5-What is Hierarchical Charts?

Further Readings

Visual basic by David I Schneider
Visual basic by Steven Holzner
Visual basic by Bill Sheldon
Visual basic by Billy Holls
Visual basic by Rob Windsor

UNIT: 2-INTRODUCTION TO VISUAL BASIC

Contents

Introduction

Visual Basic is a much-enhanced version of the BASIC programming language and the BASIC Integrated Development Environment (IDE). The bottom line of the enhancement is the VB can create Windows programs whereas BASIC could only create DOS programs. Ok, so the modifications are **very** major, but the idea holds true that Visual Basic is BASIC for Windows.

Visual basic is a high level programming language developed from the earlier DOS version called BASIC. Though, Visual Basic .NET is the latest technology introduced by Microsoft with tons of new features including the .NET framework and educational institues, Universities and Software Development companies have migrated to VB .NET, Visual Basic 6 is still widely learned and taught.

Learning Visual Basic 6 is quite easier than other programming languages such as C++, C#, Java etc. This is because Visual Basic enables you to work in a graphical user interface where you can just drag and drop controls that you want to work with where you have to write bunches of code to create in C++ or C# or even in Java. If you are new to programming and want to start it in the smoothest and easiest way, then you should start it with Visual Basic.

This Visual Basic is for anyone who wants to learn to program his or her computer with Visual Basic. More importantly, this is for anyone who’s flipped through other programming resources only to be discouraged by obtuse language, jargon-ridden prose, and stuffed-shirt attitudes. The Visual Basic (VB6) beginner's conversational style incorporates plain-English explanations along with short programming examples to lead the novice programmer by the hand through the techno jungle of computer programming.

Visual Basic is initiated by using the Programs option
➔Microsoft Visual Basic 6.0
➔Visual Basic 6.0. Clicking the Visual Basic icon, we can view a copyright screen enlisting the details of the license holder of the copy of Visual Basic 6.0. Then it opens in to a new screen as shown in figure 1 below, with the interface elements Such as MenuBar, ToolBar, The New Project dialog box. These elements permit the user to build different types of Visual Basic applications.

Critical Visual Basic Elements

Although Visual Basic has grown into a fairly complex programming tool, it is still the case that a programmer can pretty much ignore all of the capabilities he doesn't need (or understand) and still create very useful applications.
But, no matter how much a beginner, or how advanced you are, there are still some fundamental areas in which you must be proficient to become a VB programmer.

When you start VB, you will see a group of windows that are known as the VB IDE (integrated development environment). As a programmer you will spend the majority of you time here, so you might as well get used to the IDE and spend a fair amount of time exploring the menu options that the IDE provides. Pay closed attention to the keyboard shortcuts that are available. If you've read my other Beginner sections you'll know that I am a **big** fan of the keyboard and have argued that the #1 productivity tool you have is good typing skills. While in the IDE, you'll find the keyboard shortcuts invaluable in writing your program quickly.

The second aspect of VB which programmers of all skill levels will have in common is the VB language itself. Most of the questions I get through email are focused on "how do I ..." and the answer is almost always couched in terms of the code that it takes to perform the task. If ever there was less boring reading than the VB language reference manual, I don't know what it is, but it also provides one of the biggest payoffs of studying that I can recommend. At least 90% of every question I answer for visitors to my site **is in the manual!**

Every Visual Basic application will consist of controls, usually a lot of them! In my opinion, the availability of controls (built-in, or controls you can purchase) is the single biggest reason why VB has reached the level of popularity that it currently enjoys. Because controls represent hundreds (if not thousands) of hours of manpower to come up with full-featured, debugged code which you can reuse in your program, controls are easily the most cost-effective, and the most time-effective way that a VB programmer has to add features to his program. Bottom line is that any good programmer must be an expert at handling controls. In my experience, once you've mastered the intrinsic controls, learning new controls becomes very straightforward.

Because VB has moved to the event-driven model of programming, the last critical VB topic I will mention is that of events. Events are not very complicated but the concept is significantly different than the old-style linear programming of the original BASIC. Simply put, when a VB program is started it sits and waits for an event to occur. The event can be a key press by the user or the movement of a mouse. Either way, the VB programming model is that your program will only react to events. When an event occurs, VB will execute the code associated with that event. So your job as a programmer is to basically create the code which your program executes in response to those events.

Programming Basics

To toss in a bit of philosophy, I view a programmer's job as getting to the end product with a minimum of effort. When you're paid by the hour you have an obligation to your employer (even if it's yourself) to minimize the cost/schedule of completing the assignment. To that end, I will emphasize over and over again the importance of using existing capabilities (such as your own reusable code, VB-provided controls, or even controls that you purchase).

In light of this optimize-your-time philosophy, I also believe that every VB programmer should strive to become an expert in the following two areas: Databases and Reporting. It's very common for applications to store data and to provide that data in a printed format (by some estimates, 80% of all VB applications use databases). And while this tutorial will discuss various ways to perform these tasks, most programmers will find that the capabilities of VB for creating/editing Access (a Microsoft product) databases and VB's capabilities for reporting that data (as exemplified by the Crystal Reports control or the newer built-in VB reporting features) are the most effective tools you can use for increasing your effectiveness in creating VB applications.

If you want to become a serious VB programmer, you **must** become an expert in these areas. I'm not saying that Microsoft's built-in tools are the only, or best, tools available for creating/editing databases and reporting on them. What I am saying is that tools which provide similar functions will be one of the most commonly used tools of those available to a programmer and that any serious programmer must take the time to develop strong skills with these tools.

One by-product programming strategy of my philosophy is that new lies should not get too anxious to jump into the more advanced features of VB. Take the time to learn the basics, and especially how to apply them with ingenuity. Once you've exhausted the potential of the fundamentals is the time to consider more advanced techniques. In my experience, over 90% of my programs consist of fundamental programming techniques, while only rarely am I compelled to dip into the more complex features that VB has to offer.

Visual Basic History (VB3, VB4, VB5 and VB6)

Starting with VB5, Visual Basic became an exclusively 32-bit programming language, suitable for programming only Win9X or NT systems. If you must program for Win 3.x, then you'll have to drop back to either VB3 or VB4, both of which are in pretty short supply. VB4 had the dual ability to support Win3.x as well as Win9X/NT systems but my personal recommendation is that if you need

32-bit system support, go straight to VB6 and if 16-bit is your need then stick with VB3.

The VB Learning Edition is the most affordable, and truth is that you can do a lot with it, particularly if you use the Windows API to augment its capabilities. However, in light of its better database features and its greater variety of controls, I suggest you go straight to the Professional Edition if at all possible. The price is steep, but it really does pay itself back in terms of time savings. If you need the VB Enterprise edition then you should have it paid for by the "Enterprise" which requires it. Individual programmers generally do not need the Enterprise edition.

The Integrated Development Environment

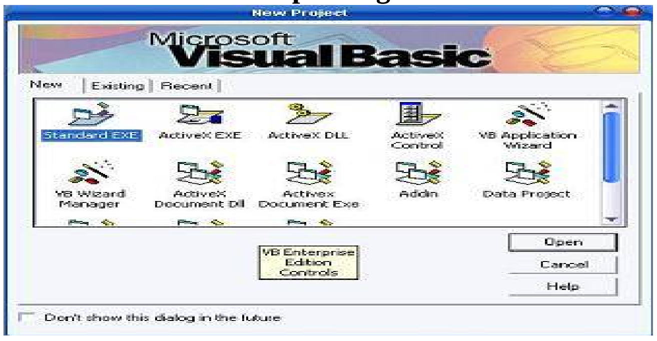
One of the most significant changes in Visual Basic 6.0 is the Integrated Development Environment (IDE). IDE is a term commonly used in the programming world to describe the interface and environment that we use to create our applications. It is called integrated because we can access virtually all of the development tools that we need from one screen called an interface. The IDE is also commonly referred to as the design environment, or the program.

The Visual Basic IDE is made up of a number of components

- Menu Bar
- Tool Bar
- Project Explorer
- Properties window
- Form Layout Window
- Toolbox
- Form Designer
- Object Browser

In previous versions of Visual Basic, the IDE was designed as a Single Document Interface (SDI). In a Single Document Interface, each window is a free-floating window that is contained within a main window and can move anywhere on the screen as long as Visual Basic is the current application. But, in Visual Basic 6.0, the IDE is in a Multiple Document Interface (MDI) format. In this format, the windows associated with the project will stay within a single container known as the parent. Code and form-based windows will stay within the main container form.

Figure 1-The Visual Basic startup dialog box



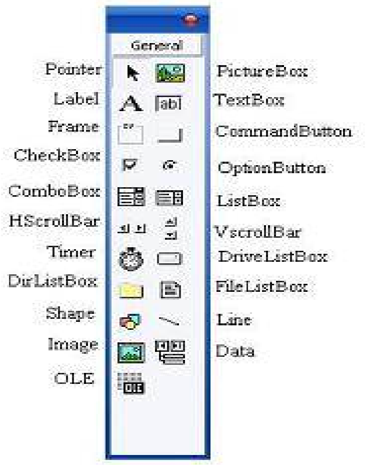
Menu Bar

This Menu Bar displays the commands that are required to build an application. The main menu items have sub menu items that can be chosen when needed. The toolbars in the menu bar provide quick access to the commonly used commands and a button in the toolbar is clicked once to carry out the action represented by it.

Toolbox

The Toolbox contains a set of controls that are used to place on a Form at design time thereby creating the user interface area. Additional controls can be included in the toolbox by using the Components menu item on the Project menu. A Toolbox is represented in figure 2 shown below.

Figure 2 Toolbox windows with its controls available commonly.



Control	Description
Pointer	Provides a way to move and resize the controls form
PictureBox	Displays icons/bitmaps and metafiles. It displays text or acts as a visual container for other controls.
TextBox	Used to display message and enter text.
Frame	Serves as a visual and functional container for controls
CommandButton	Used to carry out the specified action when the user chooses it.
CheckBox	Displays a True/False or Yes/No option.

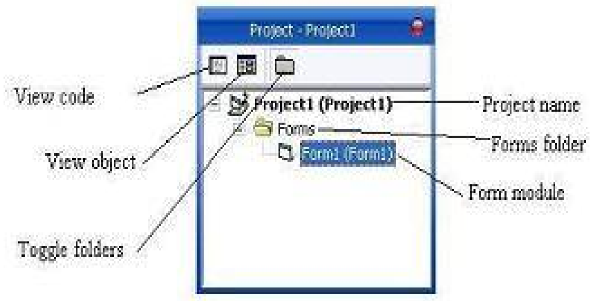
OptionButton	OptionButton control which is a part of an option group allows the user to select only one option even it displays mulitiple choices.
ListBox	Displays a list of items from which a user can select one.
ComboBox	Contains a TextBox and a ListBox. This allows the user to select an ietm from the dropdown ListBox, or to type in a
HScrollBar and VScrollBar	These controls allow the user to select a value within the specified range of values
Timer	Executes the timer events at specified intervals of time
DriveListBox	Displays the valid disk drives and allows the user to select one of them.
DirListBox	Allows the user to select the directories and paths, which are displayed.
FileListBox	Displays a set of files from which a user can select the desired one.
Shape	Used to add shape (rectangle, square or circle) to a Form
Line	Used to draw straight line to the Form
Image	used to display images such as icons, bitmaps and metafiles. But less
Data	Enables the use to connect to an existing database and display information from it.
OLE	Used to link or embed an object, display and manipulate data from other windows based applications.

Label	Displays a text that the user cannot modify or interact with.

Project Explorer

Docked on the right side of the screen, just under the tollbar, is the Project Explorer window. The Project Explorer as shown in in figure servres as a quick reference to the various elements of a project namely form, classes and modules. All of the object that make up the application are packed in a project. A simple project will typically contain one form, which is a window that is designed as part of a program's interface. It is possible to develop any number of forms for use in a program, although a program may consist of a single form. In addition to forms, the Project Explorer window also lists code modules and classes.

Figure 3 Project Explorer



Properties Window

The Properties Window is docked under the Project Explorer window. The Properties Window exposes the various characteristics of selected objects. Each and every form in an application is considered an object. Now, each object in Visual Basic has characteristics such as color and size. Other characteristics affect not just the appearance of the object but the way it behaves too. All these characteristics of an object are called its properties. Thus, a form has properties and any controls placed on it will have properties too. All of these properties are displayed in the Properties Window.

Object Browser

The Object Browser allows us to browse through the various properties, events and methods that are made available to us. It is accessed by selecting Object Browser from the View menu or pressing the key F2. The left column of the Object Browser lists the objects and classes that are available in the projects that are opened and the controls that have been referenced in them. It is possible for us to scroll through the list and select the object or class that we wish to inspect. After an object is picked up from the Classes list, we can see its members (properties, methods and events) in the right column.

A property is represented by a small icon that has a hand holding a piece of paper. Methods are denoted by little green blocks, while events are denoted by yellow lightning bolt icon.

Object naming conversions of controls (prefix)

Form	-frm
Label	-lbl
TextBox	-txt
CommandButton	-cmd
CheckBox	-chk
OptionButton	-opt
ComboBox	-cbo
ListBox	-lst
Frame	-fme
PictureBox	-pic
Image	-img
Shape	-shp
Line	-lin
HScrollBar	-hsb
VScrollBar	-vsb

All the controls in the ToolBox except the Pointer are objects in Visual Basic. These objects have associated properties, methods and events.

Real world objects are loaded with properties. For example, a flower is loaded certain color, shape and fragrance. Similarly programming objects are loaded with properties. A property is a named attribute of a programming object. Properties define the characteristics of an object such as Size, Color etc. or sometimes the way in which it behaves.

For example, a TextBox accepts properties such as Enabled, Font, MultiLine, Text, Visible, Width, etc.

- Enables property allows the TextBox to be enabled or disabled at run time depending on the condition set to True or False.
- Font property sets a particular font in the TextBox.
- MultiLine property allows the TextBox to accept and display multiple lines at run time.
- Text property of the TextBox control sets a particular text in the control.
- Visible property is used to hide the object at run time.
- Width property sets the TextBox to the desired width at design time.

The properties that are discussed above are design-time properties that can be set at the design time by selecting the Properties Window. But certain properties cannot be set at design time. For example, the Current X and Current Y properties of a Form cannot be set at the design time.

A method is an action that can be performed on objects. For example, a cat is an object. Its properties might include long white hair, blue eyes, 3 pounds weight etc. A complete definition of cat must only encompass on its looks, but should also include a complete itemization of its activities. Therefore, a cat's methods might be move, jump, play, breath etc.

Similarly in object-oriented programming, a method is a connected or built-in procedure, a block of code that can be invoked to impart some action on a particular object. A method requires an object to provide them with a context. For example, the word Move has no meaning in Visual Basic, but the statement Text1.Move 700, 400 performs a very precise action. The TextBox control has other associated methods such as Refresh, SetFocus, etc.

- The Refresh method enforces a complete repaint of the control or a Form.

  For example, Text1.Refresh refreshes the TextBox.
- The Setfocus method moves the focus on the control.

For Example Text1.SetFocus sets the focus to TextBox control Text1.

Evolution of Visual Basic

- VB 1.0 was introduced in 1991. The approach for connecting the programming language to the graphical user interface is derived from a system called Tripod (sometimes also known as Ruby), originally developed by Alan Cooper, which was further developed by Cooper and his associates under contract to Microsoft.

Timeline of Visual Basic

- Visual Basic 1.0 (May 1991) was released for Windows.
- Visual Basic 1.0 for DOS was released in September 1992. The language itself was not quite compatible with Visual Basic for Windows, as it was actually the next version of Microsoft's DOS-based BASIC compilers, Microsoft QuickBASIC compiler
- QuickBASIC and BASIC Professional Development System. The interface was barely graphical, using extended ASCII characters to simulate the appearance of a GUI.
- Visual Basic 2.0 was released in November 1992. The programming environment was easier to use, and its speed was improved.
- Visual Basic 3.0 was released in the summer of 1993 and came in Standard and Professional versions. VB3 included a database engine that could read and write Access databases.
- Visual Basic 4.0 (August 1995) was the first version that could create 32-bit as well as 16-bit Windows programs. It also introduced the ability to write classes in Visual Basic.
- With version 5.0 (February 1997), Microsoft released Visual Basic exclusively for 32-bit versions of Windows. Programmers who preferred to write 16-bit programs were able to import programs written in Visual Basic 4.0 to Visual Basic 5.0, and Visual Basic 5.0 programs can easily be converted with Visual Basic 4.0. Visual Basic 5.0 also introduced the ability to create custom user controls,

as well as the ability to compile to native Windows executable code, speeding up runtime code execution.

- Visual Basic 6.0 (Mid 1998) improved in a number of areas, including the ability to create web-based applications using Internet Explorer. Visual Basic 6 is no longer supported.

Event Driven Programming

Visual Basic programs are built around events. Events are various things that can happen in a program. This will become clearer when studied in contrast to procedural programming. In procedural languages, an application is written is executed by checking for the program logically through the program statements, one after another. For a temporary phase, the control may be transferred to some other point in a program. While in an event driven application, the program statements are executed only when a particular event calls a specific part of the code that is assigned to the event.

Let us consider a TextBox control and a few of its associated events to understand the concept of event driven programming. The TextBox control supports various events such as Change, Click, MouseMove and many more that will be listed in the Properties dropdown list in the code window for the TextBox control. We will look into a few of them as given below.

- The code entered in the Change event fires when there is a change in the contents of the TextBox
- The Click event fires when the TextBox control is clicked.
- The MouseMove event fires when the mouse is moved over the TextBox

As explained above, several events are associated with different controls and forms, some of the events being common to most of them and few being specific to each control.

Visual Basic uses building blocks such as Variables, Data Types, Procedures, Functions and Control Structures in its programming environment. This section concentrates on the programming fundamentals of Visual Basic with the blocks specified.

Modules

Code in Visual Basic is stored in the form of modules. The three kinds of modules are Form Modules, Standard Modules and Class Modules. A simple application may contain a single Form, and the code resides in that Form module itself. As the application grows, additional Forms are added and there may be a common code to be executed in several Forms. To avoid the duplication of code, a separate module containing a procedure is created that implements the common code. This is a standard Module.

Class module (.CLS filename extension) are the foundation of the object oriented programming in Visual Basic. New objects can be created by writing code in class modules. Each module can contain:

Declarations: May include constant, type, variable and DLL procedure declarations.

Procedures: A sub function, or property procedure that contain pieces of code that can be executed as a unit.

These are the rules to follow when naming elements in VB - variables, constants, controls, procedures, and so on:

- A name must begin with a letter.
- May be as much as 255 characters long (but don't forget that somebody has to type the stuff!).
- Must not contain a space or an embedded period or type-declaration characters used to specify a data type; these are! # % $ & @
- Must not be a reserved word (that is part of the code, like Option, for example)
- The dash, although legal, should be avoided because it may be confused with the minus sign. Instead of First-name use First_name or FirstName.

Some Features of Visual Basic

- Full set of objects - you 'draw' the application
- Lots of icons and pictures for your use
- Response to mouse and keyboard actions
- Clipboard and printer access
- Full array of mathematical, string handling and graphics functions
- Can handle fixed and dynamic variable and control arrays
- Sequential and random access file support
- Useful debugger and error-handling facilities
- Powerful database access tools
- ActiveX support
- Package & Deployment Wizard makes distributing your applications simple

Visual Basic 6.0 versus Other Versions of Visual Basic

· The original Visual Basic for DOS and Visual Basic for Windows were introduced in 1991.

· Visual Basic 3.0 (a vast improvement over previous versions) was released in 1993.

· Visual Basic 4.0 released in late 1995 (added 32 bit application support).

· Visual Basic 5.0 released in late 1996. New environment, supported creation of
ActiveX controls, deleted 16 bit application support.

· And, now Visual Basic 6.0 - some identified new features of Visual Basic 6.0:

- Faster compiler
- New ActiveX data control object
- Allows database integration with wide variety of applications
- New data report designer
- New Package & Deployment Wizard
- Additional internet capabilities

16 Bits versus 32 Bits

· Applications built using the Visual Basic 3.0 and the 16 bit version of Visual Basic 4.0 will run under Windows 3.1, Windows for Workgroups, Windows NT, or Windows 95

· Applications built using the 32 bit version of Visual Basic 4.0, Visual Basic 5.0 and Visual Basic 6.0 will only run with Windows 95 or Windows NT (Version 3.5.1 or higher).

· In this class, we will use Visual Basic 6.0 under Windows 95, recognizing such applications will not operate in 16 bit environments.

Review & Self Assessment Question

Q1-What do you mean by Visual Basic Language?
Q2-Describe Integrated Development Environment?
Q3-Write about Project explorer?
Q4-What is object browser?
Q5- What is Event Driven Programming?

Further Readings

Visual basic by David I Schneider

Visual basic by Steven Holzner

Visual basic by Bill Sheldon

Visual basic by Billy Holls

Visual basic by Rob Windsor

UNIT: 3- VISUAL BASIC FUNDAMENTAL

Contents

Variable:

Variables are the memory locations which are used to store values temporarily. A defined naming strategy has to be followed while naming a variable. A variable name must begin with an alphabet letter and should not exceed 255 characters. It must be unique within the same scope. It should not contain any special character like %, &,!, #, @ or $.

There are many ways of declaring variables in Visual Basic. Depending on where the variables are declared and how they are declared, we can determine how they can be used by our application. The different ways of declaring variables in Visual Basic are listed below and elucidated in this section.

Explicit Declaration

Declaring a variable tells Visual Basic to reserve space in memory. It is not must that a variable should be declared before using it. Automatically whenever Visual Basic encounters a new variable, it assigns the default variable type and value. This is called implicit declaration. Though this type of declaration is easier for the user, to have more control over the variables, it is advisable to declare them explicitly. The variables are declared with a Dim statement to name the variable and its type. The As type clause in the Dim statement allows to define the data type or object type of the variable. This is called explicit declaration.

Syntax

```
Dim variable [As Type]
```

For example,

```
Dim strName As String
Dim intCounter As Integer
```

Using Option Explicit statement

It may be convenient to declare variables implicitly, but it can lead to errors that may not be recognized at run time. Say, for example a variable by name intcount is used implicitly and is assigned to a value. In the next step, this field is incremented by 1 by the following statement

Intcount = Intcount + 1

This calculation will result in intcount yielding a value of 1 as intcount would have been initialized to zero. This is because the intcount variable has been typed as intcont in the right hand side of the second variable. But Visual Basic does not see this as a mistake and considers it to be new variable and therefore gives a wrong result.

In Visual Basic, to prevent errors of this nature, we can declare a variable by adding the following statement to the general declaration section of the Form.

Option Explicit

This forces the user to declare all the variables. The Option Explicit statement checks in the module for usage of any undeclared variables and reports an error to the user. The user can thus rectify the error on seeing this error message.

The Option Explicit statement can be explicitly placed in the general declaration section of each module using the following steps.

- Click Options item in the Tools menu
- Click the Editor tab in the Options dialog box
- Check Require Variable Declaration option and then click the OK button

Scope of variables

A variable is scoped to a procedure-level (local) or module-level variable depending on how it is declared. The scope of a variable, procedure or object determines which part of the code in our application are aware of the variable's existence. A variable is declared in general declaration section of e Form, and hence is available to all the procedures. Local variables are recognized only in the procedure in which they are declared. They can be declared with Dim and Static keywords. If we want a variable to be available to all of the procedures within the same module, or to all the procedures in an application, a variable is declared with broader scope.

Local Variables

A local variable is one that is declared inside a procedure. This variable is only available to the code inside the procedure and can be declared using the Dim statements as given below.

Dim sum As Integer

The local variables exist as long as the procedure in which they are declared, is executing. Once a procedure is executed, the values of its local variables are lost and the memory used by these variables is freed and can be reclaimed. Variables that are declared with keyword Dim exist only as long as the procedure is being executed.

Static Variables

Static variables are not reinitialized each time Visual Invokes a procedure and therefore retains or preserves value even when a procedure ends. In case we need to keep track of the number of times a command button in an application is clicked, a static counter variable has to be declared. These static variables are also ideal for making controls alternately visible or invisible. A static variable is declared as given below.

```
Static intPermanent As Integer
```

Variables have a lifetime in addition to scope. The values in a module-level and public variables are preserved for the lifetime of an application whereas local variables declared with Dim exist only while the procedure in which they are declared is still being executed. The value of a local variable can be preserved using the Static keyword. The following procedure calculates the running total by adding new values to the previous values stored in the static variable value.

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

If the variable Accumulate was declared with Dim instead of static, the previously accumulated values would not be preserved accross calls to the procedure, and the procedure would return the same value with which it was called. To make all variables in a procedure static, the Static keyword is placed at the beginning of the procedure heading as given in the below statement.

```
Static Function RunningTotal ( )
```

Example

The following is an example of an event procedure for a CommandButton that counts and displays the number of clicks made.

```
Private Sub Command1_Click ( )
Static Counter As Integer
Counter = Counter + 1
Print Counter
End Sub
```

The first time we click the CommandButton, the Counter starts with its default value of zero. Visual Basic then adds 1 to it and prints the result.

Module Level Variables

A module level variable is available to all the procedures in the module. They are declared using the Public or the Private keyword. If you declare a variable using a Private or a Dim statement in the declaration section of a module—a standard BAS module, a form module, a class module, and so on—you're creating a private module-level variable. Such variables are visible only from within the module they belong to and can't be accessed from the outside. In general, these variables are useful for sharing data among procedures in the same module:

```
 In the declarative section of any module
Private LoginTime As Date ' A private module-level variable
Dim     Login Password As String ' Another private module-level variable
```

You can also use the Public attribute for module-level variables, for all module types except BAS modules. (Public variables in BAS modules are global variables.) In this case, you're creating a strange beast: a Public module-level variable that can be accessed by all procedures in the module to share data and that also can be accessed from outside the module. In this case, however, it's more appropriate to describe such a variable as a property:

```
In the declarative section of Form1 module
Public CustomerName As String ' A Public property
```

You can access a module property as a regular variable from inside the module and as a custom property from the outside:

```
From outside Form1 module...
Form1.CustomerName = "John Smith"
```

The lifetime of a module-level variable coincides with the lifetime of the module itself. Private variables in standard BAS modules live for the entire life of the application, even if they can be accessed only while Visual Basic is executing code in that module. Variables in form and class modules exist only when that module is loaded in memory. In other words, while a form is active (but not necessarily visible to the user) all its variables take some memory, and this memory is released only when the form is completely unloaded from memory. The next time the form is re created, Visual Basic reallocates memory for all variables and resets them to their default values.

Public vs. Local Variables

A variable can have the same name and different scope. For example, we can have a public variable named R and within a procedure we can declare a local variable R. References to the name R within the procedure would access the local variable and references to R outside the procedure would access the public variable.

Constants

Constants are named storage locations in memory, the value of which does not change during program Execution. They remain the same throughout the program execution. When the user wants to use a value that never changes, a constant can be declared and created. The Const statement is used to create a constant. Constants can be declared in local, form, module or global scope and can be public or private as for variables. Constants can be declared as illustrated below.

Public Const gravity constant As Single = 9.81

Predefined Visual Basic Constants

The predefined constants can be used anywhere in the code in place of the actual numeric values. This makes the code easier to read and write.

For example consider a statement that will set the window state of a form to be maximized.

Form1.Windowstate = 2

The same task can be performed using a Visual Basic constant

Form1.WindowState = vb Maximized

Data Type

By default Visual Basic variables are of variant data types. The variant data type can store numeric, date/time or string data. When a variable is declared, a data type is supplied for it that determines the kind of data they can store. The fundamental data types in Visual Basic including variant are integer, long, single, double, string, currency, byte and Boolean. Visual Basic supports a vast array of data types. Each data type has limits to the kind of information and the minimum and maximum values it can hold. In addition, some types can interchange with some other types. A list of Visual Basic's simple data types are given below.

1. Numeric

Byte	Store integer values in the range of 0 – 255
Integer	Store integer values in the range of (-32,768) - (+ 32,767)
Long	Store integer values in the range of (- 2,147,483,468) - (+ 2,147,483,468)
Single	Store floating point value in the range of (-3.4x10-38) - (+ 3.4x1038)
Double	Store large floating value which exceeding the single data type value

Currency	store monetary values. It supports 4 digits to the right of decimal point and 15 digits to the left

2. String
Use to store alphanumeric values. A variable length string can store approximately 4 billion characters
3. Date
Use to store date and time values. A variable declared as date type can store both date and time values and it can store date values 01/01/0100 up to 12/31/9999
4. Boolean
Boolean data types hold either a true or false value. These are not stored as numeric values and cannot be used as such. Values are internally stored as -1 (True) and 0 (False) and any non-zero value is considered as true.
5. Variant
Stores any type of data and is the default Visual Basic data type. In Visual Basic if we declare a variable without any data type by default the data type is assigned as default.

Operators in Visual Basic

Arithmetical Operators

Operators	Description	Example	Result
+	Add	5+5	10
-	Substract	10-5	5
/	Divide	25/5	5
\	Integer Division	20\3	6
*	Multiply	5*4	20
^	Exponent	3^3	27
Mod	Remainder of	20 Mod 6	2
&	String	"George"&"	"George Bush"

Relational Operators

Operators	Description	Example	Result
>	Greater than	10>8	True
<	Less than	10<8	False
>=	Greater than or	20>=10	True

	equal to		
<=	Less than or equal to	10<=20	True
<>	Not Equal to	5<>4	True
=	Equal to	5=7	False

Logical Operators

Operators	Description
OR	Operation will be true if either of the operands is true
AND	Operation will be true only if both the operands are true

Review & Self Assessment Question

Q1- What is variable? Explain with example.
Q2-What do you mean by Scope of Variable?
Q3-What is data type? Explain it.
Q4-What do you mean by Module level variable.

Further Readings

Visual basic by David I Schneider

Visual basic by Steven Holzner

Visual basic by Bill Sheldon

Visual basic by Billy Holls

Visual basic by Rob Windsor

UNIT: 4 - CONTROL STRUCTURE

Contents

Control Statement

Control Statements are used to control the flow of program's execution. Visual Basic supports control structures such as if... Then, if...Then ...Else, Select...Case, and Loop structures such as Do While...Loop, While...Wend, For...Next etc method.

If...Then selection structure

The if...Then selection structure performs an indicated action only when the condition is True; otherwise the action is skipped.

Syntax of the If...Then selection

```
If <condition> Then
statement
End If
```

Example

```
 If average>75 Then
txtGrade.Text = "A"
End If
```

If...Then...Else selection structure

The If...Then...Else selection structure allows the programmer to specify that a different action is to be performed when the condition is True than when the condition is False.

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

```
If <condition > Then
statements
Else
statements
End If
```

Example :

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

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

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

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

You can use Nested If either of the methods as shown above

Method 1

```
If < condition 1 > Then
statements
ElseIf < condition 2 > Then
statements
ElseIf < condition 3 > Then
statements
Else
Statements
End If
```

Method 2

```
If < condition 1 > Then
statements
Else
If < condition 2 > Then
statements
Else
If < condition 3 > Then
statements
Else
Statements
End If
End If
EndIf
```

Example

Assume you have to find the grade using nested if and display in a text box

```
If average > 75 Then
txtGrade.Text = "A"
ElseIf average > 65 Then
txtGrade.Text = "B"
ElseIf average > 55 Then
txtGrade.text = "C"
ElseIf average > 45 Then
txtGrade.Text = "S"
Else
txtGrade.Text = "F"
End If
```

Select...Case selection structure

Select...Case structure is an alternative to If...Then...ElseIf for selectively executing a single block of statements from among multiple block of statements. Select...case is more convenient to use than the If...Else...End If. The following program block illustrate the working of Select...Case.

Syntax of the Select...Case selection structure

```
Select Case Index
Case 0
Statements
Case 1
Statements
End Select
```

Example:

Assume you have to find the grade using select...case and display in the text box

```
Dim average as Integer
average = txtAverage.Text
Select Case average
Case 100 To 75
txtGrade.Text ="A"
Case 74 To 65
txtGrade.Text ="B"
Case 64 To 55
txtGrade.Text ="C"
Case 54 To 45
txtGrade.Text ="S"
Case 44 To 0
txtGrade.Text ="F"
Case Else
MsgBox "Invalid average marks"
End Select
```

Looping Statement

A repetition structure allows the programmer to that an action is to be repeated until given condition is true.

Do While... Loop Statement

The **Do While...Loop** is used to execute statements until a certain condition is met.

Example: The following Do Loop counts from 1 to 100.

```
Dim number As Integer
number = 1
Do While number <= 100
number = number + 1
Loop
```

A variable number is initialized to 1 and then the Do While Loop starts. First, the condition is tested; if condition is True, then the statements are executed. When it gets to the Loop it goes back to the Do and tests condition again. If condition is False on the first pass, the statements are never executed.

While... Wend Statement

A **While...Wend** statement behaves like the **Do While...Loop** statement.
Example: The following **While...Wend** counts from 1 to 100

```
Dim number As Integer
number = 1
While number <=100
number = number + 1
Wend
```

Do...Loop While Statement

The **Do...Loop** While statement first executes the statements and then test the condition after each execution.
Example: The following program block illustrates the structure:

```
Dim number As Long
number = 0
Do
number = number + 1
Loop While number < 201
```

The programs execute the statements between Do and Loop While structure in any case. Then it determines whether the counter is less than 501. If so, the program again executes the statements between Do and Loop While else exits the Loop.

Do Until...Loop Statement

Unlike the **Do While...Loop** and **While...Wend** repetition structures, the **Do Until... Loop** structure tests a condition for falsity. Statements in the body of a **Do Until...Loop** are executed repeatedly as long as the loop-continuation test evaluates to False.
An example for **Do Until...Loop** statement. The coding is typed inside the click event of the command button

```
Dim number As Long
number=0
Do Until number > 1000
number = number + 1
Print number
Loop
```

Numbers between 1 to 1000 will be displayed on the form as soon as you click on the command button.

The For...Next Loop

The **For...Next** Loop is another way to make loops in Visual Basic. **For...Next** repetition structure handles all the details of counter-controlled repetition.

Example: The following loop counts the numbers from 1 to 100:

```
Dim x As Integer
For x = 1 To 50
```

```
Print x
Next
```

In order to count the numbers from 1 to 50 in steps of 2, the following loop can be used

```
For x = 1 To 50 Step 2
Print x
Next
```

The following loop counts numbers as 1, 3, 5, 7..etc

The above coding will display numbers vertically on the form. In order to display numbers horizontally the following method can be used.

```
For x = 1 To 50
Print x & Space$ (2);
Next
```

To increase the space between the numbers increase the value inside the brackets after the & Space$.

Following example is a **For...Next** repetition structure which is with the If condition used.

```
Dim number As Integer
For number = 1 To 10
If number = 4 Then
Print "This is number 4"
Else
Print number
End If
Next
```

In the output instead of number 4 you will get the "This is number 4".

A **For...Next** loop condition can be terminated by an **Exit For** statement. Consider the following statement block.

```
Dim x As Integer
For x = 1 To 10
Print x
If x = 5 Then
Print "The program exited at x=5"
Exit For
End If
Next
```

The preceding code increments the value of x by 1 until it reaches the condition x = 5. The **Exit For** statement is executed and it terminates the **For...Next** loop. The Following statement blocks containing **Do...While** loop is terminated using **Exit Do** statement.

```
Dim x As Integer
Do While x < 10
Print x
x = x + 1
If x = 5 Then
Print "The program is exited at x=5"
```

```
        Exit Do
        End If
        Loop
```

With...End With statement

When properties are set for objects or methods are called, a lot of coding is included that acts on the same object. It is easier to read the code by implementing the **With...End With** statement. Multiple properties can be set and multiple methods can be called by using the **With...End With** statement. The code is executed more quickly and efficiently as the object is evaluated only once. The concept can be clearly understood with following example.

```
With Text1
.Font.Size = 14
.Font.Bold = True
.ForeColor = vbRed
.Height = 230
.Text = "Hello World"
End With
```

In the above coding, the object Text1, which is a text box is evaluated only once instead of every associated property or method. This makes the coding simpler and efficient.

Review & Self Assessment Question

Q1-What is control statement?
Q2-What is while –wend statement? Explain with example?
Q3-Explain the For-Next loop? With example.
Q4-What are the differences between Entry Controlled loop and Exit Controlled loop.

Further Readings

Visual basic by David I Schneider
Visual basic by Steven Holzner
Visual basic by Bill Sheldon
Visual basic by Billy Holls
Visual basic by Rob Windsor

UNIT: 5 - WORKING WITH VISUAL BASIC CONTROL

Contents

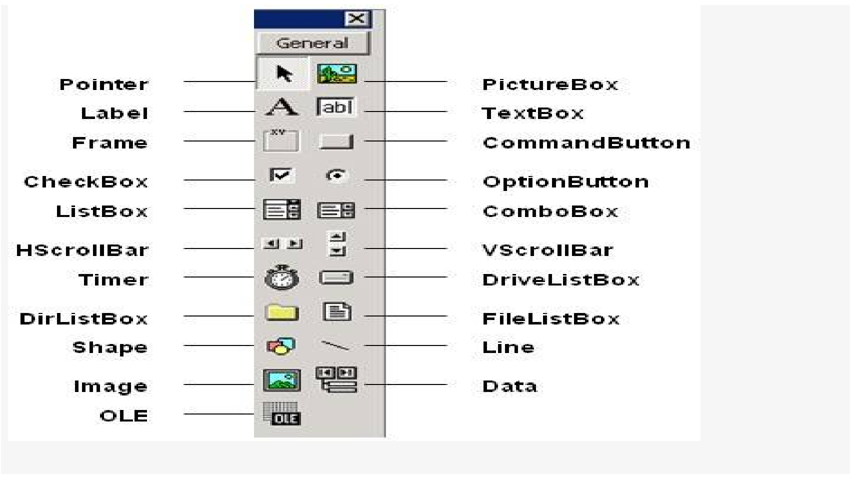
Control

A control is an object that can be drawn on a Form object to enable or enhance user interaction with an application. Controls have properties that define aspects their appearance, such as position, size and colour, and aspects of their behavior, such as their response to the user input.

There are three categories of controls in Visual Basic 6:

1. **Intrinsic Controls:** The IDE buttons and frame controls. They are inside the Visual Basic .exe file. They are always in the toolbox.
2. **ActiveX Controls**: The ActiveX control exist as separate files with a .ocx file name extension. These include controls that are available in all editions of Visual Basic and those that are available only in the Professional and Enterprise editions (such as Listview, Toolbar, Animation, and Tabbed Dialog).
3. **Insertable Objects** (like the Excel Worksheet object; with a list of the employees, or a Project Calendar object having the scheduling information for a project.) These can be added to the toolbox, so they can be considered controls.

1. **Intrinsic Controls:** The IDE buttons and frame controls. They are inside the Visual Basic .exe file. They are always in the toolbox. Like this

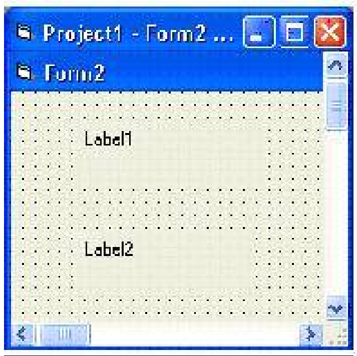


Label:

Labels are one of the most commonly used controls in Visual Basic. They are also the simplest to use. Label is used to display title, some text on the screen. This is placed against the other controls such as TextBox, which is used to accept some input from the user. It provides the message what data to enter. Labels are drawn with a black font by default. However, you can set the label's text color to match your application's color scheme.

1. Create a new project
2. Add two Labels from toolbox on the form.

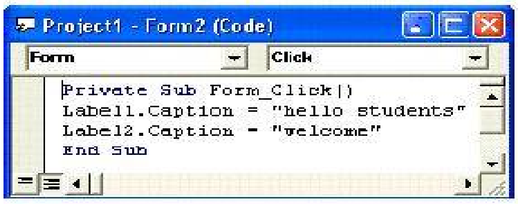
DESIGNING OF LABEL:-



3. Double click on Form to open code window and type the following code

```
Private Sub Form_Click()
Label1.Caption = “hello students”
Label2.Caption = “welcome”
End Sub
```

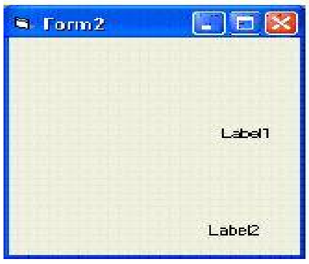
CODING OF LABEL:-



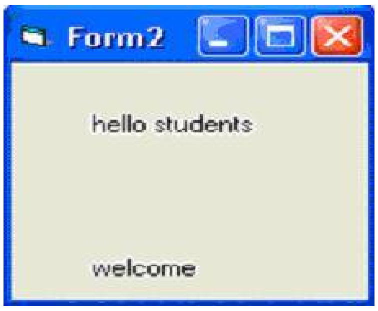
```
Private Sub Form_Click()
Label1.Caption = "hello students"
Label2.Caption = "welcome"
End Sub
```

4. Use the File menu to save your work as Main.frm and (your intitials) Project1.vbp
5. Press F5 to run

OUTPUT OF LABEL:



5. Click on the Form then

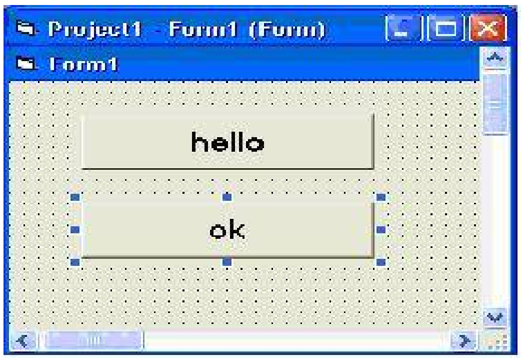


Command Button

Command button control is used to get the user's response depending on which some specific action can taken. Command buttons determine when the user wants to do something such as exit the application or begin printing. In almost every case, you will perform these tasks to add a command button to an application For example you can see the "OK" button when you click on "OK" button an event is fire and a msg box is open.

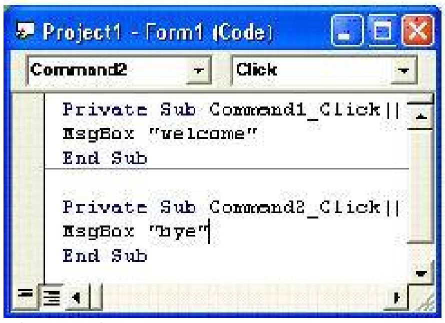
1. **Open Visual Basic 6**.0 and create a new**Standard EXE Project**.
2. From **Standard EXE project** Dialogbox choose **Standard EXE form.**
3. Add Two **Command Button** from **toolbox** on the form.
4. From the **properties window** change the caption of the **Command Buttons**

DESIGNING OF COMMMAND BUTTON:



5. After design the form open the **code window** and write code:

CODING OF COMMMAND BUTTON:

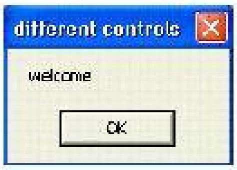


```
Private Sub Command1_Click()
MsgBox "welcome"
End Sub

Private Sub Command2_Click()
MsgBox "bye"
End Sub
```

6. Then press f5 key for output

7. When you click Command Button1 whose caption is “hello” then a msgbox show like below:

OUTPUT OF COMMMAND BUTTON:



Text Box

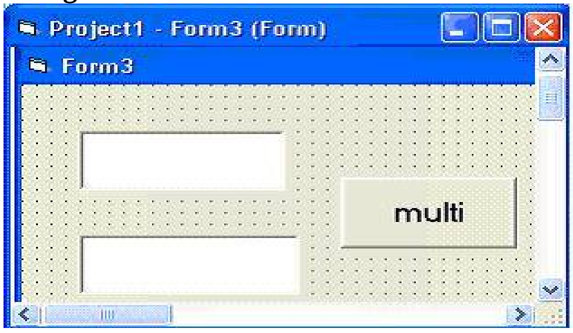
Text Box control is used to accept some data from the user. A text box’s purpose is to allow the user to input text information to be used by the program. User can type in the textbox that value can be accessed in any procedure by referring to the text property of the text box control.A typical text box is a rectangle of any size Text boxes usually display a text cursor. Text Box controls, which have a great many properties and events, are also among the most complex intrinsic controls.

1. Open Visual Basic 6**.0 and create a new**Standard EXE Project**.**

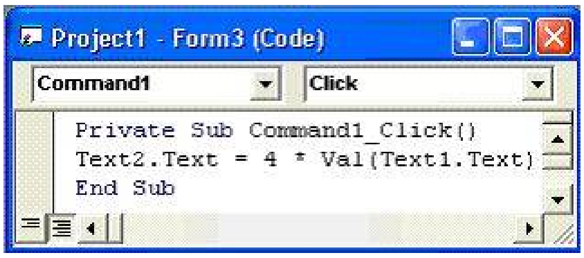
2. From Standard EXE project **Dialogbox choose** Standard EXE form.

3. Add two Text Box **from** toolbox **on the form.**

4. Add one Command Button **from** toolbox **on the form.**
5. Change the caption **of** Command Button **from** Properties Window
6. When you design the form it is look like below

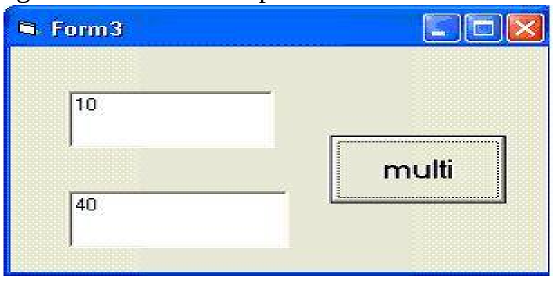


7 Then open the **code window** and write code :



```
Private Sub Command1_Click()
Text2.Text = 4 * Val(Text1.Text)
End Sub
```

9. Use the File menu to save your work as Main.frm and (your intitials) Project1.vbp
8. After coding press <F5> function key to run the application.
9. Enter the digit in **TextBox1** and press Command Button.

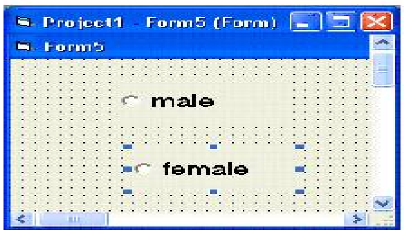


Radio Button /Option Button

A Radio Button is also known as Option button. An Option button is a round radio button that you select to turn an option on or off. This button is used where you have only one option to choose like yes or no, male or female and married or unmarried.

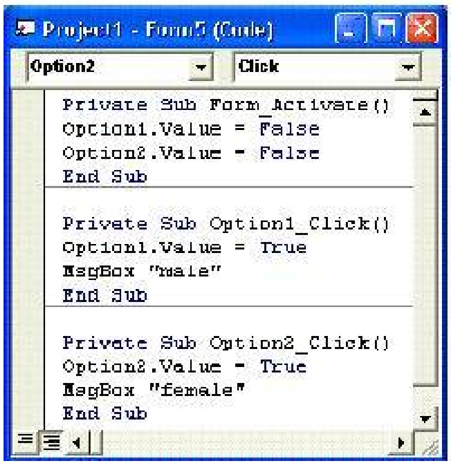
1. **Open Visual Basic 6**.0 and create a new**Standard EXE Project**.
2. From **Standard EXE project** Dialogbox choose **Standard EXE form.**
3. Add two **Radio Button** from **toolbox** on the form.
4. Use the **properties window** to change the caption of Radio Button “male” and “female”.

DESIGNING OF OPTION BUTTON:-



5. Double click on **Radio button** and enter the following code

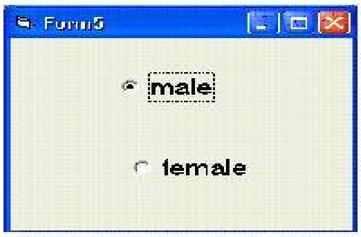
CODING OF OPTION BUTTON:-



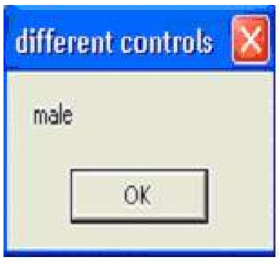
```
Private Sub Form_Activate()
Option1.Value = False
Option2.Value = False
End Sub

Private Sub Option1_Click()
Option1.Value = True
MsgBox "male"
End Sub

Private Sub Option2_Click()
Option2.Value = True
MsgBox "female"
End Sub
```

6.Press <F5> to run.

OUTPUT OF OPTION BUTTON:-



7.When you click on male button then

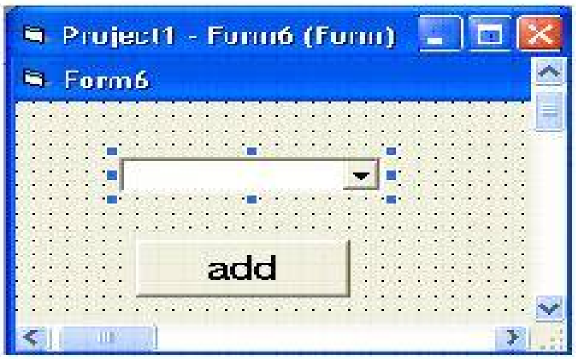


ComboBox

The **ComboBox** is used to display a list incompact form. And so this control consume less space and still can have all the elements of list in it. List elements can be added both at design time and at run time.

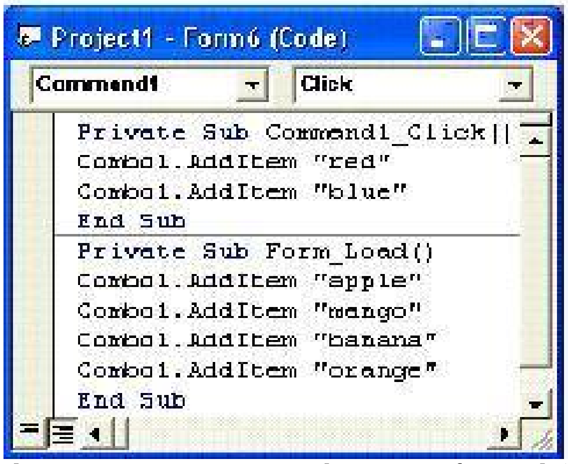
1. **Open Visual Basic 6**.0 and create a new**Standard EXE Project**.
2. From **Standard EXE project** Dialogbox choose **Standard EXE form.**
3. Add one **Combo Box** and **Command Button** from **toolbox** on the form.
4. Change the caption of command button.

DESIGNING OF COMBO BOX:-



5. Then add code in code window. Following is an example to add item to a combo box. The code is typed in the Form_Load event.

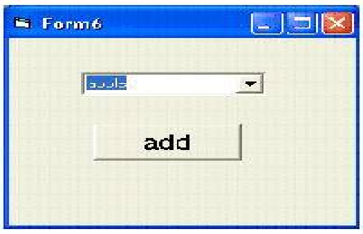
CODING OF COMBO BOX:-



```
Private Sub Command1_Click()
Combo1.AddItem "red"
Combo1.AddItem "blue"
End Sub
Private Sub Form_Load()
Combo1.AddItem "apple"
Combo1.AddItem "mango"
Combo1.AddItem "banana"
Combo1.AddItem "orange"
End Sub
```

6. Use the File menu to save your work as Main.frm and (your intitials) Project1.vbp
7.Use <f5> key to run the program.

OUTPUT OF COMBO BOX :-

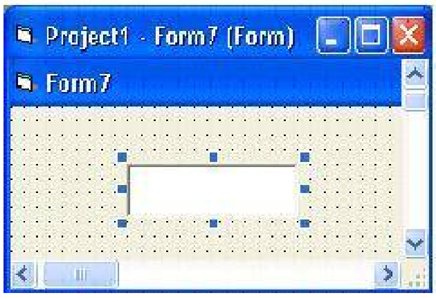


List Box

The List control is used to present a list of items where the user can click and select the items from the list. In order to add items to the list, we can use the AddItem method.

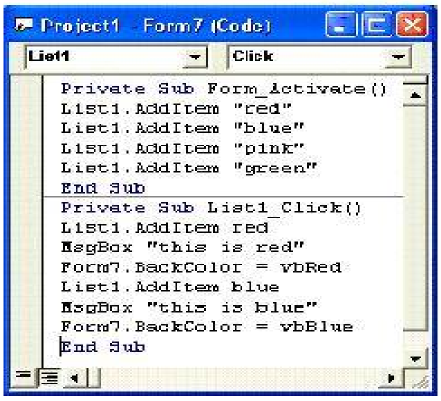
1. Start Visual Basic and pick “Standard Exe” from the list of new projects.
2. Go to the properties page (on the right) and find the **caption** property for the main form (Form1).
3. Add a List control to the form by clicking the List tool and then drawing on the form.

DESIGNING OF LIST BOX :-



4. Double click the ListBox . This will bring up the code-view Now, double window. Type in the following code into the List1_Click() subroutine shell that’s generated for you:

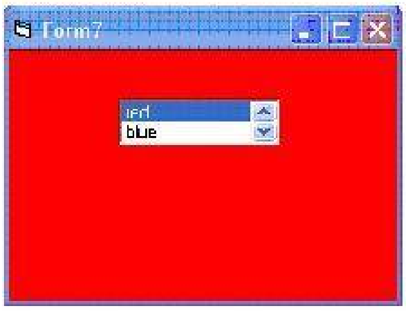
CODING OF LIST BOX :-



```
Private Sub Form_Activate()
List1.AddItem "red"
List1.AddItem "blue"
List1.AddItem "pink"
List1.AddItem "green"
End Sub
Private Sub List1_Click()
List1.AddItem red
MsgBox "this is red"
Form7.BackColor = vbRed
List1.AddItem blue
MsgBox "this is blue"
Form7.BackColor = vbBlue
End Sub
```

5. At this point, press <F5> key to run the program and select the option to see what happens.

OUTPUT OF LIST BOX :-



6. When you select red option from Listbox then form's colour is change and shows a message

Program of Window State In Visual Basic

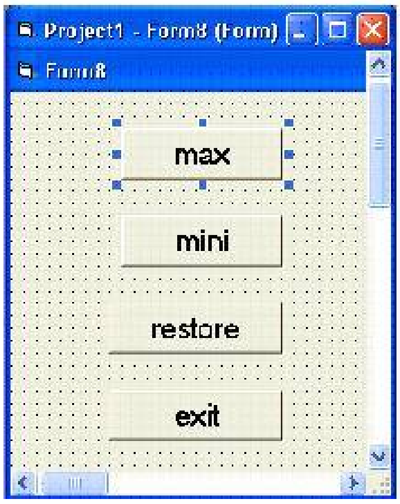
1. Start Visual Basic and pick **"Standard Exe"** from the list of new projects.

2. Go to the **properties window** (on the right) and find the **caption** property for the main form (Form1).

3. Add four Button to the form by clicking the by clicking the tool and then drawing on the form.

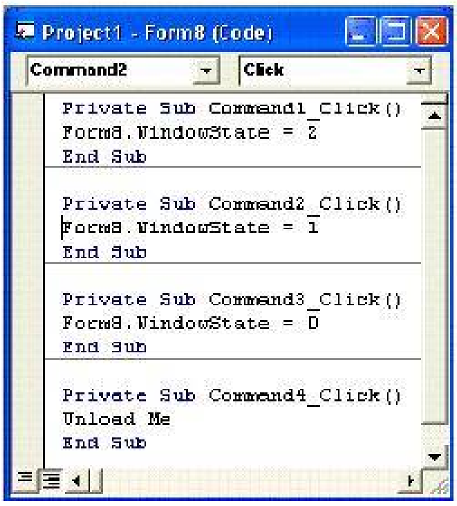
4. Go to the **properties window** and change the caption property for max(Button1),mini(Button2),restore(Button3),exit(Button4).

DESIGNING OF COMMAND BUTTTON :-



4. Double click the max button . This will bring up the code-view Now, double window. Type in the following code into the**Command1_Click()**subroutine shell that's generated for you:

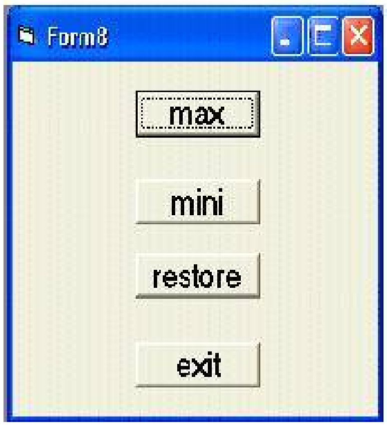
CODING OF COMMAND BUTTTON :-



```
Private Sub Command1_Click()
Form8.WindowState = 2
End Sub

Private Sub Command2_Click()
Form8.WindowState = 1
End Sub

Private Sub Command3_Click()
Form8.WindowState = 0
End Sub

Private Sub Command4_Click()
Unload Me
End Sub
```

5. At this point, press <f5> key to run the program and select the option to see what happens

OUTPUT OF COMMAND BUTTTON :-



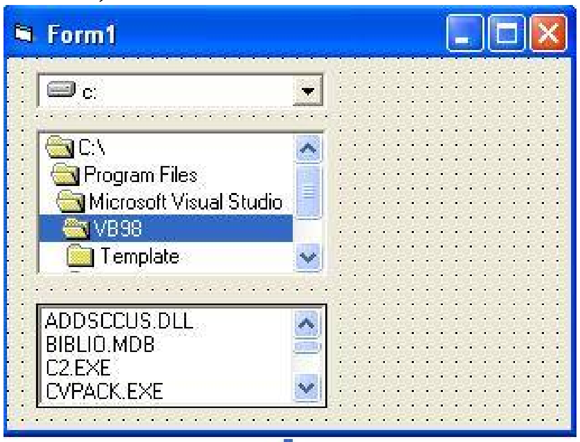
6. Click on max Button.

DriveListBox, DirListBox and FileListBox controls

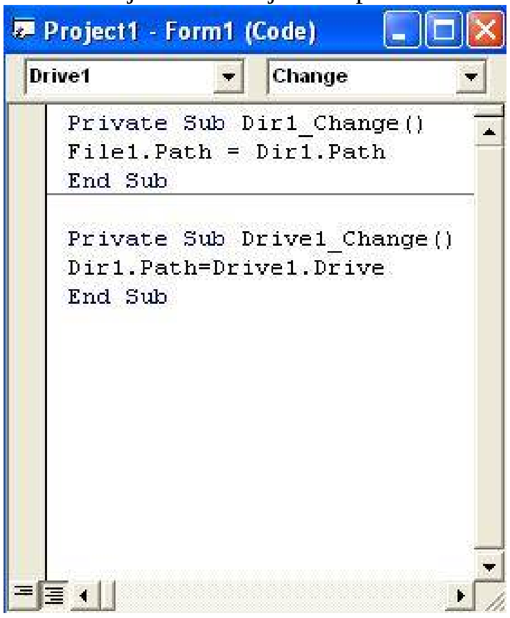
Three of the controls on the ToolBox let you access the computer's file system. They are DriveListBox, DirListBox and FileListBox controls . **The Drive ListBox** is used to display a list of drives available in your computer. When you place this control into the form and run the program,you will be able to select different drive from your computer. **The DirListBox is Directory ListBox** is used to display the list of directories or folder in a selected drive.When you place this control into the form and run the program, you will be able to select different directories from a selected drive in your computer**.**

The FileListBox is used to display the list of files in a selected directory or folder.When you place this control into the fo form and run the program, you will be able to a list of files in a selected directory.

1. Start Visual Basic and pick "Standard Exe" from the list of new projects.
2. A form1 container is open.
3. Add DriveListBox,DirListBox and FileListBox from the toolbox.

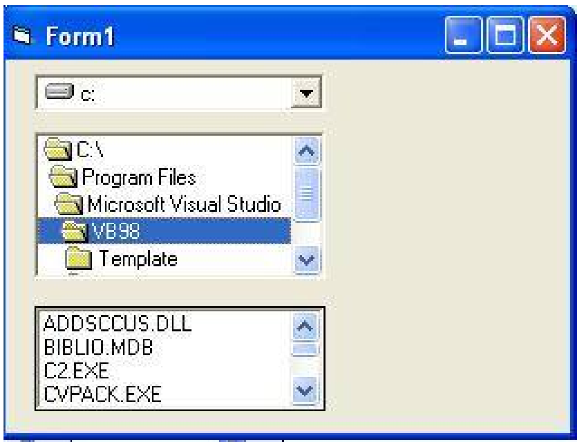


4. Double click on Drive control.Code window is open. Write code here.
5. After you select Dir1 object from object drop down list.Write code here.



```
Private Sub Dir1_Change()
File1.Path = Dir1.Path
End Sub

Private Sub Drive1_Change()
Dir1.Path=Drive1.Drive
End Sub
```

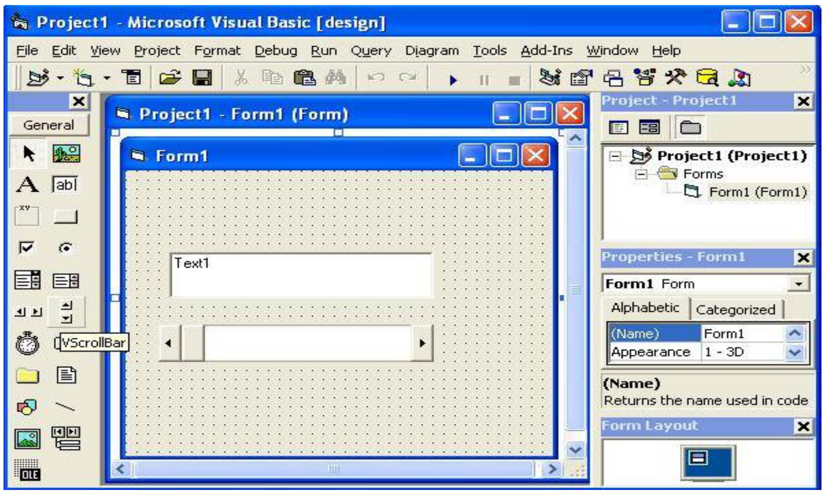
6.Press <f5> key to run the program and select the option to see what happens.



The ScrollBar

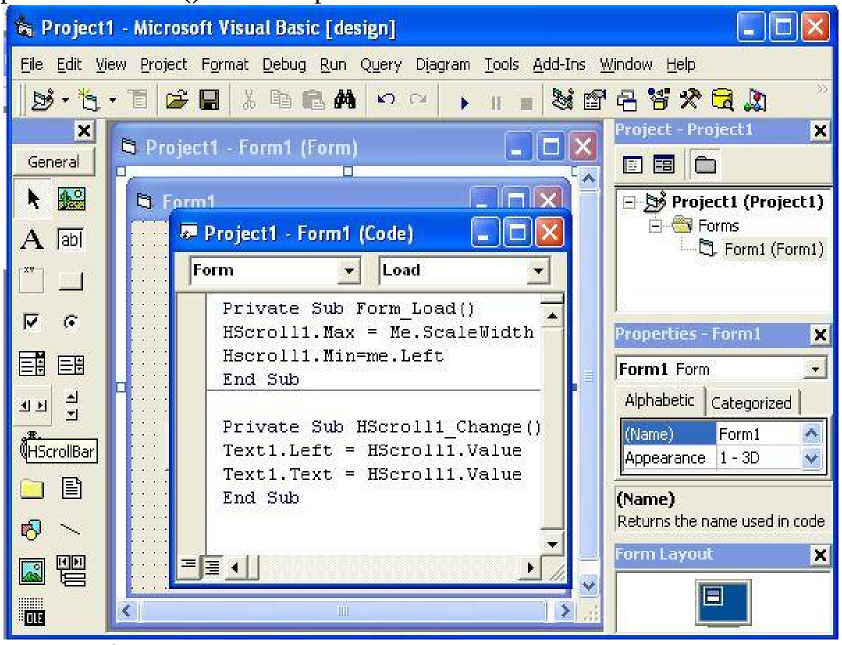
The ScrollBar is a commonly used control, which enables the user to select a value by positioning it at the desired location. A scroll bar control consists of a bar with arrows at each end. Visual basic provides two types of scroll bars-Horizontal scroll bar and vertical scroll bar.To the right of client area is vertical scroll bar, which is a common component of windows that display text .If there is too much text to fit in the window at once,scroll bars enable the user to move around within the document to look at some subsection of the whole. At the bottom of the window is a horizontal scroll bar that scrolls the text in the client area horizontally. To the right side of the horizontal scroll bar is the window's sizing handle.

1. Start Visual Basic and pick "Standard Exe" from the list of new projects.
2. A form1 container is open.
3. Add **Scrollbar and TextBox1 tool** from the **toolbox.**



4. Double click on scrollbar control. Code window is open. Write code here.

5. After you select Form1 object from object drop down list and select the procedure Load() from the procedure window .Write code here.



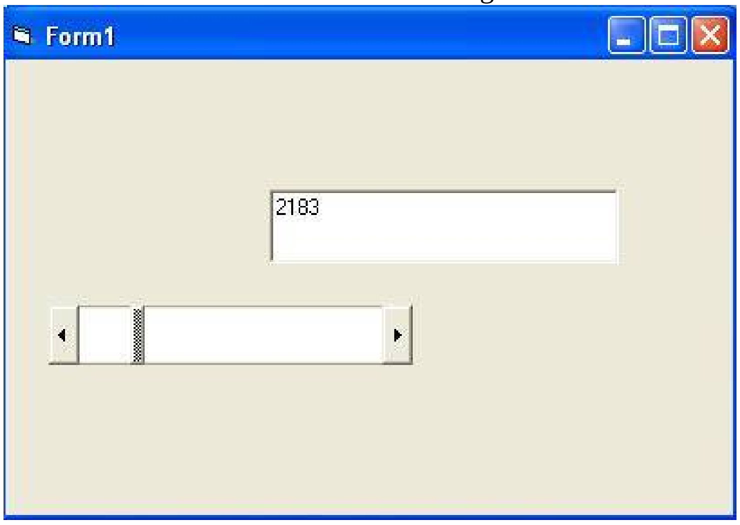
```
Private Sub Form_Load()
HScroll1.Max = Me.ScaleWidth
Hscroll1.Min=me.Left
End Sub

Private Sub HScroll1_Change()
Text1.Left = HScroll1.Value
Text1.Text = HScroll1.Value
End Sub
```

(Name)	Form1
Appearance	1 - 3D

(Name)
Returns the name used in code

6.Press <f5> key to run the program .

7. Scroll the scrollbox from scroll arrow.Its range isshown in textbox1.



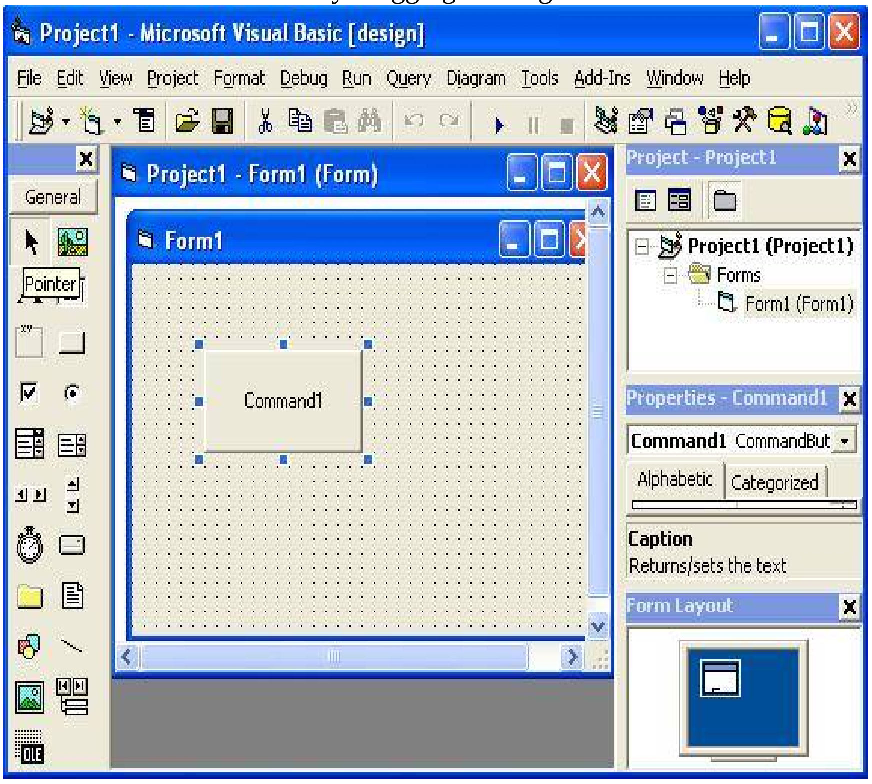
```
2183
```

Pointer Control

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

1. In the Toolbox, select the Pointer tool (if it isn't already selected).
2. On your form, select the control you want to resize.

3. Grab a sizing handle with the mouse by moving the pointer over it and then holding down the left mouse button. You know when you're over the sizing handle because the mouse pointer turns into a double-sided arrow.
4. While holding down the mouse button, notice that a box appears . The box shows you what the size of the control will be. When it's the right size, release the mouse button.
5. You can resize a control by dragging a sizing handle.

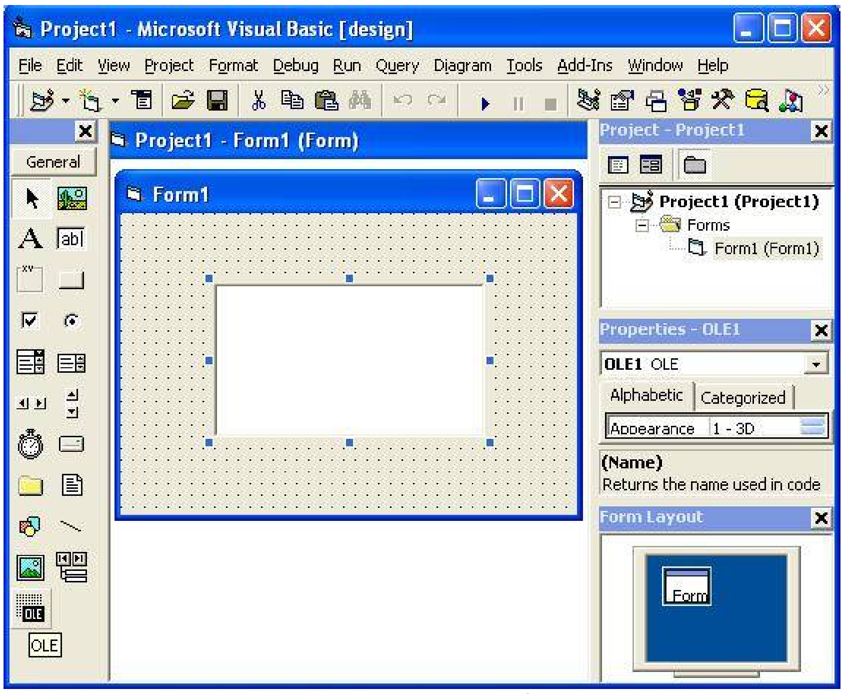


OLE control in Visual Basic

Insertable Objects (like the Excel Worksheet object; with a list of the employees, or a Project Calendar object having the scheduling information for a project.) These can be added to the toolbox, so they can be considered controls.

OLE (Object Linking and Embedding) is a means to interchange data between applications. One of the primary uses of the **OLE Container** control was to embed Word documents or Excel spreadsheets in a form. This is the control to use when you want to link or embed an object into your Visual Basic application.

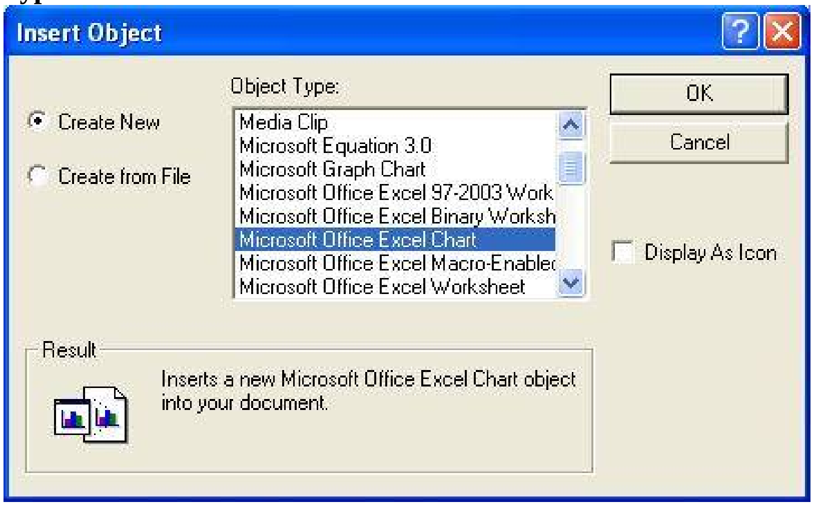
1. From the **Windows Start** menu, choose **Programs**, **Microsoft Visual Studio 6.0**, and then **Microsoft Visual Basic 6.0**.
2. Click the OLE tool form toolbox.



3. Double clicking the OLE control on your form will activate the object's application. Visual Basic also allows you to activate the object from your code. A Insert dialog box is open. There are two option create new or create from file. You select it as you want.

4. **First we create new Object**.

5. Select **Create New** and **Microsoft office Excel Chart** from the **Object Type list.**


Insert Object

(● Create New) Create from File

Object Type:
Media Clip
Microsoft Equation 3.0
Microsoft Graph Chart
Microsoft Office Excel 97-2003 Work
Microsoft Office Excel Binary Worksh
Microsoft Office Excel Chart
Microsoft Office Excel Macro-Enablec
Microsoft Office Excel Worksheet

Result
Inserts a new Microsoft Office Excel Chart object into your document.

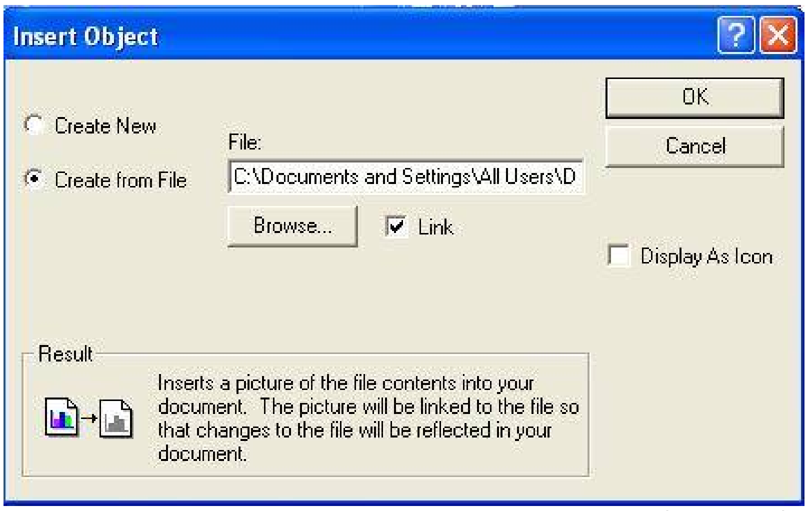
6. Creating a Excel chart will allow you to access the Excel application to define the chart.

7. Select the Excel chart.

8. Run your application. Double click on the OLE control and use it in your project as you want.

Second is linking to an existing object for this:

1. Add a OLE to the form.
2. Select Create from File and use the Browse button to find the Word document.
3. Select Link and click on OK.


Insert Object

( ) Create New
(•) Create from File

File: C:\Documents and Settings\All Users\D

[✓] Link
[ ] Display As Icon

Result
Inserts a picture of the file contents into your document. The picture will be linked to the file so that changes to the file will be reflected in your document.

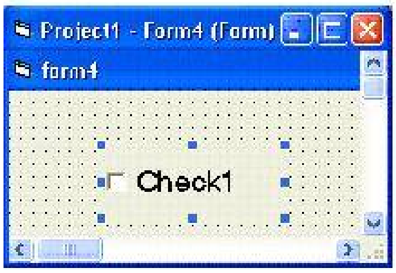
4. This Word document page has been sized so that it will fit onto a form. Change the SizeMode property of the OLE control to Stretch.
5. Run your application. Double click the OLE control to see that Word is activated with the selected document opened.
6. Close Word and stop your application.

Checkbox

A checkbox control is rather similar to an option button. The value property of both the controls is tested to check the current state. Check boxes are not mutually exclusive. A check box is a control that allows the user to set or change the value of an item as true or false. The control appears as a small square. When this empty square is clicked, it gets marked by a check symbol. These two states control the checkbox as checked or unchecked. The Checkbox List control is a single control that acts as a parent control for a collection of check-box list items. When a check box appears in a group with other similar controls, it assumes a completely independent behavior and it must be implemented as its own entity. This means that clicking a check box has no influence on the other controls.

1. **Open Visual Basic 6**.0 and create a new **Standard EXE Project**.
2. From **Standard EXE project** Dialogbox choose **Standard EXE form.**
2. Add one **checkbox** from **toolbox** on the form.

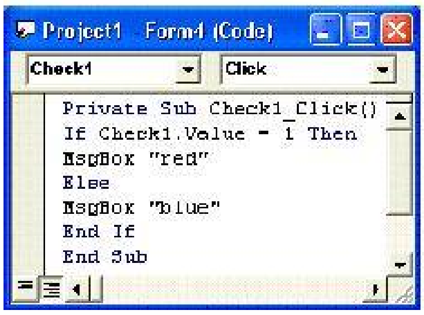
DESIGNING OF CHECKBOX:



3. **Double click** on Checkbox to open**code window** and type the following**code**

Private Sub Check1_Click()
If Check1.Value = 1 Then
MsgBox “red”
Else
MsgBox “blue”
End If
End Sub

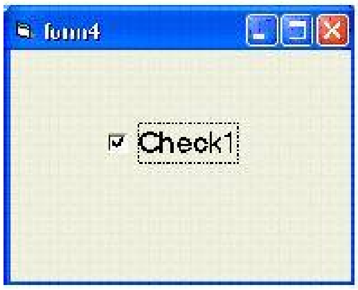
CODING OF CHECKBOX:



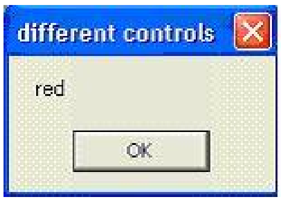
```
Private Sub Check1_Click()
If Check1.Value = 1 Then
MsgBox "red"
Else
MsgBox "blue"
End If
End Sub
```

4. Press **F5** to run

OUTPUT OF CHECKBOX:



5. When you select the checkbox output is:



red

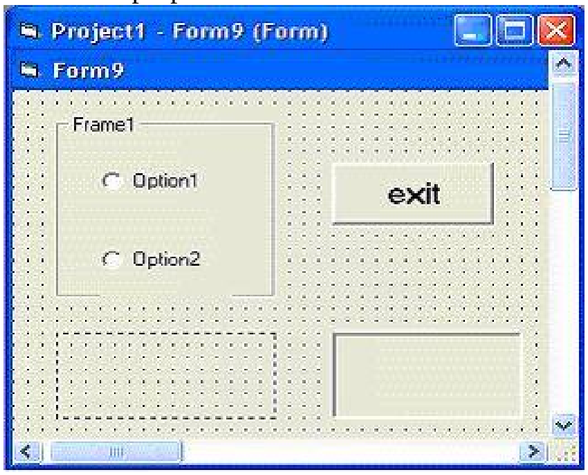
Frame:

A **frame** is an object that provide a visual a functional container for controls. It is an object that has solid border around it and a caption at the top. You can not just drag an option button over a frame to add it to the frame. Instead; you must actually draw the option button on the frame. This means you cannot draw an option button in a frame using the shortcut of double-clicking the control in the toolbox.

Image Control

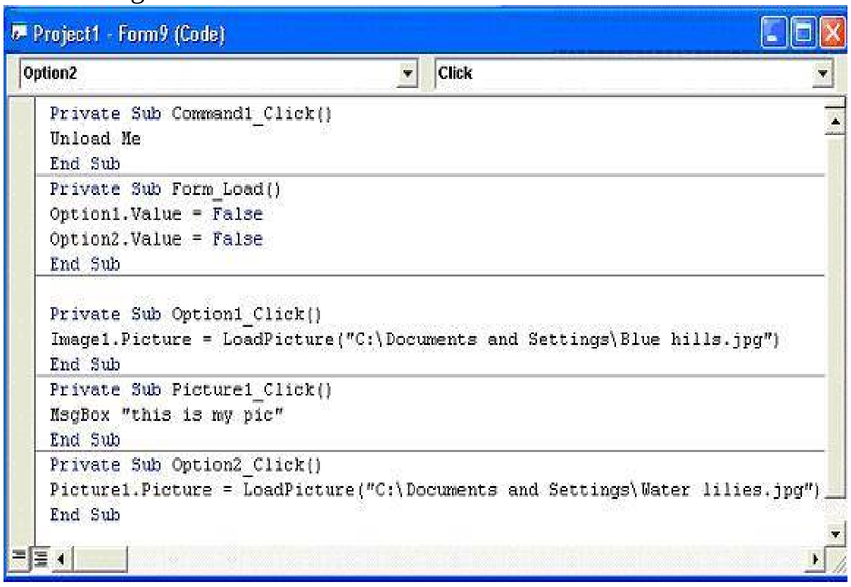
Image Control is another control that handles images and pictures. It functions almost identically to the picture box.However, there is one major difference, the image in an Image Box is stretchable, which means it can be resresized.This feature is not available in the Picture Box. Similar to the Picture Box, it can also use the LoadPicture method to load the picture. For example, the statement loads the picture “winter.jpg”: into the image box.

1. Start -> program-> Microsoft Visual Studio 6.0 -> Microsoft Visual Basic 6.0
2. Visual Basic load and appears on your screen.
3. Choose Standard Exe. From dialog box.
4. Add Frame1, Command1 button, one Picture control and one Image control from toolbox place left side on the visual basic screen.
5. Add two option button option1 and option2.
6. Set the properties of Frame1, Command1 button, Picture control and Image control from the properties window.



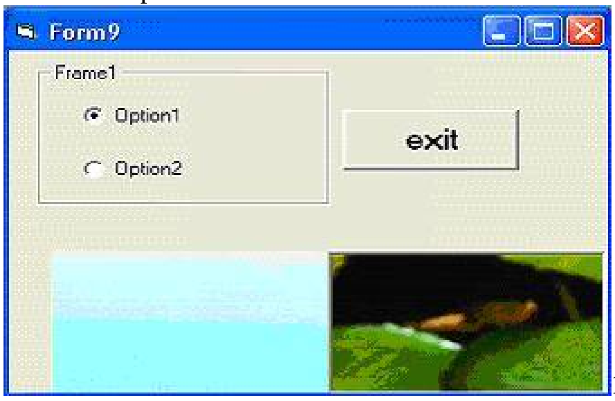
7. After you click on code window button from the project window. Code window is open.

8. First you select the object from object drop down list writ code here.As shown in figure below.

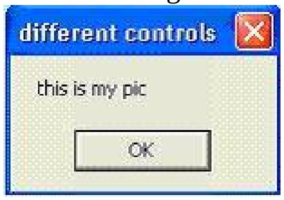


```
Private Sub Command1_Click()
Unload Me
End Sub
Private Sub Form_Load()
Option1.Value = False
Option2.Value = False
End Sub

Private Sub Option1_Click()
Image1.Picture = LoadPicture("C:\Documents and Settings\Blue hills.jpg")
End Sub
Private Sub Picture1_Click()
MsgBox "this is my pic"
End Sub
Private Sub Option2_Click()
Picture1.Picture = LoadPicture("C:\Documents and Settings\Water lilies.jpg")
End Sub
```

9. When the form is load option1 and option2 is false. Click on option1 button the Image is show in Image1 Box and when you click on option2 Button then picture is show in Picture1 Box.



10.When you click on picture1 then message box is show.



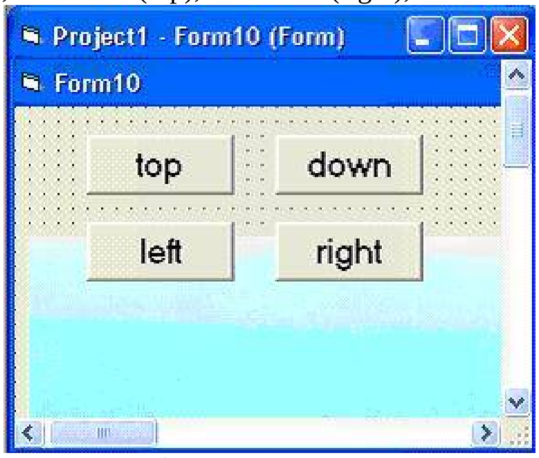
this is my pic

10. When you click on Command1 Button then form is exit.

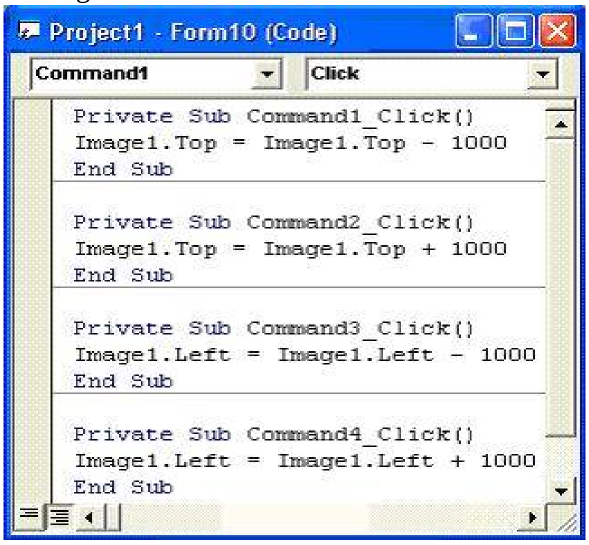
Picture Box

It control is the most powerful and complex items in the Visual Basic .The picture box is one of the controls that used to handle graphics. you can load a picture at design phase by clicking on the picture item in the properties window and select the picture from the selected folder .You can also load the picture at runtime using the LoadPicture metoh.For example, the statement will load the picture “abc.jpg” into the picture box.

1. Start -> program-> Microsoft Visual Studio 6.0 -> Microsoft Visual Basic 6.0
2. Visual Basic load and appears on your screen.
3. Choose Standard Exe. From dialog box.
4. Add Picture1 control ReRand four Button command1, command2, command3, command4 from the toolbox on the form.
5. Set the caption property of command1 (down),command2(top),command3(right),command4(left).

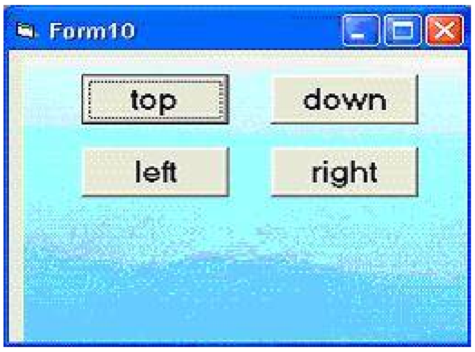


6. Click on code window from project window. Code window is open select the object one by one from the object window and write the code here as shown in figure.



```
Private Sub Command1_Click()
Image1.Top = Image1.Top - 1000
End Sub

Private Sub Command2_Click()
Image1.Top = Image1.Top + 1000
End Sub

Private Sub Command3_Click()
Image1.Left = Image1.Left - 1000
End Sub

Private Sub Command4_Click()
Image1.Left = Image1.Left + 1000
End Sub
```

7. When the coding is complete then you run the program.
8. To run the program press <f5>.output is:

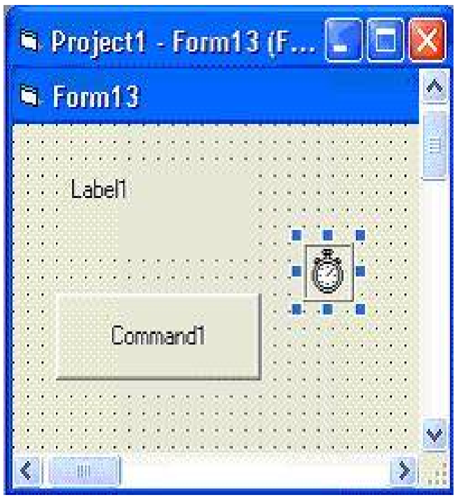


9. When you click on top button picture move on top, if click on down picture move in down, if click on left picture move in left, if click on right picture move in right.

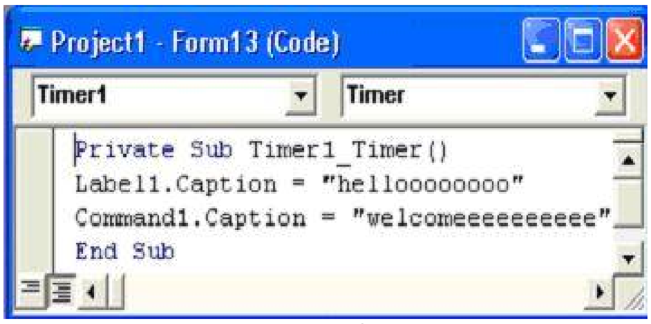
Timer control

The **Timer control** is used to fire an event at specified intervals. The code you want to execute is placed in this event. This control has no methods, only one event (Time event) and seven properties. The control is invisible at run time.

1. Start -> program-> Microsoft Visual Studio 6.0 -> Microsoft Visual Basic 6.0
2. Visual Basic load and appears on your screen.
3. Choose Standard Exe. From dialog box.
4. A form1 container is open.
5. Add one **Label1, Command1 button and Timer1 control** on the form1.

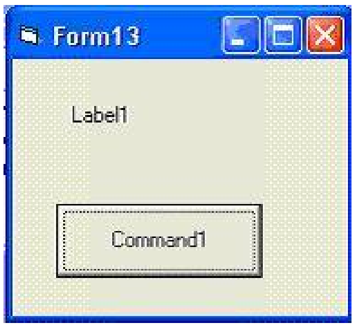


6. Set the time property **"2000"** for Timer1 control.
7. Double click on timer control. The code window is open. Write the code here.

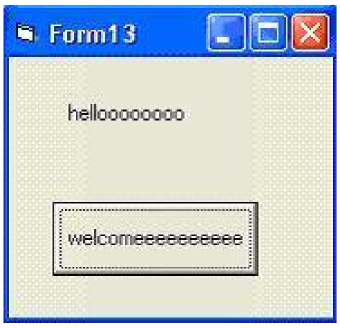


```
Private Sub Timer1_Timer()
Label1.Caption = "helloooooooo"
Command1.Caption = "welcomeeeeeeeeee"
End Sub
```

8. Run your program. Click “Debug” from the menu bar. Click “Start” from the drop down menu. Alternatively, you can press <F5> on your keyboard.Output is:



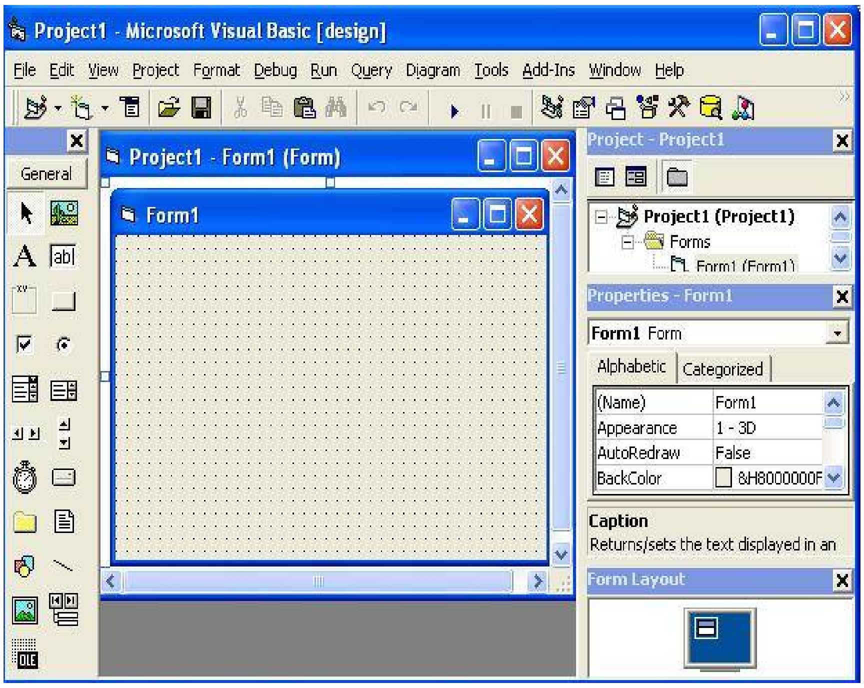
9. Time is start when you rum the program.when time is complete the caption of Label1 and command1 button is change.As shown in Figure below:



Tool-tip

A tool-tip is a small box that is displayed at the mouse location when overing above the control .The tool-tip is set for a brief period of time, it provides a help prompt to the user. A tooltip is the first line of help for a confused user. An example of its use would be on an invalid entry into a form and when the user runs their mouse over control you would use a tool-tip to tell them why we use this control or about the control.

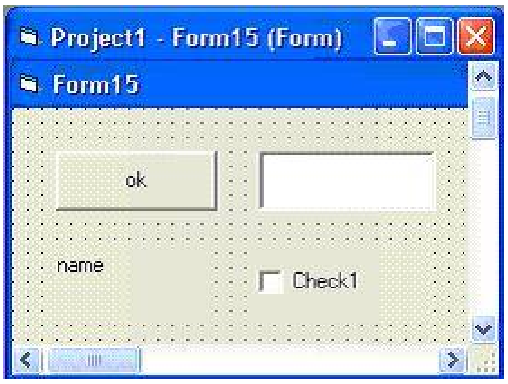
1. Start Visual Basic and pick “Standard Exe” from the of new projects. You’ll land in the screen we saw below.



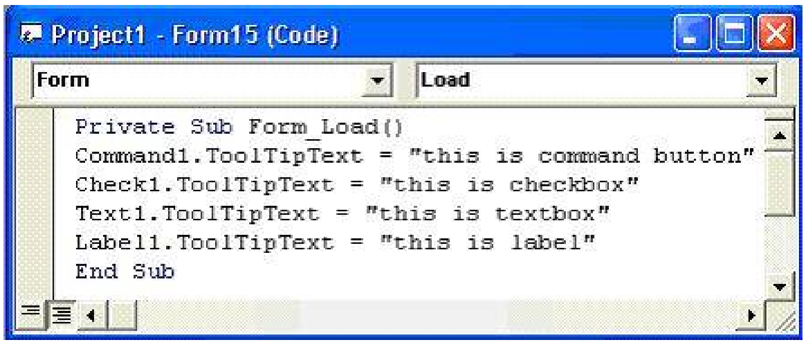
(Name)	Form1
Appearance	1 - 3D
AutoRedraw	False
BackColor	&H8000000F

Caption
Returns/sets the text displayed in an

2. Add the Coomand1 button, Checkbox1, Textbox1 and Label1 on the Form1 container.

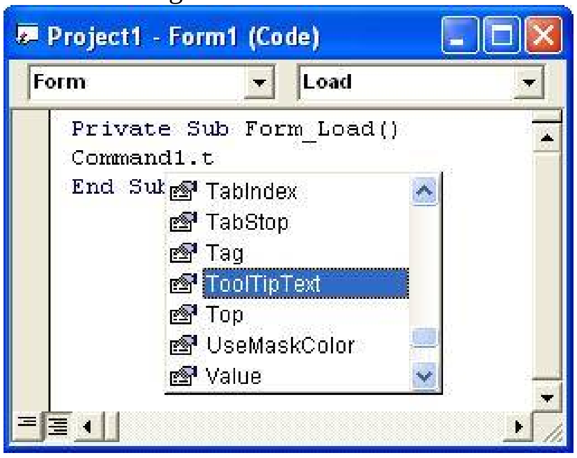


3. Double click on Form1 container. The code window is open. Write code here.



```
Private Sub Form_Load()
Command1.ToolTipText = "this is command button"
Check1.ToolTipText = "this is checkbox"
Text1.ToolTipText = "this is textbox"
Label1.ToolTipText = "this is label"
End Sub
```

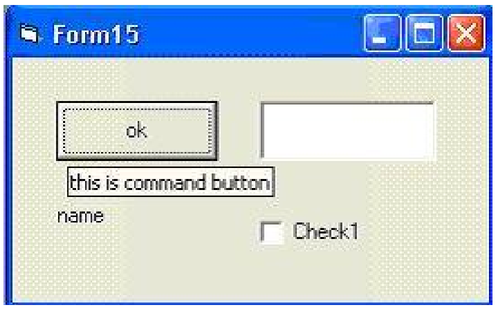
4. When you write the code and type command1 and type the dot(Command1.) a popup menu is open select the tooltip option from the popup menu as shown in figure:



```
Private Sub Form_Load()
Command1.t
End Sub
```

5. Press <f5> key to run the program and when you point out the cursor on the button to see what happens.

6. A **tooltip** is shown:



Review & Self Assessment Question

Q1-What is control? Explain with its type.
Q2-Explain Command Button with example?
Q3-What are the differences between Image box and Picture box?
Q4-What is Frame? Explain with example.

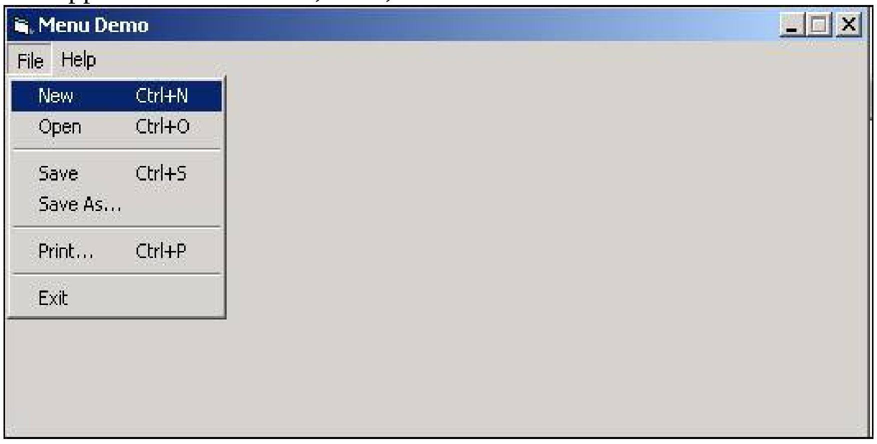
Further Readings

Visual basic by David I Schneider
Visual basic by Steven Holzner
Visual basic by Bill Sheldon
Visual basic by Billy Holls
Visual basic by Rob Windsor

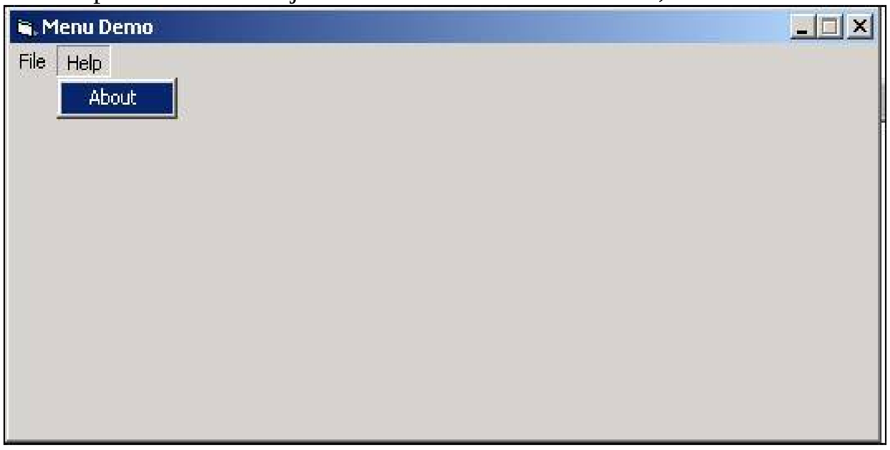
UNIT: 6-CREATING MENU, POP UPS, TOOLBARS AND DIALOG BOX

Contents

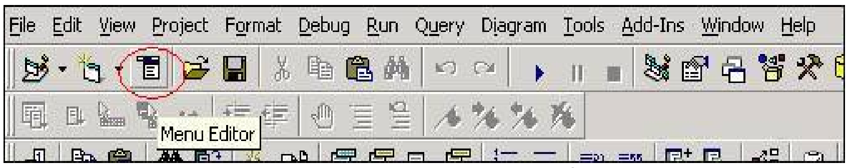
The File menu, shown below, will have the following level-two items below it: New, Open, Save, Save As, Print, and Exit. Note that separator bars appear above the Save, Print, and Exit items.



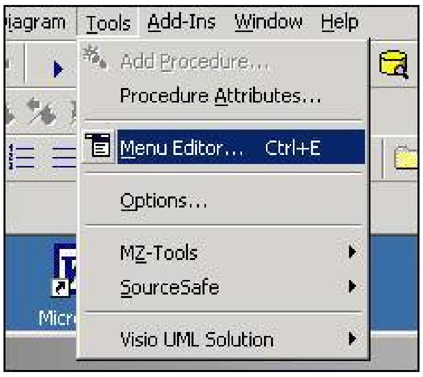
The Help menu contains just one level-two item below it, about.



To build a menu for use with your VB program, you use the Menu Editor, which appears as an icon in the toolbar of the VB IDE. It is the circled item in the screen shot below:

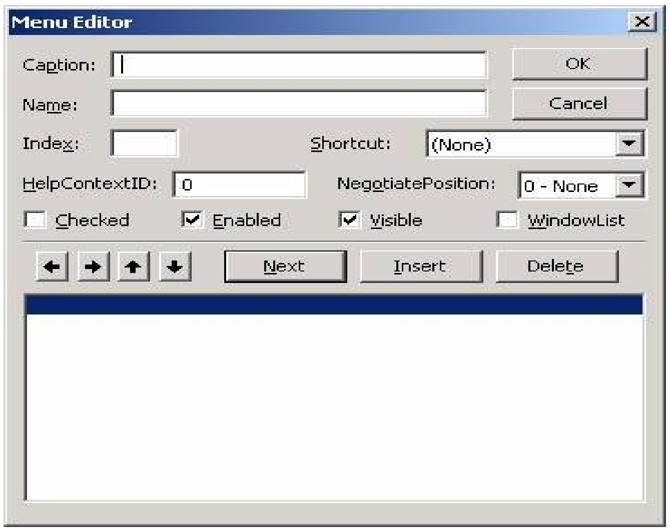

File Edit View Project Format Debug Run Query Diagram Tools Add-Ins Window Help
Menu Editor

Alternatively, you can invoke the Menu Editor from the Tools menu item as shown below:



To build the menu described above, perform the following steps.

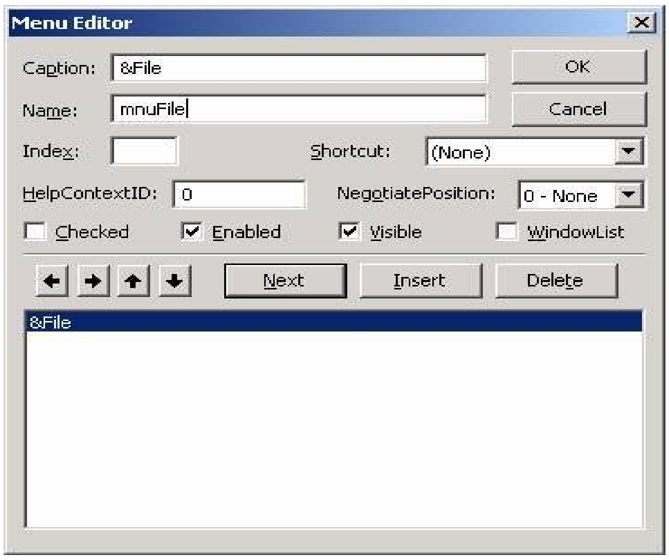
1. Start a new VB project and invoke the Menu Editor using either method shown above (click the Menu Editor toolbar icon or select the Menu Editor option from the Tools menu). The Menu Editor screen appears, as shown below:



2. For "Caption", type &File (by placing the ampersand to the left of the "F", we establish "F" as an access key for the File item it enables the user to drop down the File menu by keying "Alt+F" on the keyboard in addition to clicking the "File" item with the mouse).

For "Name", type mnuFile.

Your Menu Editor screen should look like this:

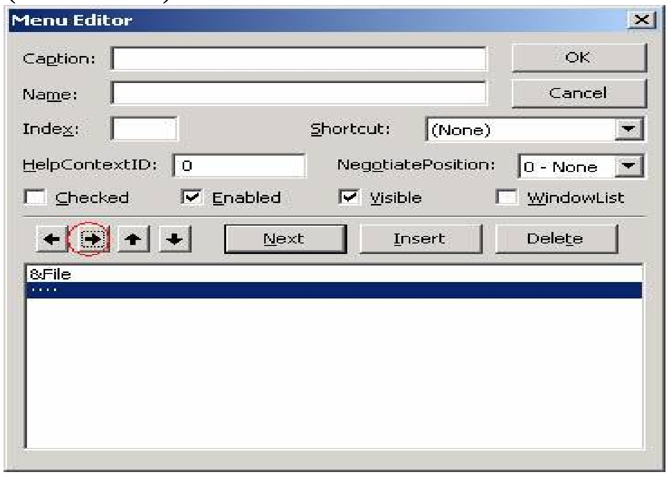


Caption: &File
Name: mnuFile
Index:
Shortcut: (None)
HelpContextID: 0
NegotiatePosition: 0 - None
Checked [ ] Enabled [✓] Visible [✓] WindowList [ ]

&File

Click the Next button.

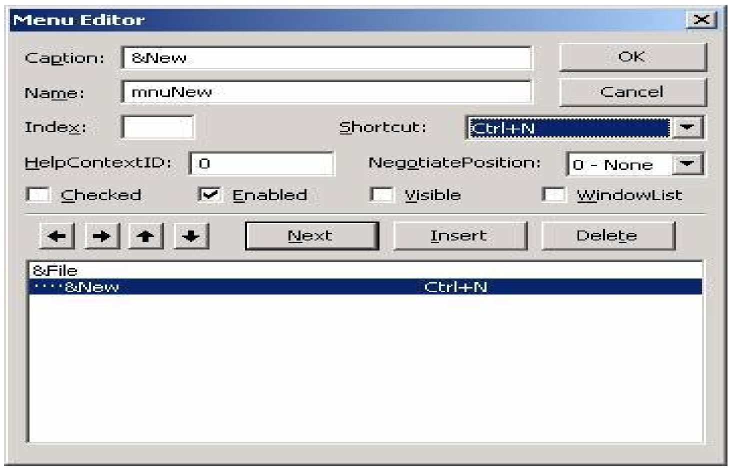
3. Click the "right-arrow" button (shown circled below). A ellipsis (...) will appear as the next item in the menu list, indicating that this item is a level-two item (below "File").



Caption:
Name:
Index:
Shortcut: (None)
HelpContextID: 0
NegotiatePosition: 0 - None
Checked [ ] Enabled [✓] Visible [✓] WindowList [ ]

&File
····

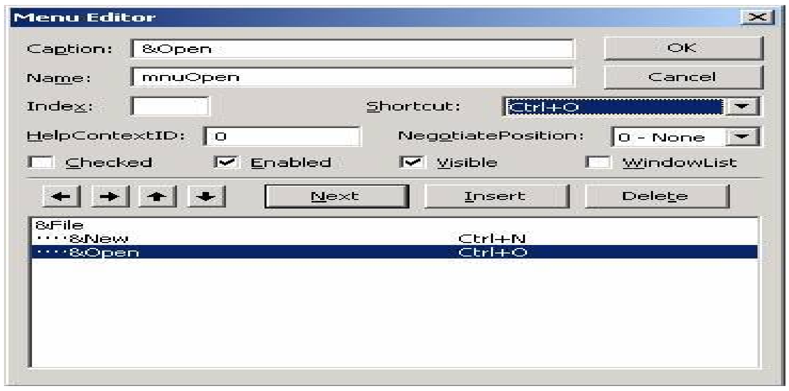
For "Caption", type &New; for "Name", type mnuNew, and for "Shortcut", select Ctrl+N. By specifying a shortcut, you allow the user to access the associated menu item by pressing that key combination. So here, you are providing the user three ways of invoking the "New" function: (1) clicking File, then clicking New on the menu; (2) keying Alt+F,N (because we set up an access key for "N" by placing an ampersand to left of "N" in "New"); or (3) keying Ctrl+N. At this point, your Menu Editor screen should look like this:


Menu Editor
Caption: &New
Name: mnuNew
Index:
Shortcut: Ctrl+N
HelpContextID: 0
NegotiatePosition: 0 - None
Checked [✓ Enabled] Visible WindowList

```
&File
····&New                                   Ctrl+N
```

Click the Next button.

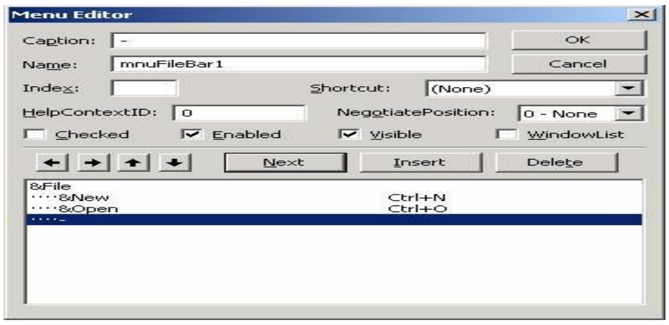
4. For "Caption", type &Open; for "Name", type mnuOpen, and for "Shortcut", select Ctrl+O. Your Menu Editor screen should look like this:


Menu Editor
Caption: &Open
Name: mnuOpen
Index:
Shortcut: Ctrl+O
HelpContextID: 0
NegotiatePosition: 0 - None
Checked [✓ Enabled] [✓ Visible] WindowList

```
&File
····&New                                   Ctrl+N
····&Open                                  Ctrl+O
```

Click the Next button.

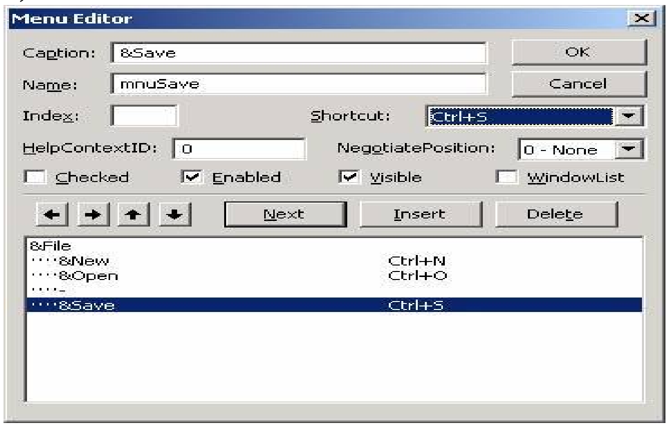
5. For "Caption", type - (a hyphen), and for "Name", type mnuFileBar1. A single hyphen as the Caption for a menu item tells VB to create a separator bar at that location. Your Menu Editor screen should look like this:


Menu Editor
Caption: -
Name: mnuFileBar1
Index:
Shortcut: (None)
HelpContextID: 0
NegotiatePosition: 0 - None
Checked [✓ Enabled] [✓ Visible] WindowList

```
&File
····&New                                   Ctrl+N
····&Open                                  Ctrl+O
····-
```

Click the Next button.

6. For "Caption", type &Save; for "Name", type mnuSave, and for "Shortcut", select Ctrl+S. Your Menu Editor screen should look like this:

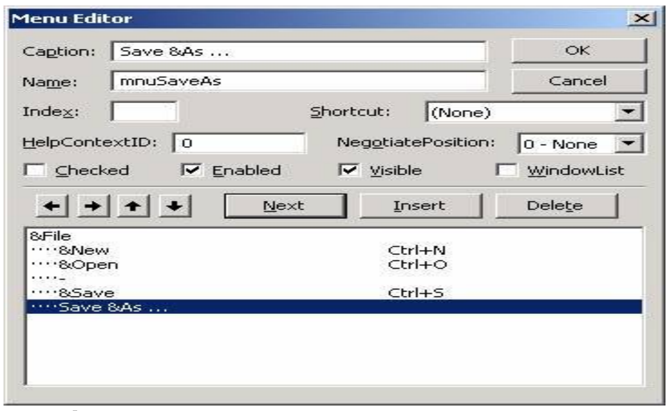


Caption: &Save
Name: mnuSave
Index:
Shortcut: Ctrl+S
HelpContextID: 0
NegotiatePosition: 0 - None
☐ Checked ☑ Enabled ☑ Visible ☐ WindowList

```
&File
····&New                                  Ctrl+N
····&Open                                 Ctrl+O
····-
····&Save                                 Ctrl+S
```

Click the Next button.

7. For "Caption", type Save &As ..., and for "Name", type mnuSaveAs. Your Menu Editor screen should look like this:

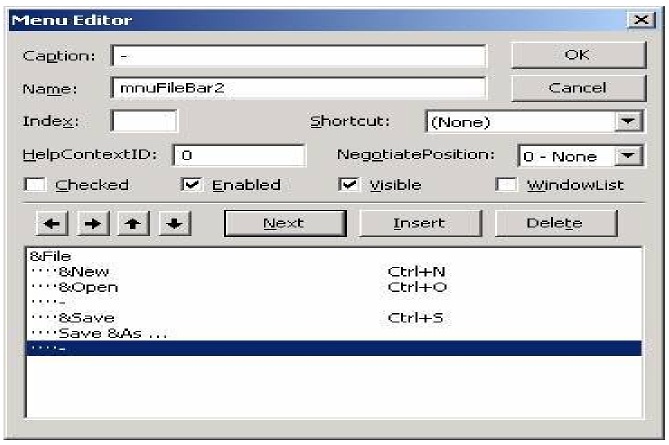


Caption: Save &As ...
Name: mnuSaveAs
Index:
Shortcut: (None)
HelpContextID: 0
NegotiatePosition: 0 - None
☐ Checked ☑ Enabled ☑ Visible ☐ WindowList

```
&File
····&New                                  Ctrl+N
····&Open                                 Ctrl+O
····-
····&Save                                 Ctrl+S
····Save &As ...
```

Click the Next button.

8. For "Caption", type -, and for "Name", type mnuFileBar2. Your Menu Editor screen should look like this:

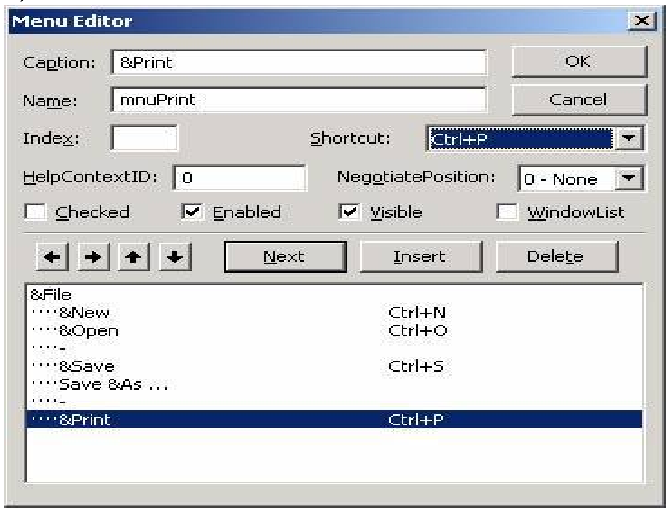


Caption: -
Name: mnuFileBar2
Index:
Shortcut: (None)
HelpContextID: 0
NegotiatePosition: 0 - None
☐ Checked ☑ Enabled ☑ Visible ☐ WindowList

```
&File
····&New                                  Ctrl+N
····&Open                                 Ctrl+O
····-
····&Save                                 Ctrl+S
····Save &As ...
····-
```

Click the Next button.

9. For "Caption", type &Print;for "Name", type mnuPrint; and for "Shortcut", select Ctrl+P. Your Menu Editor screen should look like this:

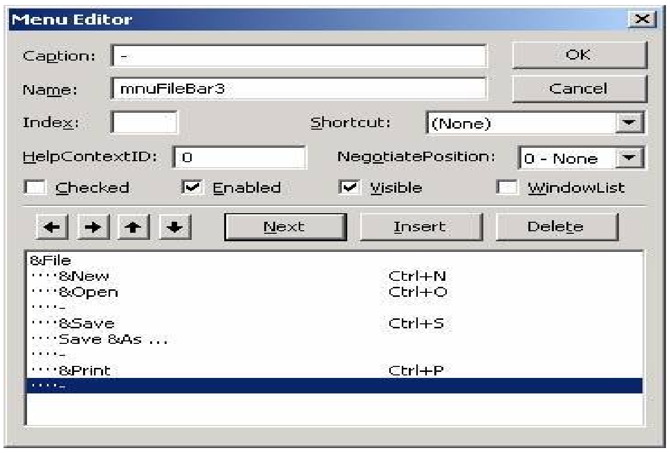


Caption: &Print
Name: mnuPrint
Index:
Shortcut: Ctrl+P
HelpContextID: 0
NegotiatePosition: 0 - None
☐ Checked ☑ Enabled ☑ Visible ☐ WindowList

```
&File
····&New                              Ctrl+N
····&Open                             Ctrl+O
····-
····&Save                             Ctrl+S
····Save &As ...
····-
····&Print                            Ctrl+P
```

Click the Next button.

10. For "Caption", type -; and for "Name", type mnuFileBar3. Your Menu Editor screen should look like this:

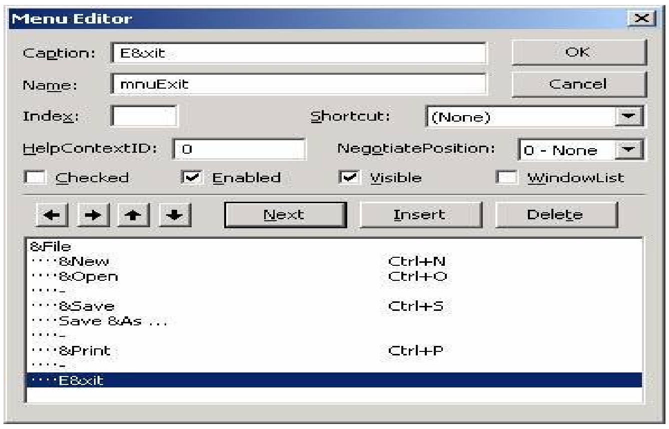


Caption: -
Name: mnuFileBar3
Index:
Shortcut: (None)
HelpContextID: 0
NegotiatePosition: 0 - None
☐ Checked ☑ Enabled ☑ Visible ☐ WindowList

```
&File
····&New                              Ctrl+N
····&Open                             Ctrl+O
····-
····&Save                             Ctrl+S
····Save &As ...
····-
····&Print                            Ctrl+P
····-
```

Click the Next button.

11. For "Caption", type E&xit, and for "Name", type mnuExit. Your Menu Editor screen should look like this:



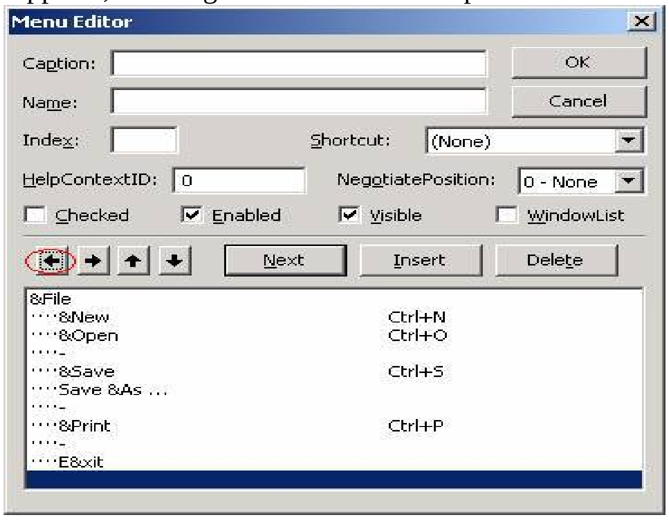
Menu Editor

Caption: E&xit
Name: mnuExit
Index:
Shortcut: (None)
HelpContextID: 0
NegotiatePosition: 0 - None
Checked [ ] Enabled [✓] Visible [✓] WindowList [ ]

```
&File
····&New                         Ctrl+N
····&Open                        Ctrl+O
····-
····&Save                        Ctrl+S
····Save &As ...
····-
····&Print                       Ctrl+P
····-
····E&xit
```

Click the Next button.

12. Click the "left-arrow" button (shown circled below). The ellipsis (...) no longer appears, meaning we are back to the top-level items.

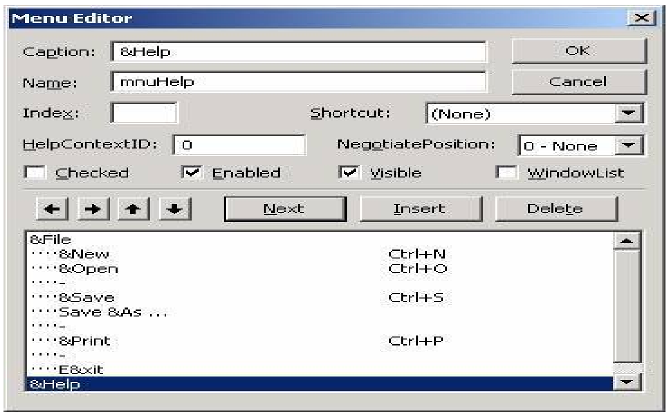


Menu Editor

Caption:
Name:
Index:
Shortcut: (None)
HelpContextID: 0
NegotiatePosition: 0 - None
Checked [ ] Enabled [✓] Visible [✓] WindowList [ ]

```
&File
····&New                         Ctrl+N
····&Open                        Ctrl+O
····-
····&Save                        Ctrl+S
····Save &As ...
····-
····&Print                       Ctrl+P
····-
····E&xit
```

For "Caption", type &Help; and for "Name", type mnuHelp. Your Menu Editor screen should look like this:



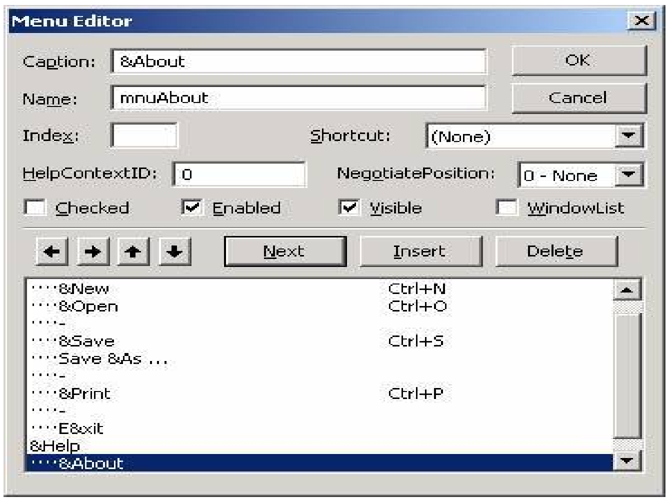
Menu Editor

Caption: &Help
Name: mnuHelp
Index:
Shortcut: (None)
HelpContextID: 0
NegotiatePosition: 0 - None
Checked [ ] Enabled [✓] Visible [✓] WindowList [ ]

```
&File
····&New                         Ctrl+N
····&Open                        Ctrl+O
····-
····&Save                        Ctrl+S
····Save &As ...
····-
····&Print                       Ctrl+P
····-
····E&xit
&Help
```

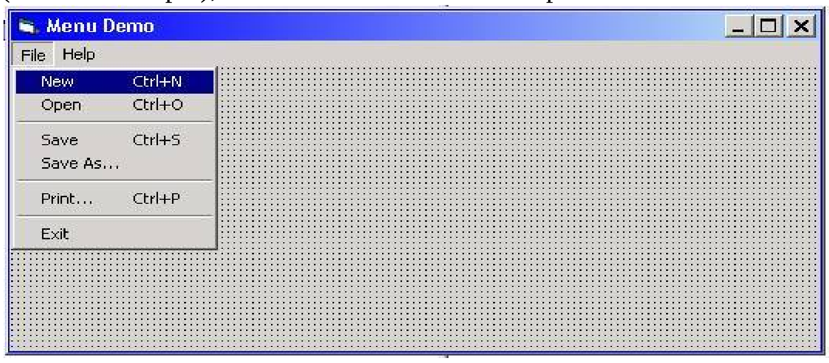
Click the Next button.

13. Click the "right-arrow" button to create a level-two item below "Help". For "Caption", type & about; and for "Name", type mnuAbout. Your Menu Editor screen should look like this:



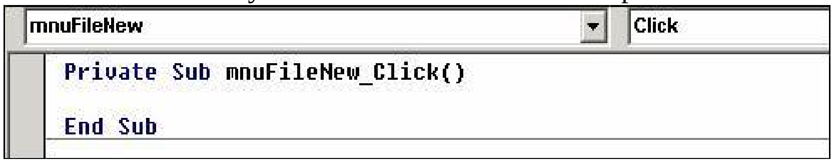
14. At this point, we are done creating our menu entries, so click the OK button. That will dismiss the menu editor and return focus to the VB IDE.

15. Back in the VB IDE, your form will now have a menu, based on what you have set up in the Menu Editor. If you click on a top-level menu item (File for example), the level-two menu will drop down:



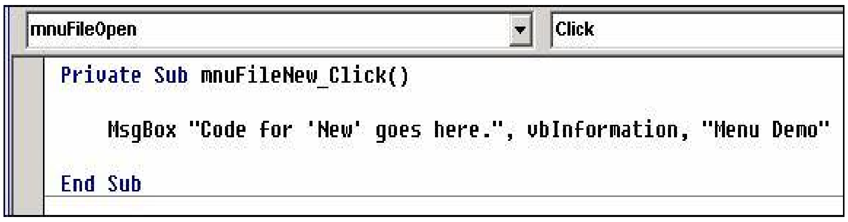
16. Click on the New menu item. The code window for the mnuFileNew_Click event opens, as shown below.

Note: Click is the only event that a menu item can respond to.



```
Private Sub mnuFileNew_Click()

End Sub
```

In thePlace mnuFileNew_Click event, place the code you want to execute when the user clicks the New menu item. Since this is just a demo, we will place a simple MsgBox statement in the event procedure:
MsgBox "Code for 'New' goes here.", vbInformation, "Menu Demo"



```
Private Sub mnuFileNew_Click()

    MsgBox "Code for 'New' goes here.", vbInformation, "Menu Demo"

End Sub
```

17. Code similar MsgBox statements for the Open, Save, Save As, and Print menu items:

```
Private Sub mnuFileOpen_Click()
MsgBox "Code for 'Open' goes here.", vbInformation, "Menu Demo"
End Sub
Private Sub mnuFileSave_Click()
MsgBox "Code for 'Save' goes here.", vbInformation, "Menu Demo"
End Sub
Private Sub mnuFileSaveAs_Click()
MsgBox "Code for 'Save As' goes here.", vbInformation, "Menu Demo"
End Sub
Private Sub mnuFilePrint_Click()
MsgBox "Code for 'Print' goes here.", vbInformation, "Menu Demo"
End Sub
```

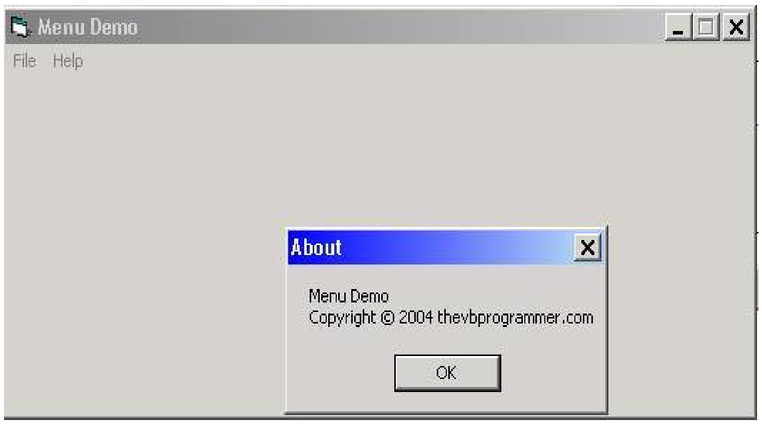
18. For the Exit menu item Click event, code the statement Unload Me.

```
Private Sub mnuFileExit_Click()
Unload Me
End Sub
```

19. For the About menu item Click event, code as shown below:

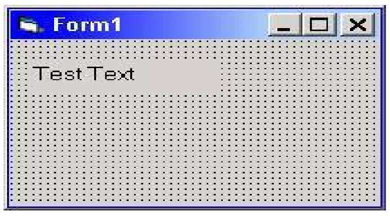
```
Private Sub mnuHelpAbout_Click()
MsgBox "Menu Demo" & vbCrLf _
& "Copyright " & Chr$(169) & " 2004 thevbprogrammer.com", , "About"
End Sub
```

20. Run the program. Note how the code executes when you click on the various menu items. Also test the use of the access keys (e.g., Alt+F, N) and shortcut keys (e.g., Ctrl-O).

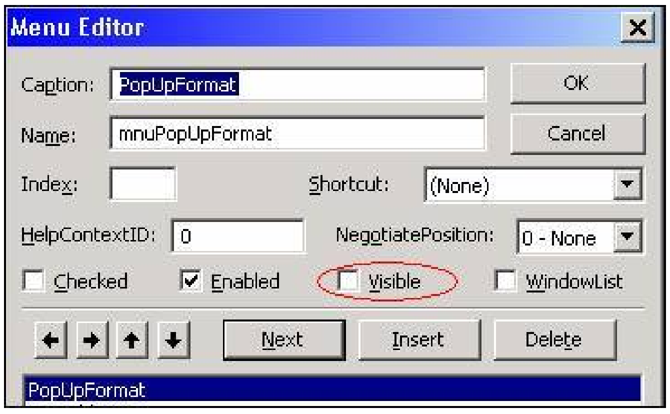


Menu Demo
Copyright © 2004 thevbprogrammer.com

21. save the program and exit VB.
EXAMPLE 2:
This example shows you how to create a popup menu (sometimes called a context menu or a right-click menu).
1. Start a new VB project and place a label on the form. Name the label lblTestText. Set the Caption to Test Text.



2. Open the Menu Editor, and create a top-level item with a Caption value of PopUpFormat and the NamemnuPopuUpFormat. Also importantly uncheck the visible checkbox (see the circled item below). In order for a menu to be a pop-up menu, it must be invisible.



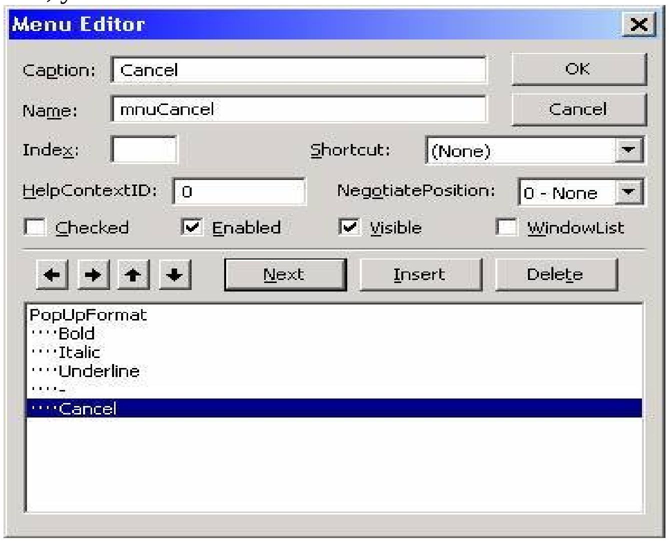
Caption: PopUpFormat
Name: mnuPopUpFormat
Index:
Shortcut: (None)
HelpContextID: 0
NegotiatePosition: 0 - None
☐ Checked ☑ Enabled ☐ Visible ☐ WindowList

PopUpFormat

3. Create the following level-two menu items below the PopUpFormat top-level menu. (When creating these level-two items, keep the Visible box checked.)

Caption	Name
Bold	mnuBold
Italic	mnuItalic
Underline	mnuUnderline
- (hyphen)	mnuFormatSep
Cancel	mnuCancel

When done, your Menu Editor should look like this:



Caption: Cancel
Name: mnuCancel
Index:
Shortcut: (None)
HelpContextID: 0
NegotiatePosition: 0 - None
☐ Checked ☑ Enabled ☑ Visible ☐ WindowList

```
PopUpFormat
····Bold
····Italic
····Underline
····-
····Cancel
```

4. Click OK to save your changes. Note: When you return to the IDE, you will NOT see this menu on the form (remember it's a pop-up menu, and it will only be visible when invoked through code).
5. Code the lblTestText_MouseDown event as shown below. Note that the Button parameter is tested forvbRightButton as is conventional, we only want to pop up the menu if the user right-clicks on the label. If the user clicks the right mouse button, the PopupMenu statement is executed. It is this statement that makes the pop-up menu appear.

```
Private Sub lblTestText_MouseDown(Button As Integer, _
Shift As Integer, _
X As Single, _
Y As Single)
If Button = vbRightButton Then
PopupMenu mnuPopUpFormat, vbPopupMenuRightButton
End If
End Sub
```

The full syntax for the PopupMenu method, from MSDN, is: object.PopupMenu menuname, flags, x, y, boldcommand

The PopupMenu method syntax has these parts:

Part	Description
object	Optional. An object expression that evaluates to an object in the Applies To list. If object is omitted, the form with the focus is assumed to be object.
Menuname	Required. The name of the pop-up menu to be displayed. The specified menu must have at least one submenu.
Flags	Optional. A value or constant that specifies the location and behavior of a pop-up menu, described as follows: Note: To specify both a "location" constant and a "behavior" constant, add the two values together. For example: PopupMenu MyMenu, vbPopupMenuRightAlign + vbPopupMenuRightButton
X	Optional. Specifies the x-coordinate where the pop-up menu is displayed. If omitted, the mouse coordinate is used.
Y	Optional. Specifies the y-coordinate where the pop-up menu is displayed. If omitted, the mouse coordinate is used.
boldcommand	Optional. Specifies the name of a menu control in the pop-up menu to display its caption in bold text. If omitted, no controls in the pop-up menu appear in bold.

5. Code the mnuBold_Click event as shown below. Note that the Checked property of the menu item is used. When set to True, this causes a checkmark to appear to the left of the menu item. The Checked property is typically used as a toggle.

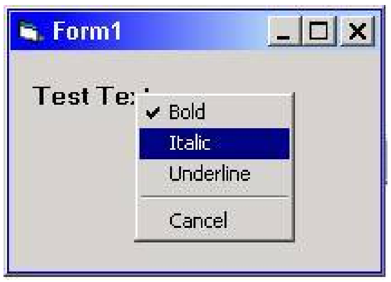
```
Private Sub mnuBold_Click()
If mnuBold.Checked Then
lblTestText.FontBold = False
mnuBold.Checked = False
Else
lblTestText.FontBold = True
mnuBold.Checked = True
End If
End Sub
```

6. Code the mnuItalic_Click and mnuUnderline_Click events in a similar fashion as shown below.

```
Private Sub mnuItalic_Click()
If mnuItalic.Checked Then
lblTestText.FontItalic = False
mnuItalic.Checked = False
Else
```

```
lblTestText.FontItalic = True
mnuItalic.Checked = True
End If
End Sub
Private Sub mnuUnderline_Click()
If mnuUnderline.Checked Then
lblTestText.FontUnderline = False
mnuUnderline.Checked = False
Else
lblTestText.FontUnderline = True
mnuUnderline.Checked = True
End If
End Sub
```

7. Run the program and check out the various options you have coded.



Review & Self Assessment Question

Q1-Write the steps to create the menu using menu editor option?
Q2- What do you mean by popup Menu?

Further Readings

Visual basic by David I Schneider
Visual basic by Steven Holzner
Visual basic by Bill Sheldon
Visual basic by Billy Holls
Visual basic by Rob Windsor

UNIT: 7 - PROCEDURES & FUNCTION

Formatted: Centered, Indent: Left: 1", First line: 0", Border: Top: (Single solid line, Auto, 0.5 pt Line width), Bottom: (Single solid line, Auto, 0.5 pt Line width)

Contents

Visual Basic offers different types of procedures to execute small sections of coding in applications. The various procedures are elucidated in details in this section. Visual Basic programs can be broken into smaller logical components **called Procedures**. Procedures are useful for condensing repeated operations such as the frequently used calculations, text and control manipulation etc. The benefits of using procedures in programming are:

It is easier to debug a program a program with procedures, which breaks a program into discrete logical limits.

Procedures used in one program can act as building blocks for other programs with slight modifications.

A Procedure can be Sub, Function or Property Procedure.

Sub Procedures

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

```
[Private | Public] [Static] Sub Procedure name [( arglist)]
[ statements]
End Sub
```

Arg list is a list of argument names separated by commas. Each argument acts like a variable in the procedure. There are two types of Sub Procedures namely general procedures and event procedures.

Event Procedures

An event procedure is a procedure block that contains the control's actual name, an underscore (_), and the event name. The following syntax represents the event procedure for a Form Load event.

```
Private Sub Form_Load()
....statement block..
End Sub
```

Event Procedures acquire the declarations as Private by default.

General Procedures

A general procedure is declared when several event procedures perform the same actions. It is a good programming practice to write common statements in a separate procedure (general procedure) and then call them in the event procedure.

In order to add General procedure:

- The Code window is opened for the module to which the procedure is to be added.
- The Add Procedure option is chosen from the Tools menu, which opens an Add Procedure dialog box as shown in the figure given below.
- The name of the procedure is typed in the Name textbox
- Under Type, Sub is selected to create a Sub procedure, Function to create a Function procedure or Property to create a Property procedure.
- Under Scope, Public is selected to create a procedure that can be invoked outside the module, or Private to create a procedure that can be invoked only from within the modulez

We can also create a new procedure in the current module by typing Sub ProcedureName, Function ProcedureName, or Property ProcedureName in the Code window. A Function procedure returns a value and a Sub Procedure does not return a value.

Function Procedures

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

The following function procedure calculates the third side or hypotenuse of a right triangle, where A and B are the other two sides. It takes two arguments A and B (of data type Double) and finally returns the results.

```
Function Hypotenuse (A As Double, B As Double) As Double
Hypotenuse = sqr (A^2 + B^2)
End Function
```

The above function procedure is written in the general declarations section of the Code window. A function can also be written by selecting the Add Procedure dialog box from the Tools menu and by choosing the required scope and type.

Property Procedures

A property procedure is used to create and manipulate custom properties. It is used to create read only properties for Forms, Standard modules and Class modules. Visual Basic provides three kind of property procedures- Property Let procedure that sets the value of a property, Property Get procedure that returns the value of a property, and Property Set procedure that sets the references to an object.

Review & Self Assessment Question

Q1-What is procedure?
Q2-What are the differences between procedure & function?
Q3-What do you mean by property procedure?
Q4-Write about sub procedure with example?

Further Readings

Visual basic by David I Schneider
Visual basic by Steven Holzner
Visual basic by Bill Sheldon
Visual basic by Billy Holls
Visual basic by Rob Windsor

UNIT: 8 - ARRAYS IN VISUAL BASIC

Contents

Introduction to Array

An array is a consecutive group of memory locations that all have the same name and the same type. To refer to a particular location or element in the array, we specify the array name and the array element position number.

The Individual elements of an array are identified using an index. Arrays have upper and lower bounds and the elements have to lie within those bounds. Each index number in an array is allocated individual memory space and therefore users must evade declaring arrays of larger size than required. We can declare an array of any of the basic data types including variant, user-defined types and object variables. The individual elements of an array are all of the same data type.

Declaring arrays

Arrays occupy space in memory. The programmer specifies the array type and the number of elements required by the array so that the compiler may reserve the appropriate amount of memory. Arrays may be declared as Public (in a code module), module or local. Module arrays are declared in the general declarations using keyword Dim or Private. Local arrays are declared in a procedure using Dim or Static. Array must be declared explicitly with keyword "As".

The Syntax for Declaring an array is :

[Dim | Private|Public|Static|Global] array name ([lower bound to] upperbound[,….][as data type].

There are two types of arrays in Visual Basic namely:

- **Fixed-size array:** The size of array always remains the same-size doesn't change during the program execution.
- **Dynamic array:** The size of the array can be changed at the run time-size changes during the program execution.

Fixed-sized Arrays

When an upper bound is specified in the declaration, a Fixed-array is created. The upper limit should always be within the range of long data type.

Declaring a fixed-array

Dim numbers (5) As Integer

In the above illustration, numbers is the name of the array, and the number 6 included in the parentheses is the upper limit of the array. The above declaration creates an array with 6 elements, with index numbers running from 0 to 5.

If we want to specify the lower limit, then the parentheses should include both the lower and upper limit along with the To keyword. An example for this is given below.

Dim numbers (1 To 6) As Integer

In the above statement, an array of 10 elements is declared but with indexes running from 1 to 6.

A public array can be declared using the keyword Public instead of Dim as shown below.

Public numbers (5) As Integer

Multidimensional Arrays

Arrays can have multiple dimensions. A common use of multidimensional arrays is to represent tables of values consisting of information arranged in rows and columns. To identify a particular table element, we must specify two indexes: The first (by convention) identifies the element's row and the second (by convention) identifies the element's column.

Tables or arrays that require two indexes to identify a particular element are called two dimensional arrays. Note that multidimensional arrays can have more than two dimensions. Visual Basic supports at least 60 array dimensions, but most people will need to use more than two or three dimensional-arrays.

The following statement declares a two-dimensional array 50 by 50 array within a procedure.

Dim AvgMarks (50, 50)

It is also possible to define the lower limits for one or both the dimensions as for fixed size arrays. An example for this is given here.

Dim Marks (101 To 200, 1 To 100)

An example for three dimensional-array with defined lower limits is given below.

Dim Details(101 To 200, 1 To 100, 1 To 100)

Static and dynamic arrays

Basically, you can create either static or dynamic arrays. Static arrays must include a fixed number of items, and this number must be known at compile time so that the compiler can set aside the necessary amount of memory. You create a static array using a Dim statement with a constant argument:

This is a static array.

Dim Names(100) As String

Visual Basic starts indexing the array with 0. Therefore, the preceding array actually holds 101 items.

Most programs don't use static arrays because programmers rarely know at compile time how many items you need and also because static arrays can't be resized during execution. Both these issues are solved by dynamic arrays. You declare and create dynamic arrays in two distinct steps. In general, you declare the array to account for its visibility (for example, at the beginning of a module if you want to make it visible by all the procedures of the module) using a Dim command with an empty pair of brackets. Then you create the array when you actually need it, using a ReDim statement:

```
An array defined in a BAS module (with Private scope)
Dim Customers() As String
...
Sub Main()
ere you create the array.
ReDim Customer(1000) As String
End Sub
```

If you're creating an array that's local to a procedure, you can do everything with a single ReDim statement:

```
Sub PrintReport()
 This array is visible only to the procedure.
ReDim Customers(1000) As String
...
End Sub
```

If you don't specify the lower index of an array, Visual Basic assumes it to be 0, unless an Option Base 1 statement is placed at the beginning of the module. My suggestion is this: Never use an Option Base statement because it makes code reuse more difficult. (You can't cut and paste routines without worrying about the current Option Base.) If you want to explicitly use a lower index different from 0, use this syntax instead:

```
ReDim Customers(1 To 1000) As String
```

Dynamic Array

Dynamic arrays can be re-created at will, each time with a different number of items. When you re-create a dynamic array, its contents are reset to 0 (or to an empty string) and you lose the data it contains. If you want to resize an array without losing its contents, use the ReDim Preserve command:

```
        ReDim Preserve Customers(2000) As String
```

When you're resizing an array, you can't change the number of its dimensions nor the type of the values it contains. Moreover, when you're using ReDim Preserve on a multidimensional array, you can resize only its last dimension:

```
ReDim Cells(1 To 100, 10) As Integer
...
ReDim Preserve Cells(1 To 100, 20) As Integer ' This works.
ReDim Preserve Cells(1 To 200, 20) As Integer ' This doesn't.
```

Finally, you can destroy an array using the Erase statement. If the array is dynamic, Visual Basic releases the memory allocated for its elements (and you can't read or write them any longer); if the array is static, its elements are set to 0 or to empty strings.
You can use the LBound and UBound functions to retrieve the lower and upper indices. If the array has two or more dimensions, you need to pass a second argument to these functions to specify the dimension you need:

```
Print LBound(Cells, 1) ' Displays 1, lower index of 1st dimension
Print LBound(Cells) ' Same as above
Print UBound(Cells, 2) ' Displays 20, upper index of 2nd dimension
Evaluate total number of elements.
NumEls = (UBound(Cells) _ LBound(Cells) + 1) * _
(UBound(Cells, 2) _ LBound(Cells, 2) + 1)
```

Arrays within UDTs
UDT structures can include both static and dynamic arrays. Here's a sample structure that contains both types:

```
Type MyUDT
StaticArr(100) As Long
DynamicArr() As Long
End Type
...
Dim udt As MyUDT
' You must DIMension the dynamic array before using it.
ReDim udt.DynamicArr(100) As Long
' You don't have to do that with static arrays.
udt.StaticArr(1) = 1234
```

The memory needed by a static array is allocated within the UDT structure; for example, the StaticArr array in the preceding code snippet takes exactly 400 bytes. Conversely, a dynamic array in a UDT takes only 4 bytes, which form a pointer to the memory area where the actual data is stored. Dynamic arrays are advantageous when each individual UDT variable might host a different number of array items. As with all dynamic arrays, if you don't dimension a dynamic array within a UDT before accessing its items, you get an error 9—"**Subscript out of range**."
Variables of different data types when combined as a single variable to hold several related information’s is called a User-Defined data type.
A Type statement is used to define a user-defined type in the General declaration section of a form or module. User-defined data types can only be private in form while in standard modules can be public or private. An example for a user defined data type to hold the product details is as given below.

```
Private Type ProductDetails
ProdID as String
ProdName as String
Price as Currency
End Type
```

CONTROL ARRAYS

We have seen many examples of the usefulness of subscripted variables. They are essential for writing concise solutions to many programming problems. Because of the great utility that subscripts provide, Visual Basic also provides a means of constructing arrays of text boxes, labels, command buttons, and so on. Because text boxes, labels, and command buttons are referred to generically in Visual Basic as controls, arrays of these objects are called **control arrays**.

Unlike variable arrays, which can only be created by Dim and ReDim statements once a program is running, at least one element of a control array must be created when the form is designed. The remaining elements can be created either during form design, or, perhaps more typically, with the Load statement when the program is run.

To create the first element of an array of text boxes, create an ordinary text box, then access the Properties window, and select the property called Index. By default this property is blank. Change the Index property to 0 (zero). Your text box is now the first element in a subscripted control array. If the name of a text box is txtBox and its Index property is 0, then assigning a value to the text box during run time requires a statement of the form

```
txtBox(0).Text = value
```

Arrays are not of much use if they contain only a single element. To create additional elements of the txtBox( ) control array during form design, make sure that the element you just created is active by clicking on it. Next, press Ctrl+C (or open the Edit menu and select Copy). Visual Basic has now recorded all the properties associated with txtBox(0) and is ready to reproduce as many copies as you desire. To create a copy, press Ctrl+V (or open the Edit menu and select Paste). The copy of txtBox(0) appears in the upper-left corner of the form. The value of the Index property for this new text box is 1; thus the text box is referred to as txtBox(1). Move this text box to the desired position. Press Ctrl+V again and another copy of txtBox(0) appears in the upper left corner of the form. Its Index property is 2. Move txtBox(2) to an appropriate position. Continue copying txtBox(0) in this manner until all desired controls have been created.

It is important to note that all property settings of txtBox(0) are being passed (as default settings) to the other elements of the txtBox( ) control array, with the exception of the Index, Top, and Left settings. Thus, as a matter of efficiency, before you begin copying txtBox(0), set all properties that you want carried over to all elements of txtBox( ). For example, if you desire to have the Text property blank for all txtBox( ) elements, set the Text property of txtBox(0) to (blank) before starting the copying process.

The preceding discussion gave a process for creating an array of text boxes. This same process applies to creating arrays of labels or any other control. In summary, the following steps create an array of controls while designing a form:

1. Add one instance of the desired control to the form.
2. Set the Index property of this control to 0.
3. Set any other properties of the control that will be common to all elements of the array.
4. Click on the control and then press Ctrl+C to prepare to make a copy of the control.
5. Press Ctrl+V to create a copy of the control. Position this control as desired.
6. Repeat Step 5 until all desired elements of the control array have been created.

Example:

```
Private Sub Form_Load()
Dim depNum As Integer
For depNum = 0 To 4
lblDepart(depNum).Caption = “Department” & Str(depNum + 1)
Next depNum
End Sub
Private Sub cmdCompute_Click()
Dim depNum As Integer, sales As Single
sales = 0
For depNum = 0 To 4
sales = sales + Val(txtSales(depNum).Text)
Next depNum
picTotal.Cls
picTotal.Print “Total sales were ”; FormatCurrency(sales)
End Sub
```

CREATING CONTROL ARRAYS AT RUN TIME

We have discussed the process for creating an entire control array while designing a form, that is, at design time. However, copying and positioning control array elements can become tedious if the number of elements is large. Also, the actual number of elements needed in a control array may not be known until a response from the user is processed at run time. In light of these concerns, Visual Basic provides a solution via the Load statement that only requires us to create the first element of a control array during design time. The remaining elements are then created as needed at run time.

Before we discuss creating arrays at run time, we must consider a preliminary topic—the Left, Top, Width, and Height properties of controls. These properties specify the location and size of controls.

The standard unit of measurement in Visual Basic is called a **twip**. There are about 1440 twips to the inch. At design time, when a control is active the two panels on the right side of the toolbar give the location and size of the control, respectively. The figure shows an active text box, named Text1. The first panel says that the left side of the text box is 960 twips from the left side of the form, and the top of the text box is 720 twips down from the title bar of the form. In terms of properties, Text1.Left is 960, and Text1.Top is 720. Similarly, the numbers 1935 and 975 in the second panel give the width and height of the text box in twips. In terms of properties, Text1.Width is 1935 and Text1.Height is 975. The figure shows the meanings of these four properties.

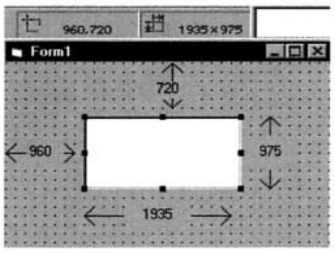
The location and size properties of a control can be altered at run time with statements such as

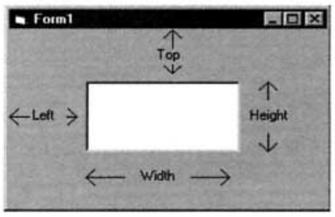
```
Text1.Left = 480
```

which moves the text box to the left or

```
Text2.Top = Text1.Top + 1.5 * Text1.Height
```

which places Text2 a comfortable distance below Text1. As a result of the second statement, the distance between the two text boxes will be half the height of Text1.





If controlName is the name of a control whose Index property was assigned a value during form design (thus creating the beginnings of a control array) and num is a whole number that has not yet been used as an index for the controlName( ) array, then the statement

```
Load controlName(num)
```

copies all the properties of controlName(0), including the Top and Left properties, and creates the element controlName(num) of the controlName( ) array. The only property of controlName(num) that may differ from that of controlName(0) is the Visible property. The Load statement always sets the visible property of the created element to False. After creating a new element of a control array, you will want to adjust the Top and Left properties so that the new element has its own unique location on the form, and then set the visible property of the new element to True.

SORTING AND SEARCHING

A **sort** is an algorithm for ordering an array. Of the many different techniques for sorting an array we discuss two, the **bubble sort** and the **Shell sort**. Both require the interchange of values stored in a pair of variables. If var1, var2, and temp are all variables of the same data type (such as all String), then the statements

```
temp = var1
var1 = var2
var2 = temp
```

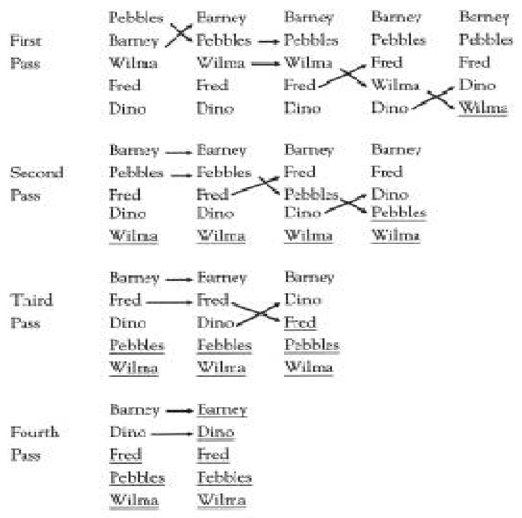
assign var1's value to var2, and var2's value to var1.

BUBBLE SORT

The bubble sort is an algorithm that compares adjacent items and swaps those that are out of order. If this process is repeated enough times, the list will be ordered. Let's carry out this process on the list Pebbles, Barney, Wilma, Fred, Dino. The steps for each pass through the list are as follows:

1. Compare the first and second items. If they are out of order, swap them.
2. Compare the second and third items. If they are out of order, swap them.
3. Repeat this pattern for all remaining pairs. The final comparison and possible swap are between the second-to-last and last elements.

The first time through the list, this process is repeated to the end of the list. This is called the first pass. After the first pass, the last item (Wilma) will be in its proper position. Therefore, the second pass does not have to consider it and so requires one less comparison. At the end of the second pass, the last two items will be in their proper position. (The items that must have reached their proper position have been underlined.) Each successive pass requires one less comparison. After four passes, the last four items will be in their proper positions, and hence, the first will be also.


First Pass
Pebbles Barney Wilma Fred Dino
Barney Pebbles Wilma Fred Dino
Barney Pebbles Wilma Fred Dino
Barney Pebbles Fred Wilma Dino
Barney Pebbles Fred Dino Wilma
Second Pass
Barney Pebbles Fred Dino Wilma
Barney Pebbles Fred Dino Wilma
Barney Fred Pebbles Dino Wilma
Barney Fred Dino Pebbles Wilma
Third Pass
Barney Fred Dino Pebbles Wilma
Barney Fred Dino Pebbles Wilma
Barney Dino Fred Pebbles Wilma
Fourth Pass
Barney Dino Fred Pebbles Wilma
Barney Dino Fred Pebbles Wilma

SEARCHING

Suppose we had an array of 1000 names in alphabetical order and wanted to locate a specific person in the list. One approach would be to start with the first name and consider each name until a match was found. This process is called a **sequential search**. We would find a person whose name begins with "A" rather quickly, but 1000 comparisons might be necessary to find a person whose name begins with "Z." For much longer lists, searching could be a time-consuming matter. However, when the list has already been sorted into either ascending or descending order, there is a method, called a **binary search**, which shortens the task considerably.

Let us refer to the sought item as quarry. The binary search looks for quarry by determining in which half of the list it lies. The other half is then discarded, and the retained half is temporarily regarded as the entire list. The process is repeated until the item is found. A flag can indicate if quarry has been found.

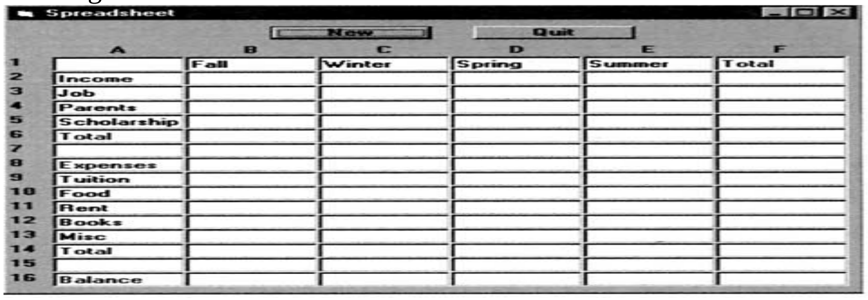
The algorithm for a binary search of an ascending list is as follows (Figure 6-8 shows the flowchart for a binary search):

1. At each stage, denote the subscript of the first item in the retained list by first and the subscript of the last item by last. Initially, the value of first is 1, the value of last is the number of items in the list, and the value of flag is False.
2. Look at the middle item of the current list, the item having the subscript middle = Int((first + last) / 2).
3. If the middle item is quarry, then flag is set to True and the search is over.
4. If the middle item is greater than quarry, then quarry should be in the first half of the list. So the subscript of quarry must lie between first and middle [minus] 1. That is, the new value of last is middle – 1.
5. If the middle item is less than quarry, then quarry should be in the second half of the list of possible items. So the subscript of quarry must lie between middle + 1 and last. That is, the new value of first is middle + 1.
6. Repeat Steps 2 through 5 until quarry is found or until the halving process uses up the entire list.In the second case, quarry was not in the original list.

THE USER INTERFACE

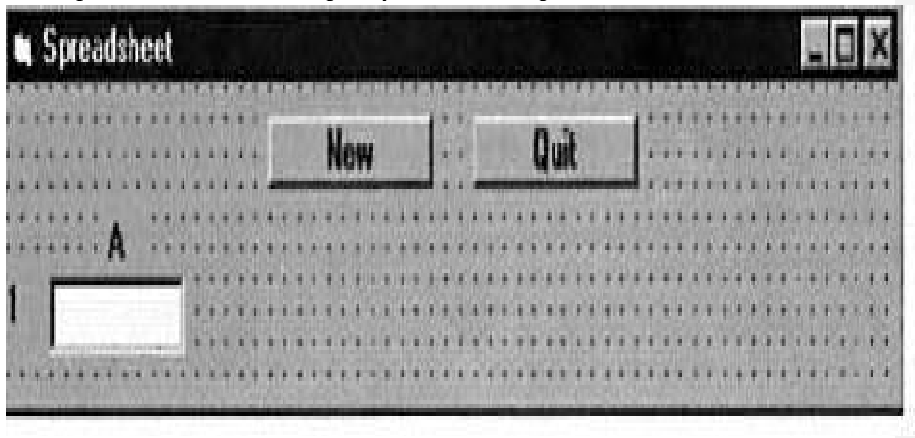
Each cell in the spreadsheet will be an element of a text box control array. A control array of labels is needed for the numeric labels to the left of each row and another control array of labels for the alphabetic labels at the top of each column. Finally, two command buttons are required.

The task of controlling which cells the user can edit will be handled by a GotFocus event. The task of updating the totals will be handled by a LostFocus event.figure shows one possible form design with all control elements loaded. For this application of a spreadsheet, the headings that have been assigned to cells in rows 1, 2, 6, 8, 14, and 16 are fixed; the user will not be allowed to edit them. The other entries in column A, the category names, may be edited by the user, but we have provided the set from Figure 6-9 as the default.



	A	B	C	D	E	F
1		Fall	Winter	Spring	Summer	Total
2	Income					
3	Job					
4	Parents					
5	Scholarship					
6	Total					
7						
8	Expenses					
9	Tuition					
10	Food					
11	Rent					
12	Books					
13	Misc					
14	Total					
15						
16	Balance					

Because processing the totals for the spreadsheet involves adding columns and rows of cells, coding is simplified by using a control array of text boxes so that an index in a For...Next loop can step through a set of cells. A two-dimensional array of text boxes seems natural for the spreadsheet. Unfortunately, only a single index is available for control arrays in Visual Basic. However, a single dimensional array of text boxes can be used without much difficulty if we define a function Indx that connects a pair of row (1 to 16) and column (1 to 6) values to a unique index (1 to 96) value. An example of such a rule would be Indx(row,column)=(row–1)*6+column. Successive values of this function are generated by going from left to right across row 1, then left to right across row 2, and so on.
A solution to the spreadsheet problem that uses one control array of text boxes and two control arrays of labels follows. The text box control array txtCell( ) provides the 96 text boxes needed for the spreadsheet cells. Because the proposed Indx function advances by one as we move from left to right across a row of cells, the cells must be positioned in this order as they are loaded. The label control array lblRowLab( ) provides a label for each of the rows of cells, while label control array lblColLab( ) provides a label for each column of cells. The figure shows the layout of the form at design time.The Height and Width properties given for the text box will assure enough room on the screen for all 96 cells. These dimensions can be obtained by creating a normal size text box, then reducing its width by one set of grid marks and its height by two sets of grid marks.



Review & Self Assessment Question

Q1- What is array?
Q2- What are the differences between fixed size array and Dynamic Array?
Q3-What is multi Dimensional Array?
Q4-What is dynamic Array?
Q5-What do you mean by control array?

Further Readings

Visual basic by David I Schneider

Visual basic by Steven Holzner
Visual basic by Bill Sheldon
Visual basic by Billy Holls
Visual basic by Rob Windsor

UNIT-9 USER DEFINED DATA TYPE

Contents

The user defined data type can be declared with a variable using the Dim statement as in any other variable declaration statement. An array of these user-defined data types can also be declared. An example to consolidate these two features is given below.

```
Dim ElectronicGoods as ProductDetails ' One Record
Dim ElectronicGoods(10) as ProductDetails ' An array of 11 records
```

User Define Data Type

A User-Defined data type can be referenced in an application by using the variable name in the procedure along with the item name in the Type block. Say, for example if the text property of a TextBox namely text1 is to be assigned the name of the electronic good, the statement can be written as given below.

```
Text1.Text = ElectronicGoods.ProdName
```

If the same is implemented as an array, then the statement becomes

```
Text1.Text = ElectronicGoods(i).ProdName
```

User-defined data types can also be passed to procedures to allow many related items as one argument.

```
Sub ProdData( ElectronicGoods as ProductDetails)
Text1.Text = ElectronicGoods.ProdName
Text1.Text = ElectronicGoods.Price
End Sub
```

Data Type Conversion

Visual Basic functions either to convert a string into an integer or vice versa and many more conversion functions. A complete listing of all the conversion functions offered by Visual Basic is elucidated below.

Conversion To	Function
Boolean	Cbool
Byte	Cbyte
Currency	Ccur
Date	Cdate
Decimals	Cdec
Double	CDbl

Integer	Cint
Long	CLng
Single	CSng
String	CStr
Variant	Cvar
Error	CVErr

A conversion function should always be placed at the right hand side of the calculation statement.

Review & Self Assessment Question

Q1- What do you mean by user defined data type?
Q2- Explain data type conversion?

Further Readings

Visual basic by David I Schneider
Visual basic by Steven Holzner
Visual basic by Bill Sheldon
Visual basic by Billy Holls
Visual basic by Rob Windsor

UNIT: 10 - VISUAL BASIC BUILT-IN FUNCTIONS

Contents

Introduction

A function is similar to a procedure but the main purpose of the function is to accept a certain input from the user and return a value which is passed on to the main program to finish the execution.

There are two types of functions, the built-in functions (or internal functions) and the functions created by the programmers.

The syntax of a function is

FunctionName (arguments)

The arguments are values that are passed on to the function.

Many built-in functions are offered by Visual Basic fall under various categories. These functions are procedures that return a value. The functions fall into the following basic categories that will be discussed in the following sections at length.

- Date and Time Functions
- Format Function
- String Functions
    - Not only information in the specific Date data type, it also provides a lot of date- and time-related functions. These functions are very important in all business applications and deserve an in-depth look. Date and Time are internally stored as numbers in does Visual Basic let you store date and time Visual Basic. The decimal points represents the time between 0:00:00 and 23:59:59 hours inclusive.
    - The system's current date and time can be retrieved using the Now, Date and Time functions in Visual Basic. The Now function retrieves the date and time, while Date function retrieves only date and Time function retrieves only the time.
    - To display both the date and time together a message box is displayed use the statement given below.
    - MsgBox "The current date and time of the system is" & Now
    - Here & is used as a concatenation operator to concentrate the string and the Now function. Selective portions of the date and time value can be extracted using the below listed functions.

Function	Extracted Portion

Year ( )	Year (Now)
Month ( )	Month (Now)
Day ( )	Day (Now)
WeekDay ( )	WeekDay (Now)
Hour ( )	Hour (Now)
Minute ( )	Minute (Now)
Second ( )	Second (Now)

- The calculation and conversion functions related to date and time functions are listed below.

Function	Description
DateAdd ( )	Returns a date to which a specific interval has been added
DateDiff ( )	Returns a Long data type value specifying the interval between the two values
DatePart ( )	Returns an Integer containing the specified part of a given date
DateValue ( )	Converts a string to a Date
TimeValue ( )	Converts a string to a time
DateSerial ( )	Returns a date for specified year, month and day

- DateDiff Function
- The DateDiff function returns the intervals between two dates in terms of years, months or days. The syntax for this is given below.
- DateDiff (interval, date1, date2[, firstdayofweek[, firstweekofyear]])
- Format Function
- The format function accepts a numeric value and converts it to a string in the format specified by the format argument. The syntax for this is given below.
- Format (expression[, format[,firstdayofweek[,firstweekofyear]]])
- The Format function syntax has these parts:

Part	Description

Expression	Required any valid expression
Format	Optional. A valid named or user-defined format expression.
Firstdayofweek	Optional. A contant that specifies the first day of the week.
Firstweekofyear	Optional. A contant that specifies the first week of the year

Review & Self Assessment Question

Q1- What are the differences between Built in Function and Function created by Programmer?
Q2-Explain Date and Time function with example?
Q3-What is String Function?

Further Readings

Visual basic by David I Schneider

Visual basic by Steven Holzner

Visual basic by Bill Sheldon

Visual basic by Billy Holls

Visual basic by Rob Windsor

UNIT: 11-ERROR HANDLING

Contents

Error Handling

Error Handling enables programmers to write clearer, more robust, more fault-tolerant programs. Error handling enables the programmer to attempt to recover (i.e., continue executing) from infrequent fatal errors rather than letting them occur and suffering the consequences (such as loss of application data). If an error is severe and recovery is not possible, the program can be exited "gracefully"-all files can be closed and notification can be given that the program is terminating. The recovery code is called an error handler.

Error handling is designed for dealing with synchronous errors such as an attempt to divide by 0 (that occurs as the program executes the divide instruction). Other common examples of synchronous errors are memory exhaustion, an out-of-bound array index, and arithmetic overflow. Error handling provides the programmer with a disciplined set of capabilities for dealing with these types of errors.

Error-handling code varies in nature and amount among software systems depending on the application and whether or not the software is a product for release. Products tend to contain much more error-handling code than is contained in "casual" software.

Usually, error-handling code is interspersed throughout a program's code. Errors are dealt with the places in the code where errors are likely to occur. The advantage of this approach is that a programmer reading the code can see the error handling in the immediate vicinity of the code and determine if the proper error handling has been implemented.

The problem with the scheme is that code in a sense becomes "polluted" with error handling. It becomes difficult for a programmer concerned with the application itself to read the code and determine if the code is working is correctly. Error handling often makes the code more difficult to understand and maintain.

When Error Handling should be used Error handling should be used to process only exceptional situations, despite the fact that there is nothing to prevent that programmer from using errors as an alternate form of program control.

• No matter how hard we try, errors do creep into our programs. These errors can be grouped into three categories:

1. Syntax errors
2. Run-time errors
3. Logic errors

• **Syntax errors** occur when you mistype a command or leave out an expected phrase or argument. Visual Basic detects these errors as they occur and even provides help in correcting them. You cannot run a Visual Basic program until all syntax errors have been corrected.
• **Run-time errors** are usually beyond your program's control. Examples include: when a variable takes on an unexpected value (divide by zero), when a drive door is left open, or when a file is not found. Visual Basic allows you to trap such errors and make attempts to correct them.
• Logic errors are the most difficult to find. With logic errors, the program will usually run, but will produce incorrect or unexpected results. The Visual Basic debugger is an aid in detecting logic errors.

Some ways to minimize errors:

- Design your application carefully. More design time means less debugging time.
- Use comments where applicable to help you remember what you were trying to do.
- Use consistent and meaningful naming conventions for your variables, objects, and procedures.

• Run-time errors are trappable. That is, Visual Basic recognizes an error has occurred and enables you to trap it and take corrective action. If an error occurs and is not trapped, your program will usually end in a rather unceremonious manner.
• Error trapping is enabled with the On Error statement:

On Error GoTo errlabel

Yes, this uses the dreaded GoTo statement! Any time a run-time error occurs following this line, program control is transferred to the line labeled errlabel. Recall a labeled line is simply a line with the label followed by a colon (:).
• The best way to explain how to use error trapping is to look at an outline of an example procedure with error trapping.

```
Sub SubExample ( )
[Declare variables, ...]
On Error GoTo HandleErrors
[Procedure code]
Exit Sub
HandleErrors:
Error handling code]
End Sub
```

Once you have set up the variable declarations, constant definitions, and any other procedure preliminaries, the On Error statement is executed to enable error trapping. Your normal procedure code follows this statement.

The error handling code goes at the end of the procedure, following the HandleErrors statement label. This is the code that is executed if an error is encountered anywhere in the Sub procedure. Note you must exit (with Exit Sub) from the code before reaching the HandleErrors line to avoid inadvertent execution of the error handling code.

• Since the error handling code is in the same procedure where an error occurs, all variables in that procedure are available for possible corrective action. If at some time in your procedure, you want to turn off error trapping, that is done with the following statement:

On Error GoTo

• Once a run-time error occurs, we would like to know what the error is and attempt to fix it. This is done in the error handling code.

• Visual Basic offers help in identifying run-time errors. The Err object returns, in its Number property (Err.Number), the number associated with the current error condition. (The Err function has other useful properties that we won't cover here - consult on-line help for further information.) The Error() function takes this error number as its argument and returns a string description of the error. Consult on-line help for Visual Basic run-time error numbers and their descriptions.

• Once an error has been trapped and some action taken, control must be returned to your application. That control is returned via the Resume statement. There are three options:

Resume Lets you retry the operation that caused the error. That is, control is returned to the line where the error occurred. This could be dangerous in that, if the error has not been corrected (via code or by the user), an infinite loop between the error handler and the procedure code may result.

Resume Next Program control is returned to the line immediately following the line where the error occurred.

Resume label Program control is returned to the line labeled label.

• Be careful with the Resume statement. When executing the error handling portion of the code and the end of the procedure is encountered before a Resume, an error occurs. Likewise, if a Resume is encountered outside of the error handling portion of the code, an error occurs.

• Development of an adequate error handling procedure is application dependent. You need to know what type of errors you are looking for and what corrective actions must be taken if these errors are encountered. For example, if a 'divide by zero' is found, you need to decide whether to skip the operation or do something to reset the offending denominator.

• What we develop here is a generic framework for an error handling procedure. It simply informs the user that an error has occurred, provides a description of the error, and allows the user to Abort, Retry, or Ignore. This framework is a good starting point for designing custom error handling for your applications.

• The generic code (begins with label HandleErrors) is:

```
HandleErrors:
Select Case MsgBox(Error(Err.Number), vbCritical +
vbAbortRetryIgnore, "Error Number" + Str(Err.Number))
Case vbAbort
  Resume ExitLine
Case vbRetry
  Resume
Case vbIgnore
  Resume Next
End Select
ExitLine:
Exit Sub
```

Let’s look at what goes on here. First, this routine is only executed when an error occurs. A message box is displayed, using the Visual Basic provided error description [Error(Err.Number)] as the message, uses a critical icon along with the Abort, Retry, and Ignore buttons, and uses the error number [Err.Number] as the title. This message box returns a response indicating which button was selected by the user.

If **Abort** is selected, we simply exit the procedure. (This is done using a Resume to the line labeled ExitLine. Recall all error trapping must be terminated with a Resume statement of some kind.)

If **Retry** is selected, the offending program line is retried (in a real application, you or the user would have to change something here to correct the condition causing the error).

If **Ignore** is selected, program operation continues with the line following the error causing line.

• To use this generic code in an existing procedure, you need to do three things:

1. Copy and paste the error handling code into the end of your procedure.
2. Place an Exit Sub line immediately preceding the HandleErrors labeled line.
3. Place the line, On Error GoTo HandleErrors, at the beginning of your procedure.

For example, if your procedure is the SubExample seen earlier, the modified code will look like this:

```
Sub SubExample()
  .
  . [Declare variables, ...]
  .
On Error GoTo HandleErrors
  .
  . [Procedure code]
  .
Exit Sub
```

HandleErrors:

```
Select Case MsgBox(Error(Err.Number), vbCritical +
vbAbortRetryIgnore, "Error Number" + Str(Err.Number))
Case vbAbort
```

```
  Resume ExitLine
Case vbRetry
  Resume
Case vbIgnore
  Resume Next
End Select
ExitLine:
Exit Sub
End Sub
```

Again, this is a very basic error-handling routine. You must determine its utility in your applications and make any modifications necessary. Specifically, you need code to clear error conditions before using the Retry option.

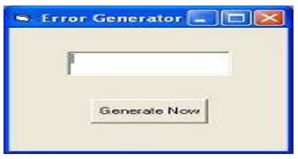
• One last thing. Once you've written an error handling routine, you need to test it to make sure it works properly. But, creating run-time errors is sometimes difficult and perhaps dangerous. Visual Basic comes to the rescue! The Visual Basic Err object has a method (Raise) associated with it that simulates the occurrence of a run-time error. To cause an error with value Number, use: Err.Raise Number

• We can use this function to completely test the operation of any error handler we write. Don’t forget to remove the Raise statement once testing is completed, though! And, to really get fancy, you can also use Raise to generate your own ‘application-defined’ errors. There are errors specific to your application that you want to trap.

• To clear an error condition (any error, not just ones generated with the Raise method), use the method Clear: Err.Clear

1. Start a new project. Add a text box and a command button.
2. Set the properties of the form and each control:

Form1:
BorderStyle - 1-Fixed Single
Caption - Error Generator
Name frmError
Command1:
Caption - Generate Error
Default - True
Name - cmdGenError
Text1:
Name - txtError
Text - [Blank]
The form should look something like this:



3. Attach this code to the cmdGenError_Click event.

```
Private Sub cmdGenError_Click()
On Error GoTo HandleErrors
Err.Raise Val(txtError.Text)
Err.Clear
Exit Sub
HandleErrors:
Select Case MsgBox(Error(Err.Number), vbCritical +
vbAbortRetryIgnore, "Error Number" + Str(Err.Number))
Case vbAbort
  Resume ExitLine
Case vbRetry
  Resume
Case vbIgnore
  Resume Next
End Select
ExitLine:
Exit Sub
End Sub
```

In this code, we simply generate an error using the number input in the text box. The generic error handler then displays a message box which you can respond to in one of three ways.

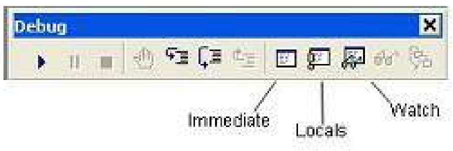
4. Save your application. Try it out using some of these typical error numbers (or use numbers found with on-line help). Notice how program control changes depending on which button is clicked.

Error Number	**Error Description**
6	Overflow
9	Subscript out of range
11	Division by zero
13	Type mismatch
16	Expression too complex
20	Resume without error
52	Bad file name or number
53	File not found
55	File already open
61	Disk full
70	Permission denied
92	For loop not initialized

• We now consider the search for, and elimination of, logic errors. These are errors that don’t prevent an application from running, but cause incorrect or unexpected results. Visual Basic provides an excellent set of debugging tools to aid in this search.

• Debugging a code is an art, not a science. There are no prescribed processes that you can follow to eliminate all logic errors in your program. The usual approach is to eliminate them as they are discovered.

• What we'll do here is present the debugging tools available in the Visual Basic environment (several of which appear as buttons on the toolbar) and describe their use with an example. You, as the program designer, should select the debugging approach and tools you feel most comfortable with.
• The interface between your application and the debugging tools is via three different debug windows: the Immediate Window, the Locals Window, and the Watch Window. These windows can be accessed from the View menu (the Immediate Window can be accessed by pressing Ctrl+G). Or, they can be selected from the Debug Toolbar (accessed using the Toolbars option under the View menu):
•

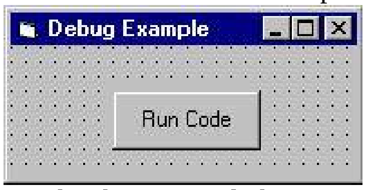


• All debugging using the debug windows is done when your application is in break mode. You can enter break mode by setting breakpoints, pressing Ctrl+Break, or the program will go into break mode if it encounters an untrapped error or a Stop statement.
• Once in break mode, the debug windows and other tools can be used to:

1. Determine values of variables
2. Set breakpoints
3. Set watch variables and expressions
4. Manually control the application
5. Determine which procedures have been called
6. Change the values of variables and properties

Example - Debugging

1. Unlike other examples, we'll do this one as a group. It will be used to demonstrate use of the debugging tools.
2. The example simply has a form with a single command button. The button is used to execute some code. We won't be real careful about proper naming conventions and such in this example.



3. The code attached to this button's Click event is a simple loop that evaluates a function at several values.

```
Private Sub Command1_Click()
Dim X As Integer, Y As Integer
X = 0
Do
Y = Fcn(X)
X = X + 1
```

```
Loop While X <= 20
End Sub
```

This code begins with an X value of 0 and computes the Y value using the general integer function Fcn. It then increments X by 1 and repeats the Loop. It continues looping While X is less than or equal to 20. The function Fcn is computed using:

```
Function Fcn(X As Integer) As Integer
Fcn = CInt(0.1 * X ^ 2)
End Function
```

Admittedly, this code doesn’t do much, especially without any output, but it makes a good example for looking at debugger use. Set up the application and get ready to try debugging.

• There are several debugging tools available for use in Visual Basic. Access to these tools is provided with both menu options and buttons on the Debug toolbar. These tools include breakpoints,watchpoints,calls,step
• The simplest tool is the use of direct prints to the immediate window.
• Printing to the Immediate Window:

You can print directly to the immediate window while an application is running. Sometimes, this is all the debugging you may need. A few carefully placed print statements can sometimes clear up all logic errors, especially in small applications.

To print to the immediate window, use the Print method:

Debug.Print [List of variables separated by commas or semi-colons]

• Debug.Print Example:

1. Place the following statement in the Command1_Click procedure after the line calling the general procedure Fcn:
   Debug.Print X; Y
   and run the application.
2. Examine the immediate window. Note how, at each iteration of the loop, the program prints the value of X and Y. You could use this information to make sure X is incrementing correctly and that Y values look acceptable.
3. Remove the Debug.Print statement.

• Breakpoints:

In the above examples, the program ran to completion before we could look at the debug window. In many applications, we want to stop the application while it is running, examine variables and then continue running. This can be done with breakpoints.

Break Point

A breakpoint is a line in the code where you want to stop (temporarily) the execution of the program, that is force the program into break mode. To set a breakpoint, put the cursor in the line of code you want to break on.

Then, press <F9> or click the Breakpoint button on the toolbar or select Toggle Breakpoint from the Debug menu. The line will be highlighted.
When you run your program, Visual Basic will stop when it reaches lines with breakpoints and allow you to use the immediate window to check variables and expressions. To continue program operation after a breakpoint, press <F5>, click the Run button on the toolbar, or choose Start from the Run menu.
You can also change variable values using the immediate window. Simply type a valid Basic expression. This can sometimes be dangerous, though, as it may change program operation completely.
• Breakpoint Example:
1. Set a breakpoint on the X = X + 1 line in the sample program. Run the program.
2. When the program stops, display the immediate window and type the following line:
Print X;Y
3. The values of these two variables will appear in the debug window. You can use a question mark (?) as shorthand for the command Print, if you'd like. Restart the application. Print the new variable values.
Try other breakpoints if you have time. Once done, all breakpoints can be cleared by Ctrl+Shift+<F9> or by choosing Clear All Breakpoints from the Debug menu. Individual breakpoints can be toggled using <F9> or the Breakpoint button on the toolbar.
• Viewing Variables in the Locals Window:

The locals window shows the value of any variables within the scope of the current procedure. As execution switches from procedure to procedure, the contents of this window changes to reflect only the variables applicable to the current procedure. Repeat the above example and notice the values of X and Y also appear in the locals window.
• Watch Expressions:

The Add Watch option on the Debug menu allows you to establish watch expressions for your application. Watch expressions can be variable values or logical expressions you want to view or test. Values of watch expressions are displayed in the watch window.
In break mode, you can use the Quick Watch button on the toolbar to add watch expressions you need. Simply put the cursor on the variable or expression you want to add to the watch list and click the Quick Watch button.
Watch expressions can be edited using the Edit Watch option on the Debug menu.
• Watch Expression Example:
1. Set a breakpoint at the X = X + 1 line in the example.

2. Set a watch expression for the variable X. Run the application. Notice X appears in the watch window. Every time you re-start the application, the value of X changes.
3. At some point in the debug procedure, add a quick watch on Y. Notice it is now in the watch window.
4. Clear the breakpoint. Add a watch on the expression: X = Y. Set Watch Type to ‘Break When Value Is True.’ Run the application. Notice it goes into break mode and displays the watch window whenever X = Y. Delete this last watch expression.

• Call Stack:

Selecting the Call Stack button from the toolbar (or pressing Ctrl+L or selecting Call Stack from the View menu) will display all active procedures, that is those that have not been exited.
Call Stack helps you unravel situations with nested procedure calls to give you some idea of where you are in the application.

• Call Stack Example:

1. Set a breakpoint on the Fcn = Cint() line in the general function procedure. Run the application. It will break at this line.
2. Press the Call Stack button. It will indicate you are currently in the Fcn procedure which was called from the Command1_Click procedure. Clear the breakpoint.

• Single Stepping (Step Into):

While at a breakpoint, you may execute your program one line at a time by pressing <F8>, choosing the Step Into option in the Debug menu, or by clicking the Step Into button on the toolbar.
This process is single stepping. It allows you to watch how variables change (in the locals window) or how your form changes, one step at a time.
You may step through several lines at a time by using Run To Cursor option. With this option, click on a line below your current point of execution. Then press Ctrl+<F8> (or choose Run To Cursor in the Debug menu). the program will run through every line up to the cursor location, then stop.

• Step Into Example:

1. Set a breakpoint on the Do line in the example. Run the application.
2. When the program breaks, use the Step Into button to single step through the program.
3. At some point, put the cursor on the Loop While line. Try the Run To Cursor option (press Ctrl+<F8>).

•Procedure Stepping (Step Over):

While single stepping your program, if you come to a procedure call you know functions properly, you can perform procedure stepping. This simply executes the entire procedure at once, rather than one step at a time.
To move through a procedure in this manner, press Shift+<F8>, choose Step Over from the Debug menu, or press the Step Over button on the toolbar.
• Step over Example:

1. Run the previous example. Single step through it a couple of times.
2. One time through, when you are at the line calling the Fcn function, press the Step Over button. Notice how the program did not single step through the function as it did previously.

• Function Exit (Step Out):
While stepping through your program, if you wish to complete the execution of a function you are in, without stepping through it line-by-line, choose the Step Out option. The function will be completed and you will be returned to the procedure accessing that function.
To perform this step out, press Ctrl+Shift+<F8>, choose Step Out from the Debug menu, or press the Step Out button on the toolbar. Try this on the previous example.
• We've looked at each debugging tool briefly. Be aware this is a cursory introduction. Use the on-line help to delve into the details of each tool described. Only through lots of use and practice can you become a proficient debugger. There are some guidelines to doing a good job, though.
• My first suggestion is: keep it simple. Many times, you only have one or two bad lines of code. And you, knowing your code best, can usually quickly narrow down the areas with bad lines. Don't set up some elaborate debugging procedure if you haven't tried a simple approach to find your error(s) first. Many times, just a few intelligently-placed Debug.Print statements or a few examinations of the immediate and locals windows can solve your problem.
• A tried and true approach to debugging can be called Divide and Conquer. If you're not sure where your error is, guess somewhere in the middle of your application code. Set a breakpoint there. If the error hasn't shown up by then, you know it's in the second half of your code. If it has shown up, it's in the first half. Repeat this division process until you've narrowed your search.
• And, of course, the best debugging strategy is to be careful when you first design and write your application to minimize searching for errors later.

Review & Self Assessment Question

Q1-Describe Error Handling?
Q2-What do you mean by logical error?
Q3-What are the differences between Syntax error and Runtime Error?
Q4-What is Break point? Explain it.

Further Readings

Visual basic by David I Schneider
Visual basic by Steven Holzner
Visual basic by Bill Sheldon
Visual basic by Billy Holls
Visual basic by Rob Windsor

UNIT: 12- CLASS & OBJECT

Contents

Is Visual Basic a real object-oriented programming (OOP) language? Is it just an object-based language? Or is it somewhere between these two extremes?

For what it's worth, my position on the question is a compromise: Visual Basic definitively is not a true OOP language and it won't be one until it possesses some essential OOP features, such as inheritance. But this deficit shouldn't excuse you're not learning in depth what classes and objects have to offer developers.

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.

- 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. In this sense, the inclusion of a tool such as Visual Modeler in the Enterprise Edition is a clear sign of Microsoft's will to pursue this goal.
- 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, and you can't write components for COM, DCOM, or MTS. In short, you can do little or nothing without a firm grasp on what objects are and how you can take advantage of them.
- If you're absolutely new to object-oriented programming, this could be the most difficult chapter of the entire book for you to grasp. To understand how objects can help you write better programs in less time, you must be ready for a conceptual leap, not unlike the leap that many programmers had to take when switching from pure procedural MS-DOS languages such as QuickBasic to newer and more sophisticated event-driven programming environments such as Visual Basic. But once you grasp the basic concepts of OOP, you'll probably agree that objects are the most

exciting thing to happen to Visual Basic since its first version. When you dive into object-oriented programming, you'll soon find yourself devising new, concise, and elegant solutions to old problems, often in less time and with less code. But I don't want to sound intimidating. As a Visual Basic programmer, you've already learned to master many advanced programming techniques concerned with, for example, events, database programming, and user interface issues. OOP isn't more difficult, it's merely different. And it's certainly a lot of fun.

- If you've ever read books or articles about OOP, you surely found dozens of different definitions for the term object. Most of the definitions are correct and confusing at the same time. The definition I like most is this one:
- An object is an entity that embeds both data and the code that deals with it.
- Let's see what this means in practice.

The Basic Concepts

I have noticed that many programmers exposed for the first time to OOP tend to confuse classes and objects, so a very short explanation is in order. 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. In a sense, classes are design time–only entities, while objects are run time–only entities.

Your users will never see a class; rather, they'll probably see and interact with objects created from your classes, such as invoices, customer data, or circles on the screen. As a programmer, your point of view is reversed because the most concrete thing you'll have in front of you while you're writing the application is the class, in the form of a class module in the Visual Basic environment. Until you run the application, an object isn't more real than a variable declared with a Dim statement in a code listing. In my opinion, this dichotomy has prevented many Visual Basic programmers from embracing the OOP paradigm. We have been spoiled by the RAD (Rapid Application Development) orientation of our favorite tool and often think of objects as visible objects, such as forms, controls, and so on. While Visual Basic can also create such visible objects—including Microsoft ActiveX controls—you won't grasp the real power of object orientation until you realize that almost everything in your program can be an object, from concrete and visible entities such as invoices, products, customers, employees, and so on to more abstract ones such as the validation process or the relationship between two tables.

The Main Benefits of OOP

Before getting practical, I'd like to hint at what object-oriented programming has to offer you. I'll do that by listing the key features of

OOPLs (object-oriented programming languages) and explaining some concepts.

Encapsulation

Encapsulation is probably the feature that programmers appreciate most in object-oriented programming. In a nutshell, an object is the sole owner of its own data. All data is stored inside a memory area that can't be directly accessed by another portion of the application, and all assignment and retrieval operations are performed through methods and properties provided by the object itself. This simple concept has at least two far-reaching consequences:

- You can check all the values assigned to object properties before they're actually stored in memory and immediately reject all invalid ones.
- You're free to change the internal implementation of the data stored in an object without changing the way the rest of the program interacts with the object. This means that you can later modify and improve the internal workings of a class without changing a single line of code elsewhere in the application.
- As with most OOP features, it's your responsibility to ensure that the class is well encapsulated. The fact that you're using a class doesn't guarantee that the goals of encapsulation are met. In this and the next chapter, I'll show you how some simple rules—and common sense—help you implement robust classes. A robust class is one that actively protects its internal data from tampering. If an object derived from a class holds valid data and all the operations you perform on that object transform the data only into other valid data (or raise an error if the operation isn't valid), you can be absolutely sure that the object will always be in a valid state and will never propagate a wrong value to the rest of the program. This is a simple but incredibly powerful concept that lets you considerably streamline the process of debugging your code.
- The second goal that every programmer should pursue is code reusability, which you achieve by creating classes that are easily maintained and reused in other applications. This is a key factor in reducing the development time and cost. Classes offer much in this respect, but again they require your cooperation. When you start writing a new class, you should always ask yourself: Is there any chance that this class can be useful in other applications? How can I make this class as independent as possible from the particular software I'm developing right now? In most cases, this means adding a few additional properties or additional arguments to methods, but the effort often pays off nicely. Don't forget that you can always resort to default values for properties and optional arguments for methods, so in most cases these enhancements won't really make the code that uses the class more complex than it actually needs to be.

- The concept of self-containment is also strictly related to code reuse and encapsulation. If you want to create a class module that's easily reusable, you absolutely must not allow that class to depend on any entity outside it, such as a global variable. This would break encapsulation (because code elsewhere in the application might change the value of the variable to some invalid data) and above all, it would prevent you from reusing the class elsewhere without also copying the global variable (and its parent BAS module). For the same reason, you should try to make the class independent of general-purpose routines located in another module. In most cases, I prefer to duplicate shorter routines in each class module, if this makes the class easily movable elsewhere.

Polymorphism

- Informally, Polymorphism is the ability of different classes to expose similar (or identical) interfaces to the outside. The most evident kind of polymorphism in Visual Basic is forms and controls. TextBox and PictureBox controls are completely different objects, but they have some properties and methods in common, such as Left,BackColor, and Move. This similarity simplifies your job as a programmer because you don't have to remember hundreds of different names and syntax formats. More important, it lets you manage a group of controls using a single variable (typed as Control, Variant, or Object) and create generic procedures that act on all the controls on a form and therefore noticeably reduce the amount of code you have to write.

Inheritance

- Inheritance is the ability, offered by many OOP languages, to derive a new class (the derived or inherited class) from another class (the base class). The derived class automatically inherits the properties and methods of the base class. For example, you could define a generic Shape class with properties such as Color and Position and then use it as a base for more specific classes (for example, Rectangle, Circle, and so on) that inherit all those generic properties. You could then add specific members, such as Width and Height for the Rectangle class andRadius for the Circle class. It's interesting to note that, while polymorphism tends to reduce the amount of code necessary to use the class, inheritance reduces the code inside the class itself and therefore simplifies the job of the class creator. Unfortunately, Visual Basic doesn't support inheritance, at least not in its more mature form of implementation inheritance. In the next chapter, I show how you can simulate inheritance by manually writing code and explain when and why this can be useful.

Your First Class Module

- Creating a class 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. In this first example, I show how to encapsulate personal data related to a person, so I'm naming this first class CPerson.

Note I admit it: I'm not a fanatic about naming conventions. Microsoft suggests that you use the cls prefix for class module names, but I don't comply simply because I feel it makes my code less readable. I often prefer to use the shorter C prefix for classes (and I for interfaces), and sometimes I use no prefix at all, especially when objects are grouped in hierarchies. Of course, this is a matter of personal preference, and I don't insist that my system is more rational than any other.

- The first version of our class includes only a few properties. These properties are exposed as Public members of the class module itself, as you can see in this code and also in Figure 6-1 on the following page:

-
  ```
  ‘ In the declaration section of the CPerson class module
  Public FirstName As String
  Public LastName As String
  ```
- This is a very simple class, but it's a good starting point for experimenting with some interesting concepts, without being distracted by details. Once you have created a class module, you can declare an object variable that refers to an instance of that class:

-
  ```
  ‘ In a form module
  Private Sub cmdCreatePerson_Click()
  Dim pers As CPerson ‘ Declare.
  Set pers = New CPerson ‘ Create.
  pers.FirstName = "John" ‘ Assign properties.
  pers.LastName = "Smith"
  Print pers.FirstName & " " & pers.LastName ‘Check that it works.
  End Sub
  ```
- The code's not very impressive, admittedly. But remember that here we're just laying down concepts whose real power will be apparent only when we apply them to more complex objects in real-world applications.

Auto-instancing object variables

- Unlike regular variables, which can be used as soon they have been declared, an object variable must be explicitly assigned an object reference before you can invoke the object's properties and methods. In fact, when an object variable hasn't been assigned yet, it contains the special Nothingvalue: In other words, it doesn't contain any valid reference to an actual object. To see what this means, just try out this code:

```
Dim pers As CPerson ‘ Declare the variable,
‘ Set pers = New CPerson ‘ but comment out the creation step.
Print pers.FirstName ‘ This raises an error 91 – "Object variable
‘ or With block variable not set"
```

- In most cases, this behavior is desirable because it doesn’t make much sense to print a property of an object that doesn’t exist. A way to avoid the error is to test the current contents of an object variable using the Is Nothing test:

```
‘ Use the variable only if it contains a valid object reference
If Not (pers Is Nothing) Then Print pers.FirstName
```

In other cases, however, you just want to create an object, any object, and then assign its properties. In these circumstances, you might find it useful to declare an auto-instancing object variable using the As Newclause:

```
Dim pers As New CPerson ‘ Auto-instancing variable
```

When at run time Visual Basic encounters a reference to an auto-instancing variable, it first determines whether it’s pointing to an existing object and creates a brand new instance of the class if necessary. Auto-instancing variables have virtues and liabilities, plus a few quirks you should be aware of:

- Auto-instancing variables obviously reduce the amount of code you need to write to be up and running with your classes. For this reason, they’re often valuable during the prototyping phase.
- Auto-instancing variables can’t be tested against the Nothing value. In fact, as soon as you use one in the Is Nothing test, Visual Basic relentlessly creates a new instance and the test always returns False. In some cases, this could be the decisive factor in whether to stay clear of auto-instancing variables.
- Auto-instancing variables tend to eliminate errors, but sometimes this is precisely what you don’t need. In fact, during the development phase you want to see errors because they’re the symptoms of other serious flaws in the code logic. Auto-instancing variables make the debugging step a little more obscure because you can never be sure when and why an object was created. This is probably the most persuasive reason not to use auto-instancing variables.
- You can’t declare an auto-instancing variable of a generic type, such as Object, Form, or MDI Form because Visual Basic must know in advance which kind of object should be created when it references that variable for the first time.
- In some complex routines, you might declare a variable but never actually use it: this happens all the time with standard variables and with object variables too, but it creates a problem with regular (non-auto-instancing) object variables. In fact, if you create the object with aSet command at the beginning of a procedure, you

might be creating an object—thus taking both time and memory—for no real purpose. On the other hand, if you delay the creation of an object until you actually need it, you could soon find yourself drowning in a sea of Set commands, each preceded by an Is Nothing test to avoid re-creating an object instanced previously. By comparison, auto-instancing variables are automatically created by Visual Basic only if and when they are referenced: In all other cases, no time or memory will be wasted without reason. This is probably the situation in which auto-instancing variables are most useful.

- Finally, each time Visual Basic references an auto-instancing variable, it incurs a small performance hit because it has to check whether it's Nothing. This overhead is usually negligible, but in some time-critical routines it could affect the overall time.
- In summary, auto-instancing variables often aren't the best choice, and in general I advise you not to use them. Most of the code shown in this chapter doesn't make use of auto-instancing variables, and you can often do without them in your own applications as well.

Property procedures

- Let's go back to the CPerson class and see how the class can protect itself from invalid assignments, such as an empty string for its FirstName or LastNameproperties. To achieve this goal, you have to change the internal implementation of the class module because in its present form you have no means of trapping the assignment operation. What you have to do is transform those values into Private members and encapsulate them in pairs of Property procedures. This example shows the code for Property Get and Let FirstName procedures, and the code for LastName is similar.

-
  ```
‘ Private member variables
Private m_FirstName As String
Private m_LastName As String
‘ Note that all Property procedures are Public by default.
Property Get FirstName() As String
‘ Simply return the current value of the member variable.
FirstName = m_FirstName
End Property
Property Let FirstName(ByVal newValue As String)
‘ Raise an error if an invalid assignment is attempted.
If newValue = "" Then Err.Raise 5 ‘ Invalid procedure argument
‘Else store in the Private member variable.
m_FirstName = newValue
End Property
```

Review & Self Assessment Question

Q1-Explain Class and Objects?

Q2-What are the main benefits of OOP's?
Q3-What do you mean by polymorphism?

Further Readings

Visual basic by David I Schneider
Visual basic by Steven Holzner
Visual basic by Bill Sheldon
Visual basic by Billy Holls
Visual basic by Rob Windsor

UNIT: 13 – COMPONENT OBJECT MODEL

Contents

Component Object Model

The Component Object Model (COM) is a Microsoft framework for managing component objects, allowing them to communicate by calling methods of other components in their object workspace. Each component in the framework is a self-contained program. Traditional programs cannot communicate with programs outside their workspace when running, without using other services such as DDE, TCP/IP, etc.

ActiveX is the name given to the services based around the COM framework. The components may be visual (embedded into other documents using Object Linking and Embedding), or a set of services (compiled as a Dynamic Link Library (DLL)).

DISTRIBUTED COM:

The Distributed Component Object Model (DCOM) extends the COM framework so that components may communicate across different computers. It achieves this by executing a Remote Procedure Call (RPC) to a server, which executes the method and returns the results to the client. Creating DCOM components allows you to extend the functionality of ASP. As the components are compiled, they tend to run much faster than calling script routines through the ASP interpreter.

REGISTERING COMPONENTS:

When you develop a COM object, the component is automatically registered on the computer when it's compiled. If you are developing a component for another computer, then you have to copy the compiled DLL to the server and register it explicitly using regsvr32.

BUILDING A COM OBJECT WIITH VISUAL BASIC:

The discussion has thought to create a **COM object using Visual Basic**. The object will be used to determine if a string is a palindrome (a string that reads the same forwards as it does backwards, ignoring case and punctuation).

A DLL project in Visual Basic is implemented as a class module. The class module allows you to define attributes (data of the class), and methods (the functionality of the class). One of the principle concepts of object-oriented programming is that the data of the class is encapsulated in the object. This means that variables of the class are not accessible directly without using some method. Methods of the class tend to be declared as Public as they must be accessible from outside the class. If the class has functions that you don't want to be accessed outside of the class, then you declare them as Private. The Palindrome component we will create has no attributes (variables), one public method to check if a string is a palindrome, and a private function to format the string to determine if it's a palindrome.

Starting The Project:

Start an ActiveX DLL project in Visual Basic. Set the name of the project to CheckPalindrome, and the name the Class Module Palindrome. The following is the code for the Class module

```
Palindrome.cls
Option Explicit
Public Function isPalindrome(ByVal strPal As String) As
Boolean
    Dim strTemp As String
    strPal = stripPunctuation(strPal)
    strTemp = StrReverse(strPal)
    isPalindrome = (strTemp = strPal)
End Function
Private Function stripPunctuation(ByVal strOriginal As String)
As String
    Dim strStripped As String
    Dim iAscii As Integer, counter As Integer
    strOriginal = LCase(strOriginal)
    For counter = 1 To Len(strOriginal)
        iAscii = Asc(Mid(strOriginal, counter, 1))
        ' The lowercase ASCII range is 97 to 122
        If iAscii > 96 And iAscii < 123 Then
            strStripped = strStripped & Chr(iAscii)
        End If
    Next counter
    stripPunctuation = strStripped
End Function
```

Create the DLL by selecting "Make CheckPalindrome.dll" from the File menu. The component will be automatically registered on the machine.

Testing The Component:

You can test the new object by creating a new Standard Exe project in Visual Basic. In the new project, select the "Project" menu item, then select "References". Scroll down to our new component"CheckPalindrome", and place a tick against it. You can now use the object directly in the project.

The following example has one TextBox called txtPal, and one CommandButton called cmdTest.

```
usepal.frm
Option Explicit
Private Sub cmdTest_Click()
    Dim objPal As New Palindrome
    If objPal.isPalindrome(txtPal.Text) = True Then
        MsgBox txtPal.Text & " is a palindrome"
    Else
        MsgBox txtPal.Text & " is not a palindrome"
    End If End Sub
```

Review & Self Assessment Question

Q1-What is Component Object Model?

Q2-What id Distributed COM Explain it?

Further Readings

Visual basic by David I Schneider

Visual basic by Steven Holzner

Visual basic by Bill Sheldon

Visual basic by Billy Holls

Visual basic by Rob Windsor

UNIT: 14 - ACTIVEX CONTROL IN VISUAL BASIC

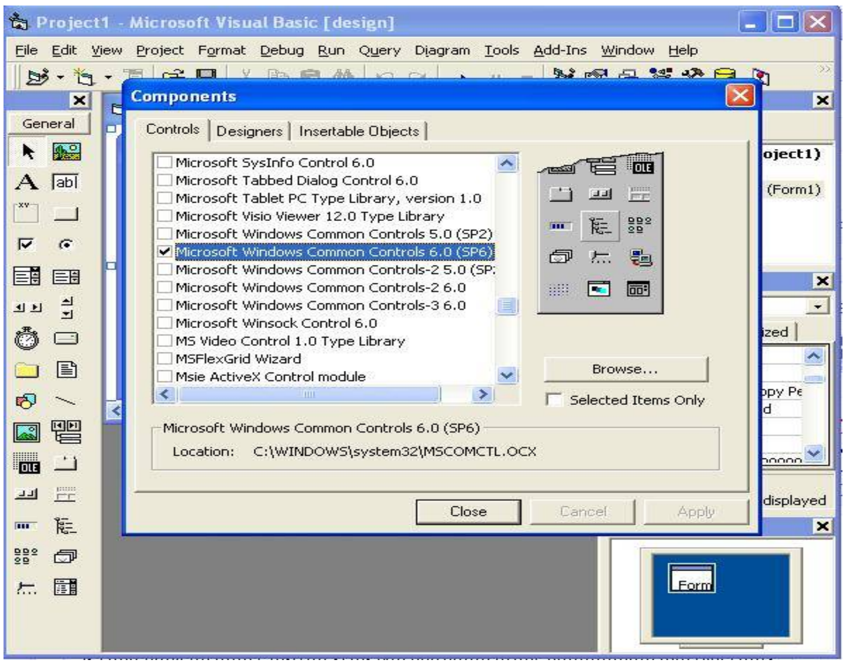
Contents

ActiveX Controls:

One of the most exciting feature of Visual Basic is the ActiveX.It is the technology developed by Microsoft that defines a communication standard between applications. The ActiveX control exists as separate files with a.ocx file name extension. These include controls that are available in all editions of Visual Basic and those that are available only in the Professional and Enterprise editions (such as Listview, Toolbar, Animation, and Tabbed Dialog).

Adding Built-in ActiveX Controls to the Toolbox:

1. From the **Project** Menu, choose **Components**. You can also right click on toolbox. A popup menu is open select **components.**
2. A component dialog box is open .Select the Microsoft windows common controls 6.0(SP6).
3. Then click on apply button .you can see some other components like Listview,
Toolbar, Animation, and Tabbed Dialog show on your toolbox. Click on close button. Use this control as you want


Project1 - Microsoft Visual Basic [design]
File Edit View Project Format Debug Run Query Diagram Tools Add-Ins Window Help
General
Components
Controls | Designers | Insertable Objects
Browse...
Close
Cancel
Apply

- [ ] Microsoft SysInfo Control 6.0
- [ ] Microsoft Tabbed Dialog Control 6.0
- [ ] Microsoft Tablet PC Type Library, version 1.0
- [ ] Microsoft Visio Viewer 12.0 Type Library
- [ ] Microsoft Windows Common Controls 5.0 (SP2)
- [x] Microsoft Windows Common Controls 6.0 (SP6)
- [ ] Microsoft Windows Common Controls-2 5.0 (SP:
- [ ] Microsoft Windows Common Controls-2 6.0
- [ ] Microsoft Windows Common Controls-3 6.0
- [ ] Microsoft Winsock Control 6.0
- [ ] MS Video Control 1.0 Type Library
- [ ] MSFlexGrid Wizard
- [ ] Msie ActiveX Control module

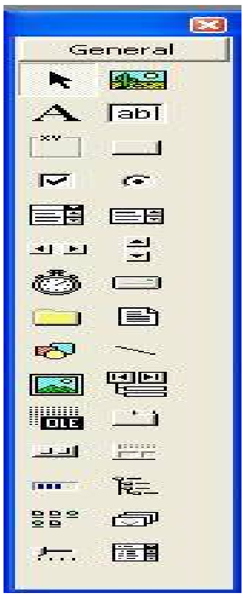
- [ ] Selected Items Only

Microsoft Windows Common Controls 6.0 (SP6)
Location: C:\WINDOWS\system32\MSCOMCTL.OCX

To build an ActiveX control from scratch and compile it so that it is usable as a command button or a list box is in other projects; you write the ActiveX Control and then compile it to an OCX file. The ocx file contains only the user controls that were part of the VB project you built it from and no extra stuff like forms or full programs.

List of Built-in ActiveX Contorls:

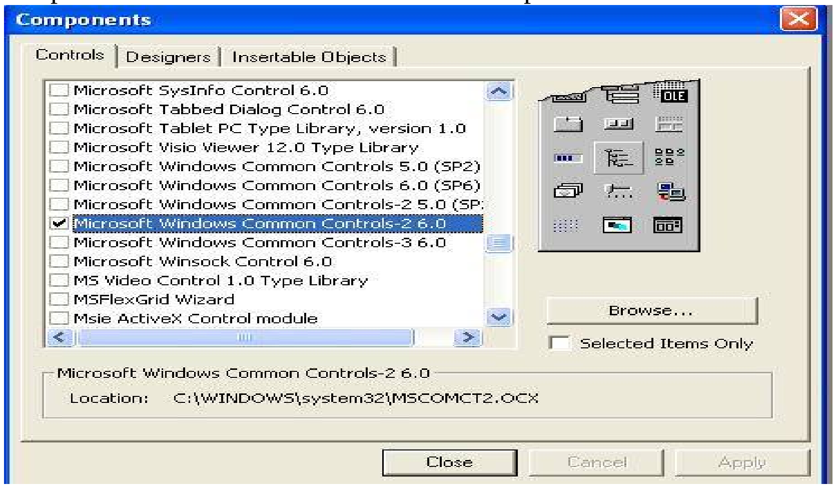
Visual basic 6.0 offers us many built -in activeX Controls of which a few are listed and briefly explained below.



MonthView

A MonthView control display the current month and lets the user scroll through of the month as well .The day, month and year can be displayed using the month view control.

1. Click on **"Start"** . Then select "All Programs, Microsoft Visual Studio 6.0 and Microsoft **Visual Basic 6.0." .**
2. Select **"Standard EXE"** from the list in the New Project dialog box and click on **"Open."**Click on "Project" on the menu bar, then selects**" Components"** from the drop-down menu.
3. Scroll down the list in the box until **Microsoft Windows Common Controls2 6.0 (SP4)** is visible. Click on the checkbox to select the component and then click on **"OK."** All components show in the Toolbox.


Components

Controls | Designers | Insertable Objects

- [ ] Microsoft SysInfo Control 6.0
- [ ] Microsoft Tabbed Dialog Control 6.0
- [ ] Microsoft Tablet PC Type Library, version 1.0
- [ ] Microsoft Visio Viewer 12.0 Type Library
- [ ] Microsoft Windows Common Controls 5.0 (SP2)
- [ ] Microsoft Windows Common Controls 6.0 (SP6)
- [ ] Microsoft Windows Common Controls-2 5.0 (SP
- [x] Microsoft Windows Common Controls-2 6.0
- [ ] Microsoft Windows Common Controls-3 6.0
- [ ] Microsoft Winsock Control 6.0
- [ ] MS Video Control 1.0 Type Library
- [ ] MSFlexGrid Wizard
- [ ] Msie ActiveX Control module

Browse...

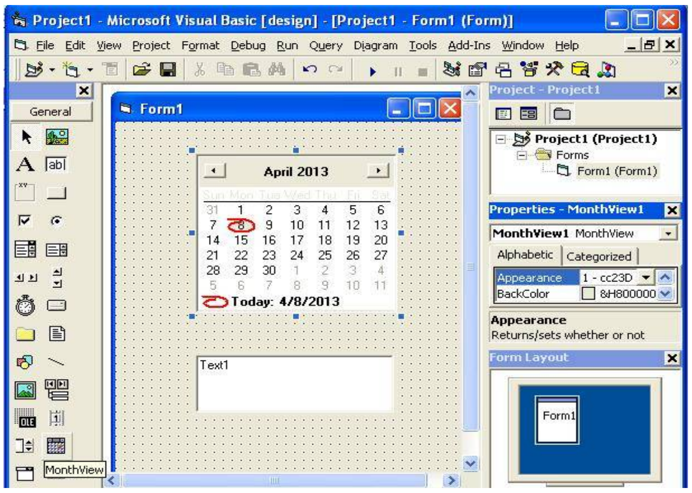
- [ ] Selected Items Only

Microsoft Windows Common Controls-2 6.0

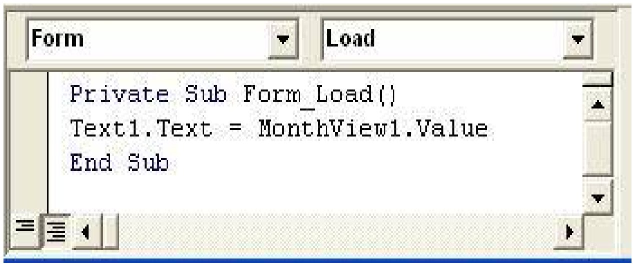
Location: C:\WINDOWS\system32\MSCOMCT2.OCX

Close | Cancel | Apply

4. Choose the**"MonthView1"** control from the list of controls in the toolbox. Place the control on the form in Visual Basic. Also place Textbox1 to the Form1.

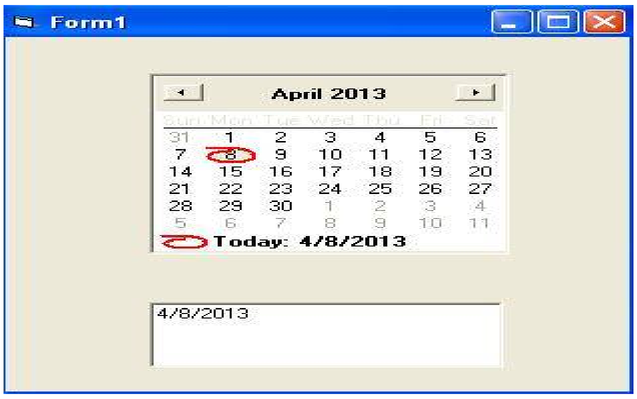


5. Double click on Form1.Code window is open, write code here as shown in figure below:



```
Private Sub Form_Load()
Text1.Text = MonthView1.Value
End Sub
```

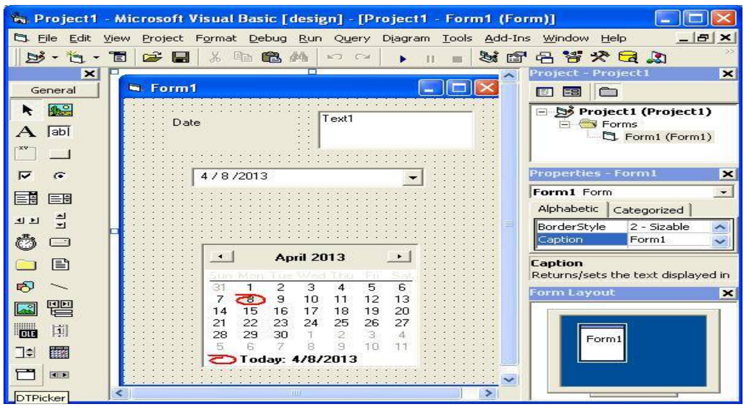
6.Press<F5> to run the program. Output is:



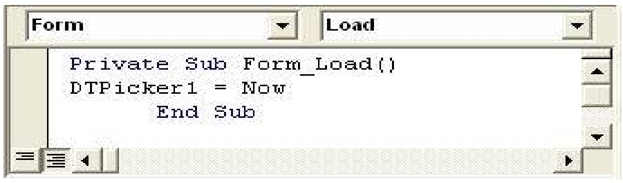
DateTimePicker

DateTimePicker is a ActiveX controls. The **Date/Time Picker** control allow the user to specify a date and time. Date/Time Picker control can display a Month View as a drop down control or the current time with an up down control to let the user select the require time.A MonthView control or Date/time Picker control can be added to a program by using the following steps The **DateTimePicker** control enables you to provide a formatted date field . Users can select a date from a dropdown calendar interface similar to the **MonthView** control.

1. Click on **“Start”** Button. select “All Programs, Microsoft Visual Studio 6.0 and **Microsoft Visual Basic 6.0.” .**
2. Select **“Standard EXE”** from the list in the New Project dialog box and click on**“Open.”** Click on “Project” on the menu bar, then select **“Components”** from the drop-down menu.
3. Scroll down the list in the box until **Microsoft Windows Common Controls2 6.0 (SP4)** is visible. Click on the checkbox to select the component and then click on**“OK.”** All components show in the Toolbox.
4. Choose the **“DT Picker1”** control from the list of controls in the toolbox. Place the control on the form in Visual Basic.
5. Also place one Label1,Textbox and a monthview control on the Form1.
6.Set the caption property “Date” of Label1 .

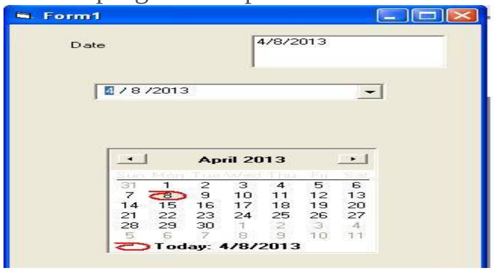


7. Double click on form1.Code window is open .write code here as shown in figure below:



```
Private Sub Form_Load()
DTPicker1 = Now
     End Sub
```

8.Press<F5> to run the program.Output is below:



Progress bar

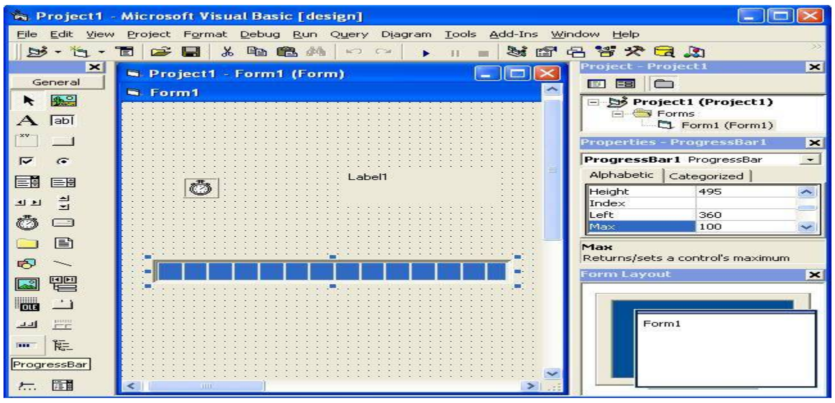
A progress bar control allows you to graphically represent the progress of a transaction. It is used to inform the user about processing. It shows the user the status of computations/processing. Number of applications, such as setups, database-driven applications, and file transfer tools, swear by progress bars.

1. Click on **"Start"** .Select "All Programs -> Microsoft Visual Studio 6.0 ->**Microsoft Visual Basic 6.0." .**

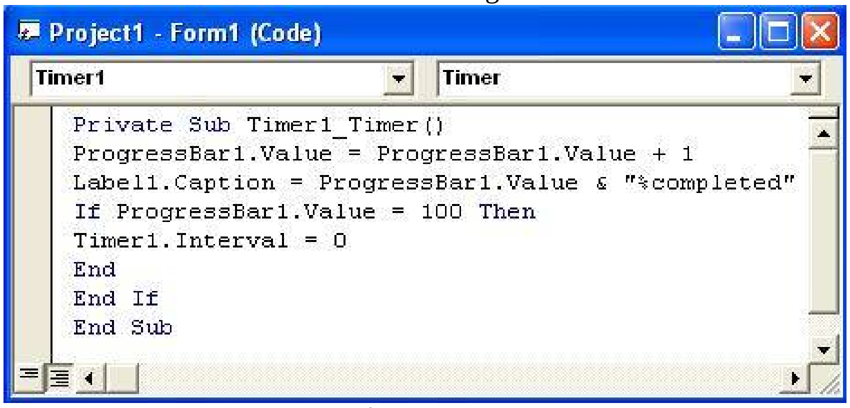
2. Select **"Standard EXE"** from the list in the New Project dialog box and click on **"Open."**Click on "Project" on the menu bar, then select **"Components"** from the drop-down menu.

3. Scroll down the list in the box until **Microsoft Windows Common Controls 6.0 (SP4)** is visible. Click on the checkbox to select the component and then click on**"OK."** All components show in the Toolbox.

4. Choose the **"Progress Bar"** control from the list of controls in the toolbox. Place the control on the form in Visual Basic.Also add one Label1 and Timer1 control on the Form1.

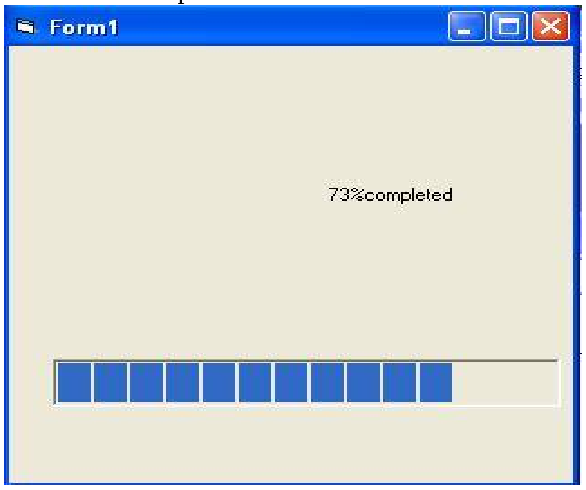


5. Change the Timer1 property "Interval=100".Double click on timer1 control and write code here as shown in figure below:



```
Private Sub Timer1_Timer()
ProgressBar1.Value = ProgressBar1.Value + 1
Label1.Caption = ProgressBar1.Value & "%completed"
If ProgressBar1.Value = 100 Then
Timer1.Interval = 0
End
End If
End Sub
```

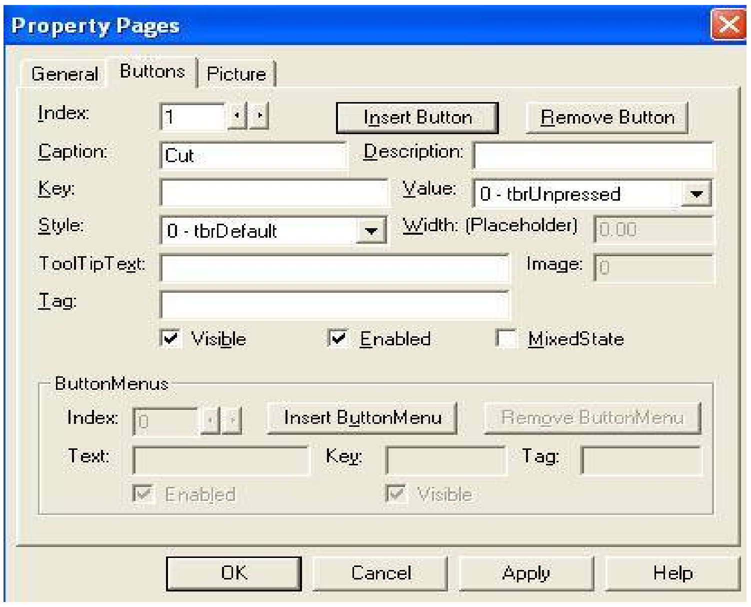
6. To Run the program press <f5>. You see when run the program time is start Progressbar value is shown in Label1.when the time is complete .The application is terminate.Output is below:



ToolBar

A **ToolBar** control consist of a collection of button objects used to create a toolbar that can be associated with an application. It has become one of the most important tools for providing an easy interface to the users. the toolbar control provides easy access to options available in your applications. A toolbar control contains a collection of buttons objects used to create a Toolbar.

1. Click on **"Start"**. Then select "All Programs, Microsoft Visual Studio 6.0 and **Microsoft Visual Basic 6.0." .**
2. Select **"Standard EXE"** from the list in the New Project dialog box and click on **"Open."** Click on "Project" on the menu bar, then select **"Components"** from the drop-down menu.
3. Scroll down the list in the box until **Microsoft Windows Common Controls 6.0 (SP4)** is visible. Click on the checkbox to select the component and then click on **"OK."** All components show in the Toolbox.
4. Choose the **"ToolBar"** control from the list of controls in the toolbox. Place the control on the form in Visual Basic.
5. Right click on toolbar a popup menu is open choose properties from popup menu. Set the properties as desired for the control. You can also add some buttons on the toolbar. For this you click on the Buttons tab and then click on the **Insert Button**. you add buttons as you wish.


Property Pages

General | Buttons | Picture

Index: 1 [Insert Button] [Remove Button]

Caption: Cut Description:

Key: Value: 0 - tbrUnpressed

Style: 0 - tbrDefault Width: (Placeholder) 0.00

ToolTipText: Image: 0

Tag:

☑ Visible ☑ Enabled ☐ MixedState

ButtonMenus

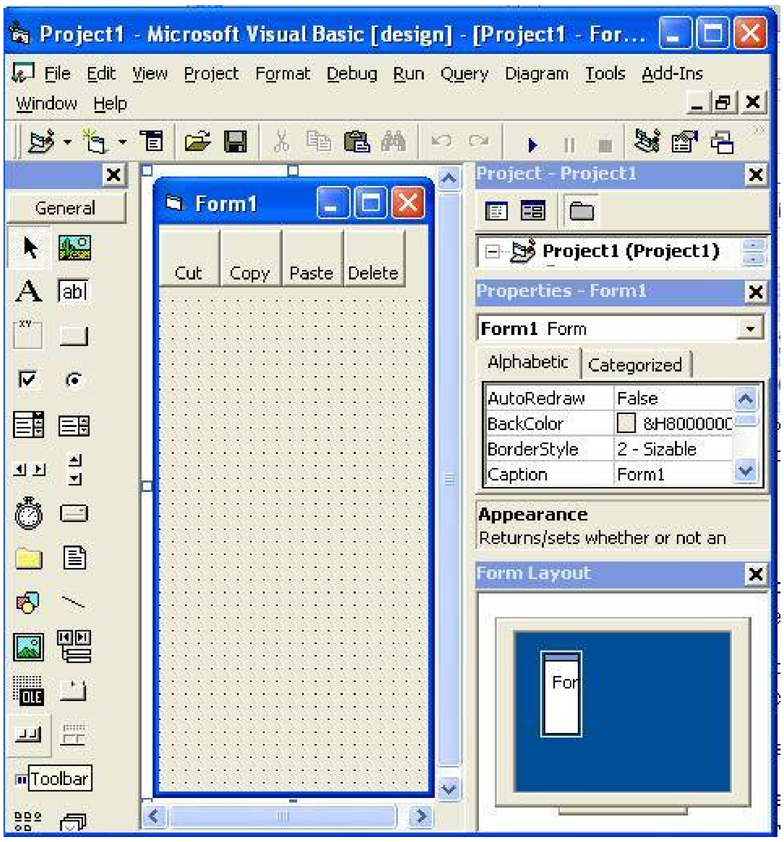
Index: 0 [Insert ButtonMenu] [Remove ButtonMenu]

Text: Key: Tag:

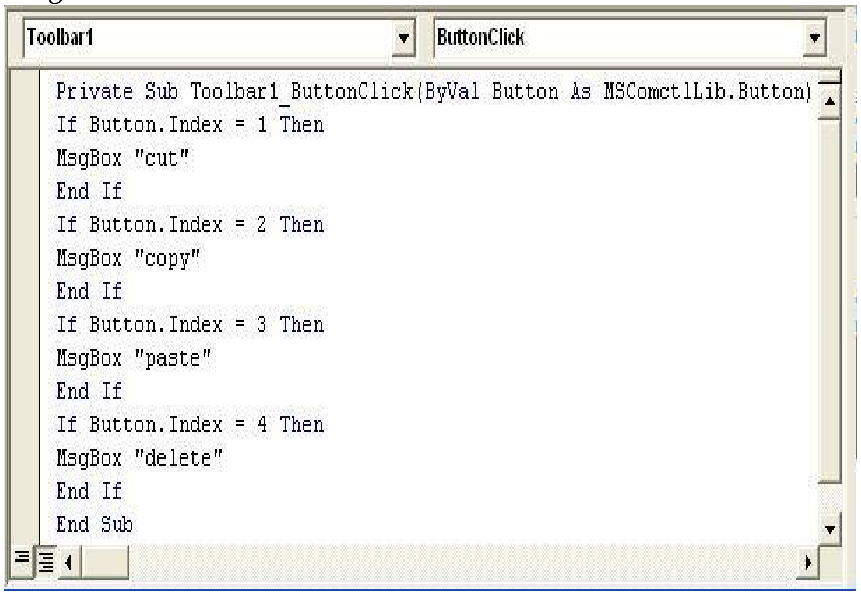
☑ Enabled ☑ Visible

[OK] [Cancel] [Apply] [Help]

6.We add four Buttons on the toolbar.It is shown in Figure below:

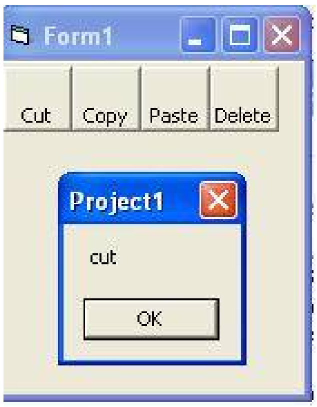


7. Double Click on toolbar.Code window is open .add code here as shown in figure:



```
Toolbar1                                ButtonClick
Private Sub Toolbar1_ButtonClick(ByVal Button As MSComctlLib.Button)
If Button.Index = 1 Then
MsgBox "cut"
End If
If Button.Index = 2 Then
MsgBox "copy"
End If
If Button.Index = 3 Then
MsgBox "paste"
End If
If Button.Index = 4 Then
MsgBox "delete"
End If
End Sub
```

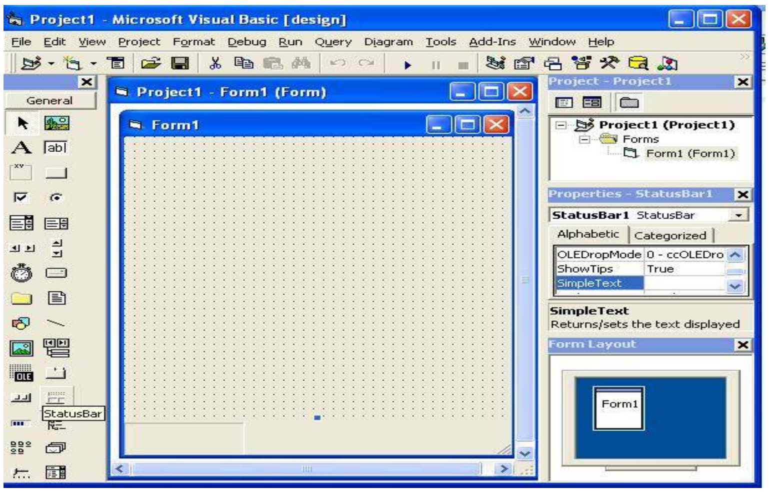
8. When you are done with these steps, press<F5> to run the program. Output is given below:



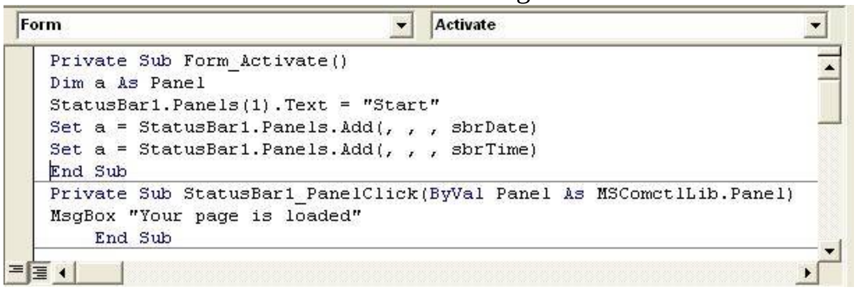
StatusBar

The **StatusBar** control is another ActiveX control.It is also available in Windows Common Controls components.With the help of StatusBar, you can easily display information about the date, time, and other details about the application and the environment to the user. A Statusbar control is a frame that can consisit of sveral panels,which informs the user of the status of an application. It is made up of Panel objects, each of which displays different types of information.

1. Click on **"Start"** . Then select "All Programs, Microsoft Visual Studio 6.0 and **"Microsoft Visual Basic 6.0." .**
2. Select **"Standard EXE"** from the list in the New Project dialog box and click on "Open." Click on "Project" on the menu bar, then select **"Components"** from the drop-down menu.
3. Scroll down the list in the box until **Microsoft Windows Common Controls 6.0 (SP4)** is visible. Click on the checkbox to select the component and then click on "OK." All components show in the Toolbox.
4. Choose the **"StatusBar"** control from the list of controls in the toolbox. Place the control on the form in Visual Basic.

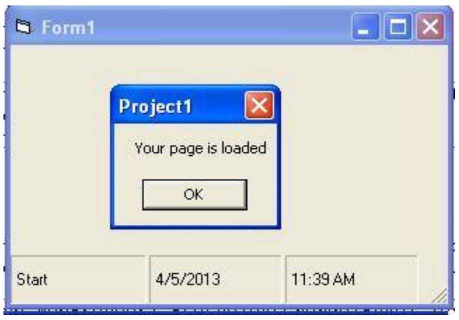


5. By default, one panel is show on the StatusBar.Double click on StatusBar and write code here as shown in figure below:



```
Private Sub Form_Activate()
Dim a As Panel
StatusBar1.Panels(1).Text = "Start"
Set a = StatusBar1.Panels.Add(, , , sbrDate)
Set a = StatusBar1.Panels.Add(, , , sbrTime)
End Sub
Private Sub StatusBar1_PanelClick(ByVal Panel As MSComctlLib.Panel)
MsgBox "Your page is loaded"
    End Sub
```

7. To Run the program press <f5>. When you click on StatusBar panel then a msgbox is shown. Output is :



ImageList

An **ImageList** Control contains a collection of images that can be used by other Windows Common Control ,Such as ListView,TreeView,TabStrip and ToolBar controls.It does not appeare on the format run time.It serve as a container for icon that are accessed by other control such as ListView,TreeView,TabStrip and ToolBar controls.

A ListView control displays data as ListItem objects.each ListItem object can have an optional icon associated with the Label of the object.

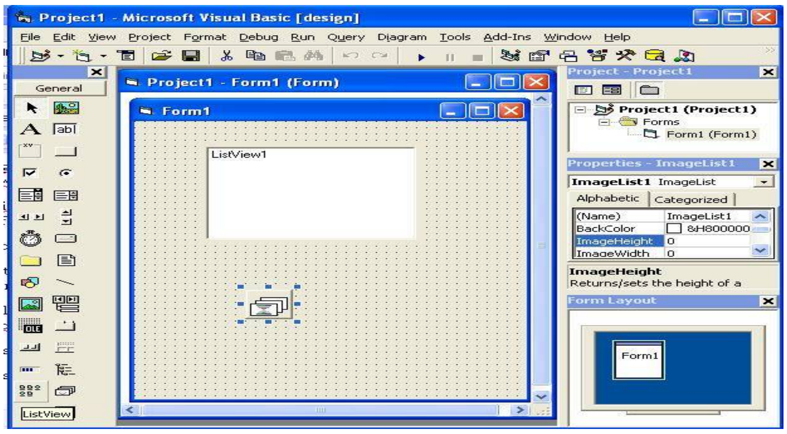
1. Click on **"Start"**. Then select "All Programs, Microsoft Visual Studio 6.0 and **Microsoft Visual Basic 6.0.".**

2. Select **"Standard EXE"** from the list in the New Project dialog box and click on**"Open."** Click on "Project" on the menu bar, then select **"Components"** from the drop-down menu.

3. Scroll down the list in the box until **Microsoft Windows Common Controls 6.0 (SP4)** is visible. Click on the checkbox to select the component and then click on**"OK."** All components show in the Toolbox.

4. Choose the **"ListView1"** control from the list of controls in the toolbox. Place the control on the form in Visual Basic.

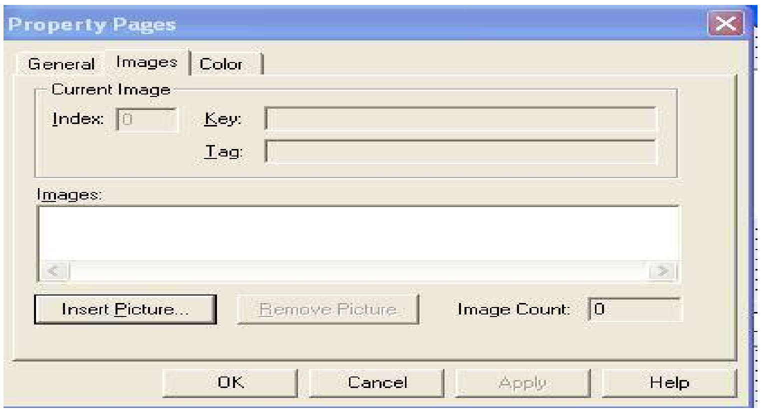
5. Choose the **"ImageList1"** control from the list of controls in the toolbox. Place the control on the form in Visual Basic.



(Name)	ImageList1
BackColor	&H800000
ImageHeight	0
ImageWidth	0

ImageHeight
Returns/sets the height of a

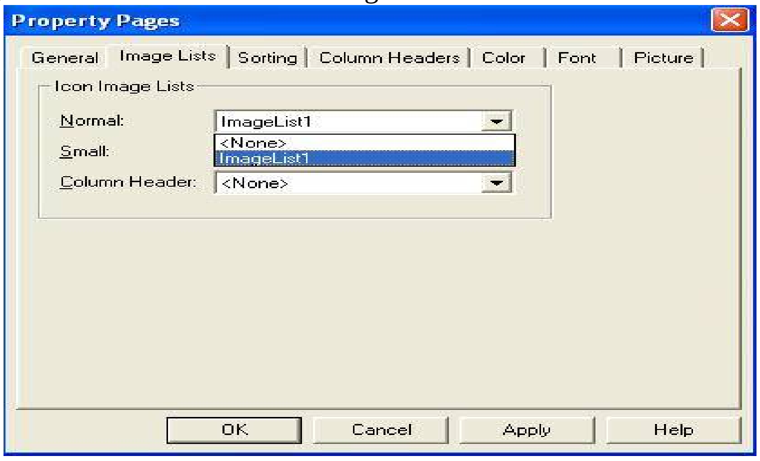
6. Right click on Image control a popup menu is open choose properties option. A Properties Page is open .Click on Images Tab and insert the image as you want. If you want insert more than one picture then click on Insert Picture Button as shown in figure below:



Index: 0

Image Count: 0

7. Right click on the ListView1 A popup menu is open choose properties option. A Properties page is open. Click on the imagelist tab and insert the imagelist1 then “OK” as shown in figure below:

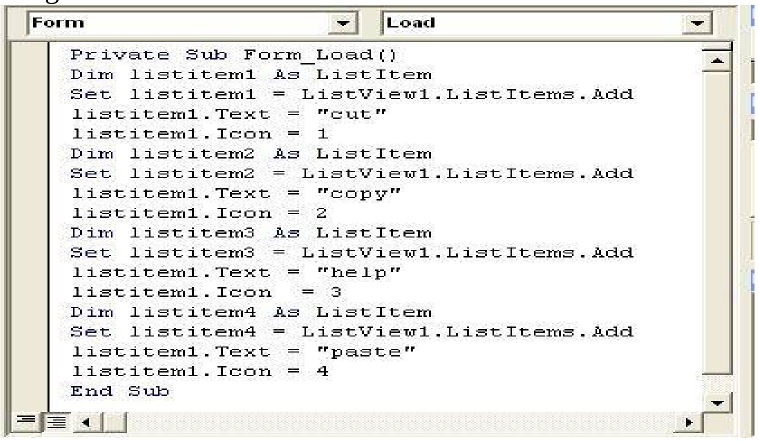


Normal: ImageList1

Small:

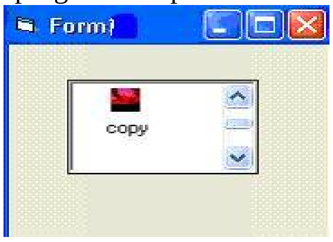
Column Header: <None>

8. Double click on Form1.Code window is open. Write code here as shown in figure:



```
Private Sub Form_Load()
Dim listitem1 As ListItem
Set listitem1 = ListView1.ListItems.Add
listitem1.Text = "cut"
listitem1.Icon = 1
Dim listitem2 As ListItem
Set listitem2 = ListView1.ListItems.Add
listitem1.Text = "copy"
listitem1.Icon = 2
Dim listitem3 As ListItem
Set listitem3 = ListView1.ListItems.Add
listitem1.Text = "help"
listitem1.Icon  = 3
Dim listitem4 As ListItem
Set listitem4 = ListView1.ListItems.Add
listitem1.Text = "paste"
listitem1.Icon = 4
End Sub
```

9. Press <F5> to run the program. Output is shown below:



Review and Self Assessment Question

Q1-What is Active X Control?

Q2-Write the steps to Add Active X Control to toolbox?

Q3-What is Date Time Picker? Explain with example.

Q4- What is Progress bar? Explain with example.

Q5- Describe Toolbar?

Further Readings

Visual basic by David I Schneider

Visual basic by Steven Holzner

Visual basic by Bill Sheldon

Visual basic by Billy Holls

Visual basic by Rob Windsor

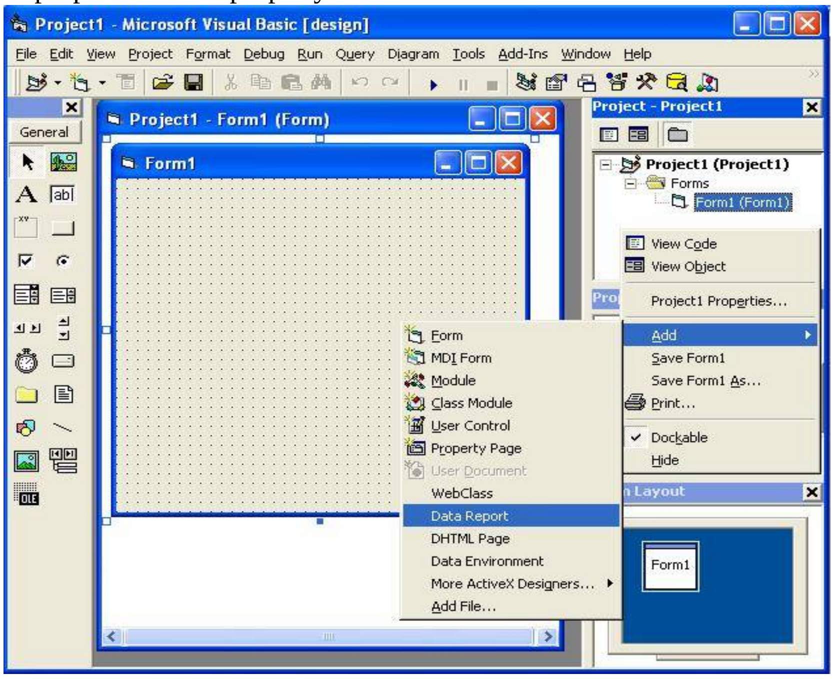
UNIT: 15 - DATA ENVIRONMENT & DATA REPORTS IN VISULA BASIC

- **Contents**
- Data Environment
- Steps to Add Data Environment
- Steps to Generate Data Report
- Review & Self Assessment Question
- Further Readings

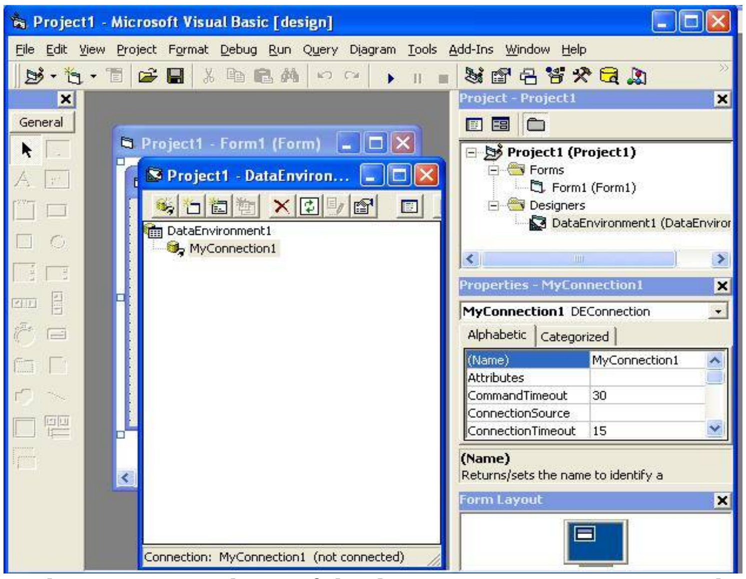
Data Environment

Visual basic allows you to add a database environment just the way you add a form. A data environment designer is an ActiveX designer tool provided by Visual basic. A data environment designer provides a interactive design time environment to access data from a database at run time. A data environment enable to perform following task:

1. Right-click at the project in the project explorer whence a pop-up menu appears.
2. Click at the add option when a list of insert able objects is displayed.
3. Choose data environment when one is added with a default name –data environment1.you can always change the name to your choice by setting its properties in the property window. :



4. Data environment opens a connection to the source database. However, you will have to set it up manually. Let us change the name of the connection1 to MyConnection1 simply type the same in the (Name) property as shown below.


Project1 - Microsoft Visual Basic [design]
File Edit View Project Format Debug Run Query Diagram Tools Add-Ins Window Help
General
Project1 - Form1 (Form)
Project1 - DataEnviron...
DataEnvironment1
MyConnection1
Project - Project1
Project1 (Project1)
Forms
Form1 (Form1)
Designers
DataEnvironment1 (DataEnviror
Properties - MyConnection1
Alphabetic
Categorized
Form Layout

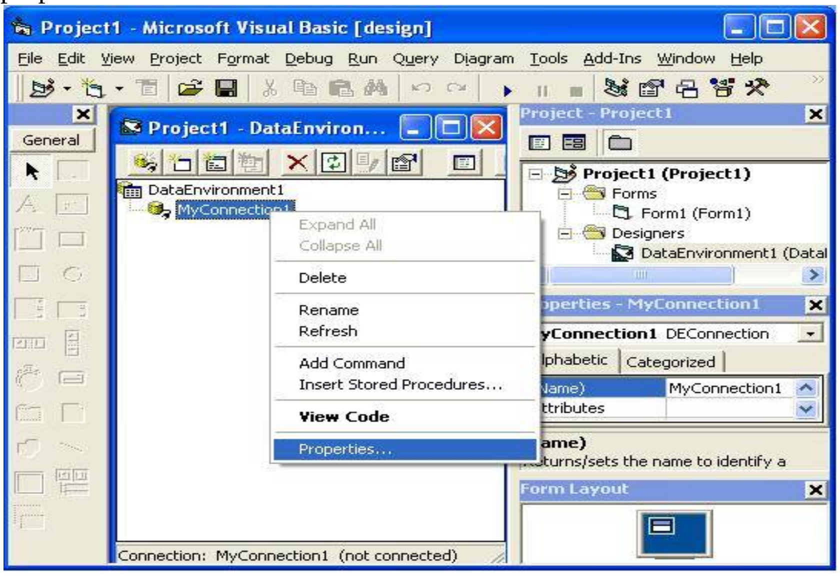
MyConnection1 DEConnection

(Name)	MyConnection1
Attributes	
CommandTimeout	30
ConnectionSource	
ConnectionTimeout	15

(Name)
Returns/sets the name to identify a

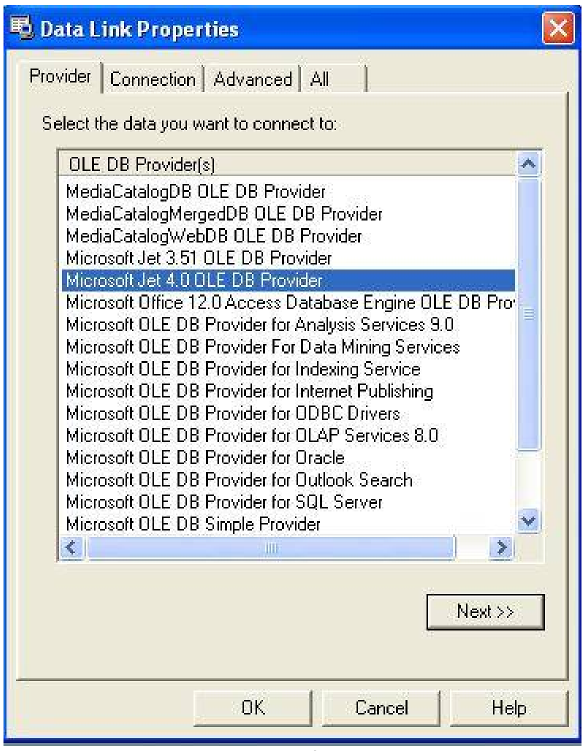
Connection: MyConnection1 (not connected)

5. The connection object of the data environment component has a number of properties that can be set to provide different functionalities to the component.
6. You can connect to a desired database by the following step:
7. Right click at the MyConnection1 or whichever connection you have. The pop menu shown below appears from where you can choose properties.



Connection: MyConnection1 (not connected)

8. When you click on properties… a dialog box shown below. This dialog box helps you set the database link to his connection. It lists all the types of database drivers currently installed on your machine. A typical dialog box appears as shown below:

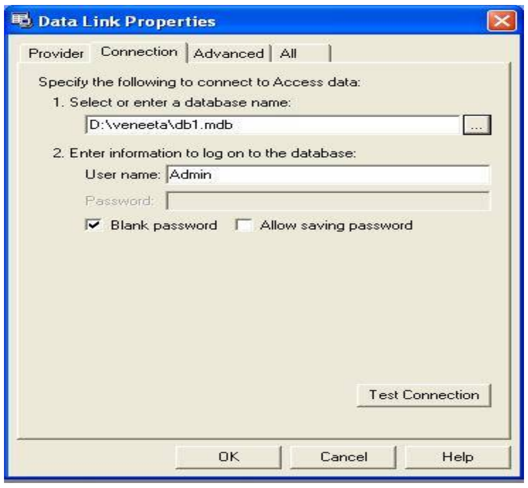

Data Link Properties

Provider | Connection | Advanced | All

Select the data you want to connect to:

OLE DB Provider(s)
MediaCatalogDB OLE DB Provider
MediaCatalogMergedDB OLE DB Provider
MediaCatalogWebDB OLE DB Provider
Microsoft Jet 3.51 OLE DB Provider
Microsoft Jet 4.0 OLE DB Provider
Microsoft Office 12.0 Access Database Engine OLE DB Pro
Microsoft OLE DB Provider for Analysis Services 9.0
Microsoft OLE DB Provider For Data Mining Services
Microsoft OLE DB Provider for Indexing Service
Microsoft OLE DB Provider for Internet Publishing
Microsoft OLE DB Provider for ODBC Drivers
Microsoft OLE DB Provider for OLAP Services 8.0
Microsoft OLE DB Provider for Oracle
Microsoft OLE DB Provider for Outlook Search
Microsoft OLE DB Provider for SQL Server
Microsoft OLE DB Simple Provider

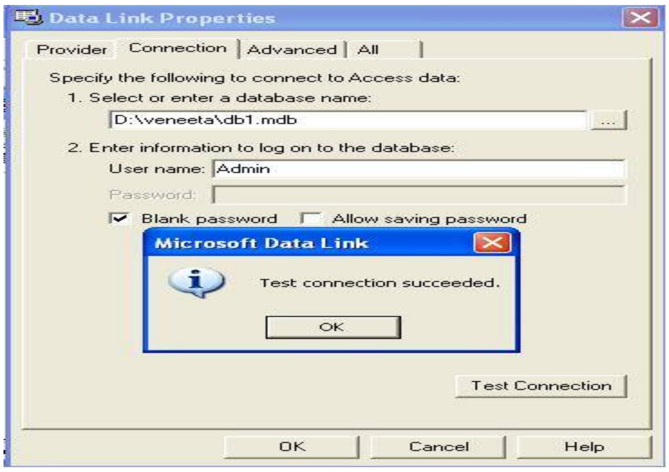
9. Select the Driver Microsoft Jet 4.0 OLE DB Provider.
10. Next, click at the Connection tab of the dialog box. The Connection tab opens up as shown below:


Data Link Properties

Provider | Connection | Advanced | All

Specify the following to connect to Access data:
1. Select or enter a database name:
D:\veneeta\db1.mdb
2. Enter information to log on to the database:
User name: Admin
Password:
☑ Blank password ☐ Allow saving password

11. Select the Microsoft Access database you wish to connect to by clicking at button place on the right side of the textbox in which you enter the name of the database. The database name is shown in the text box shown in figure above.
12. Now, you can test whether your connection was established correctly by clicking at Test Connection button as shown below:



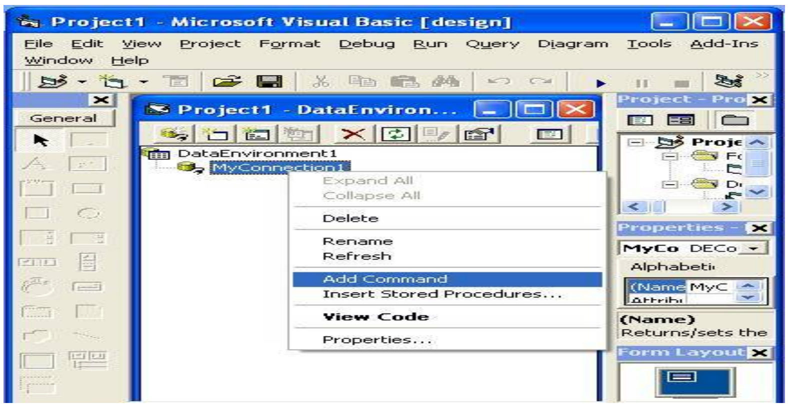
Specify the following to connect to Access data:
1. Select or enter a database name:
D:\veneeta\db1.mdb
2. Enter information to log on to the database:
User name: Admin
Password:
[✓ Blank password] Allow saving password

Test connection succeeded.

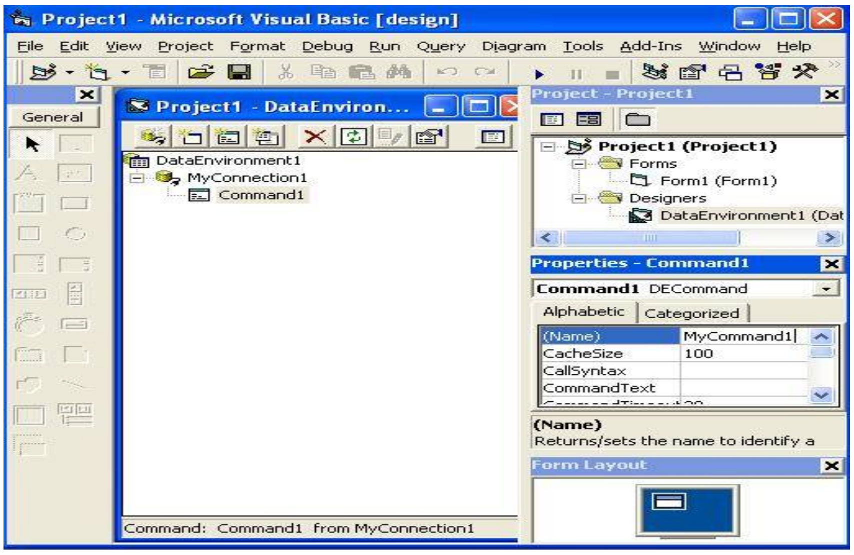
13. If the connection could not be established you will get the message for the same. Now your application is ready to use this connection to interact with the database.
When the connection is create, we can create Data Reports.

How you can create Data Reports:-

A database contains a number of objects like tables, queries. When you have created a connection to the desired database, and you need to select the table in the database from which the data will be fetch in the report. To do this right click at your data environment object and click at Add Command as shown below:



2. A command object with the default name command1 is added to the data environment as shown below:


Project1 - Microsoft Visual Basic [design]
File Edit View Project Format Debug Run Query Diagram Tools Add-Ins Window Help
General
Project1 - DataEnviron...
DataEnvironment1
MyConnection1
Command1
Command: Command1 from MyConnection1
Project - Project1
Project1 (Project1)
Forms
Form1 (Form1)
Designers
DataEnvironment1 (Dat
Properties - Command1
Form Layout

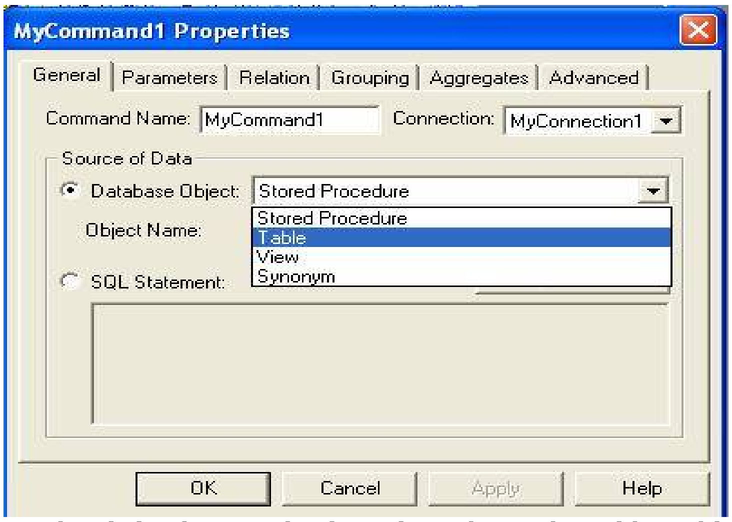
Command1 DECommand

Alphabetic | Categorized

(Name)	MyCommand1
CacheSize	100
CallSyntax	
CommandText	

(Name)
Returns/sets the name to identify a

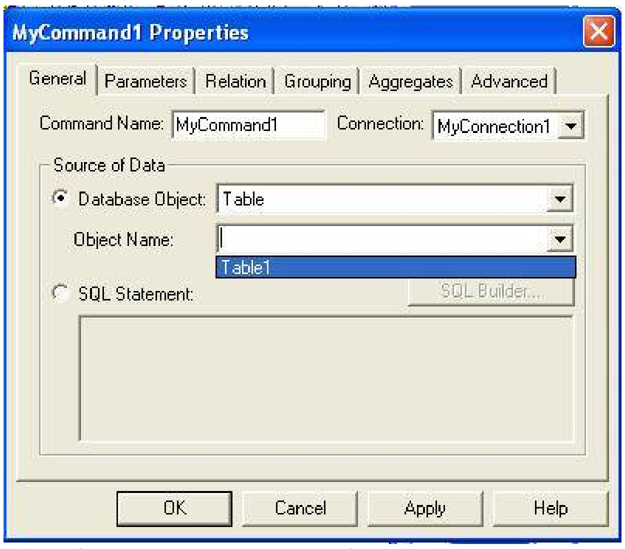
3. Change Command's name to MyCommand1 in the properties window.
4. Set the command to obtain data from a table from the connection database. To do this right click at MyCommand1 and select properties. A dialog box will appear as shown below:



Command Name: MyCommand1
Connection: MyConnection1
Database Object: Stored Procedure
Object Name:
SQL Statement:

5. In this dialog box set the desired Database Object like Table from where you wish to get the data. so, we will take our data from a table-Table1.
6. Alternatively, you can write an SQL statement for this command object by clicking SQL statement option button.

7.


MyCommand1 Properties

General | Parameters | Relation | Grouping | Aggregates | Advanced

Command Name: MyCommand1 Connection: MyConnection1

Source of Data

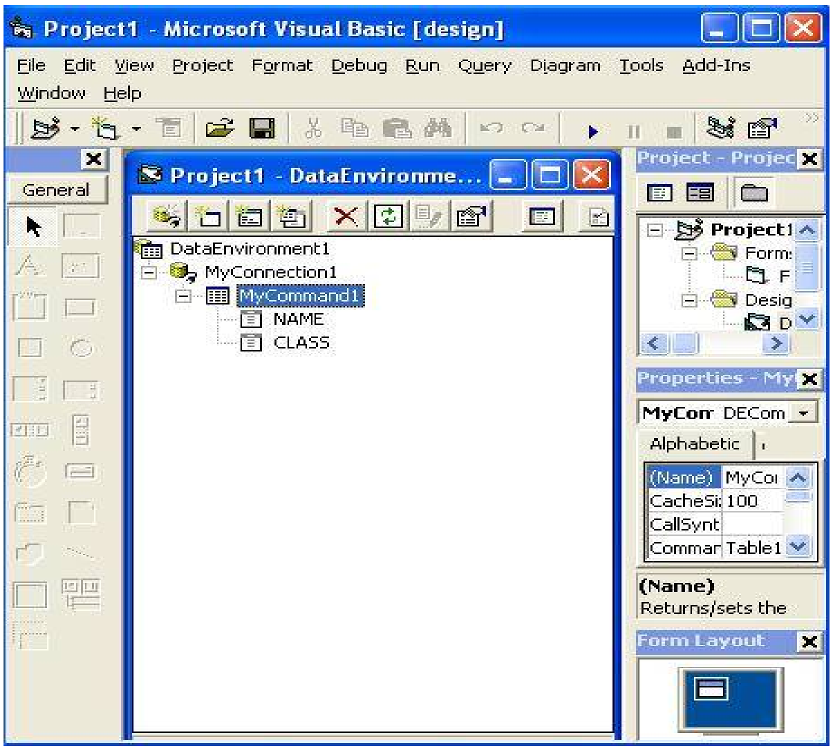
(•) Database Object: Table

Object Name: Table1

( ) SQL Statement: SQL Builder...

OK | Cancel | Apply | Help

7. In case of Table object table fields are shown in MyCommand1 command object as shown in figure below:


Project1 - Microsoft Visual Basic [design]

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

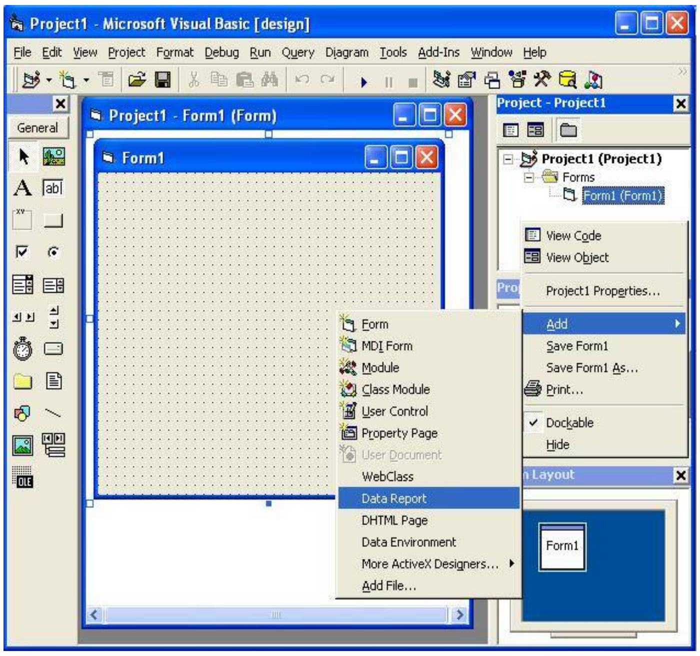
Project1 - DataEnvironme...

DataEnvironment1
MyConnection1
MyCommand1
NAME
CLASS

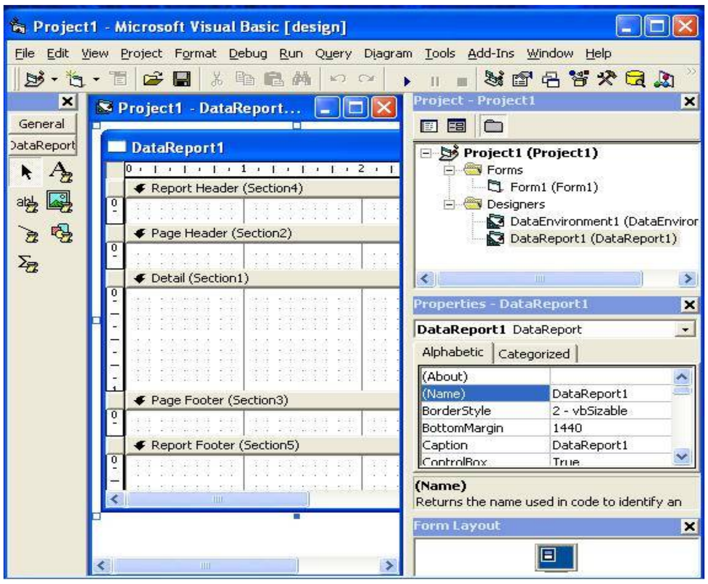
Properties - MyCommand1: MyCom DECom
(Name) MyCo
CacheSi 100
CallSynt
Commar Table1

(Name)
Returns/sets the

8. Now you are ready to design your report. To include a report in your application add one Data Report object in the Project Explorer right clicking at the project as shown below:



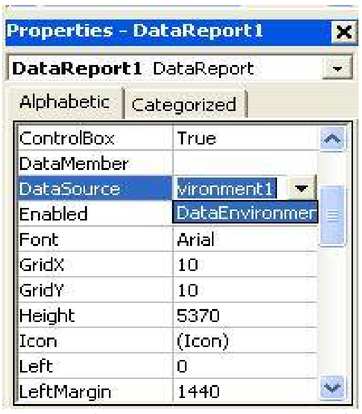
9. A data report form is added to the project and displayed for modification .Change the name of this data report to MyReport1 into the property box as shown below.



(About)	
(Name)	DataReport1
BorderStyle	2 - vbSizable
BottomMargin	1440
Caption	DataReport1
ControlBox	True

(Name)
Returns the name used in code to identify an

10. To bind it to the data source MyDataSource1, click at Data Source property of the report and select the data environment- MyDataSource as shown below.



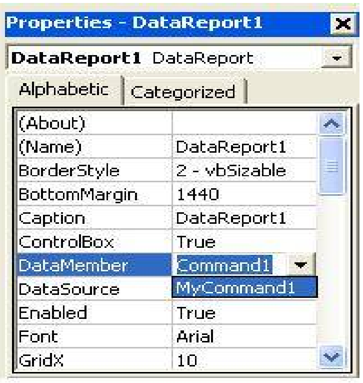
Properties - DataReport1

DataReport1 DataReport

Alphabetic | Categorized

ControlBox	True
DataMember	
DataSource	vironment1
Enabled	DataEnvironmer
Font	Arial
GridX	10
GridY	10
Height	5370
Icon	(Icon)
Left	0
LeftMargin	1440

11. Now, your report knows from where it has to fetch its data. To select the source object, set the DataMember property to MyCommand1 in the property box of the report as shown below:



Properties - DataReport1

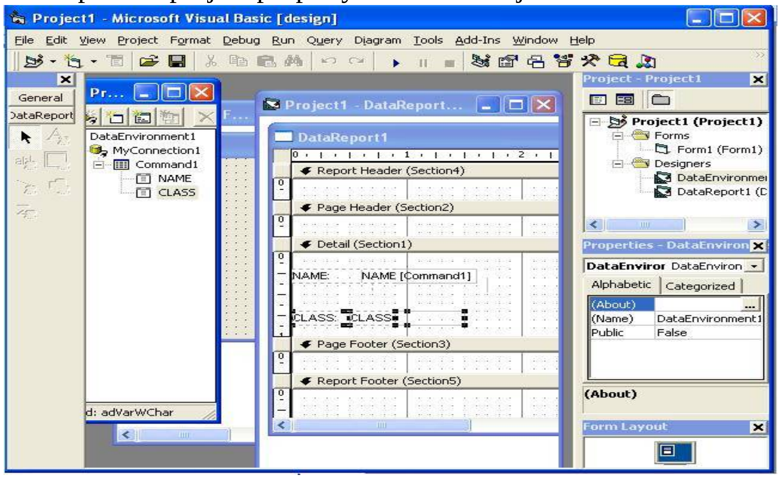
DataReport1 DataReport

Alphabetic | Categorized

(About)	
(Name)	DataReport1
BorderStyle	2 - vbSizable
BottomMargin	1440
Caption	DataReport1
ControlBox	True
DataMember	Command1
DataSource	MyCommand1
Enabled	True
Font	Arial
GridX	10

12. Now, you are ready to create your report, for this there are two way in Visual Basic.

In First way, open the DataEnvironment1 and drag the field from this and drop on the Detail (Section1). Drag and drop the fields as you want. Now click open the project property box from Project menu.



Project1 - Microsoft Visual Basic [design]

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

DataEnvironment1
MyConnection1
Command1
NAME
CLASS

Project1 - DataReport1
DataReport1
Report Header (Section4)
Page Header (Section2)
Detail (Section1)
NAME: NAME [Command1]
CLASS: CLASS
Page Footer (Section3)
Report Footer (Section5)

Project - Project1
Project1 (Project1)
Forms
Form1 (Form1)
Designers
DataEnvironme
DataReport1 (D

Properties - DataEnviron
DataEnviror DataEnviron
Alphabetic | Categorized

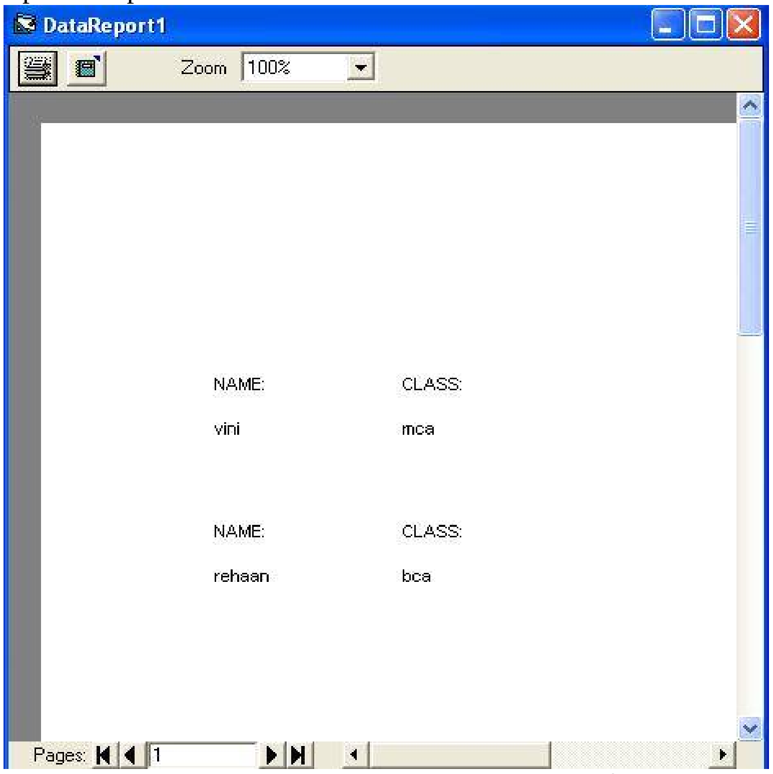
(About)	...
(Name)	DataEnvironment1
Public	False

(About)

Form Layout

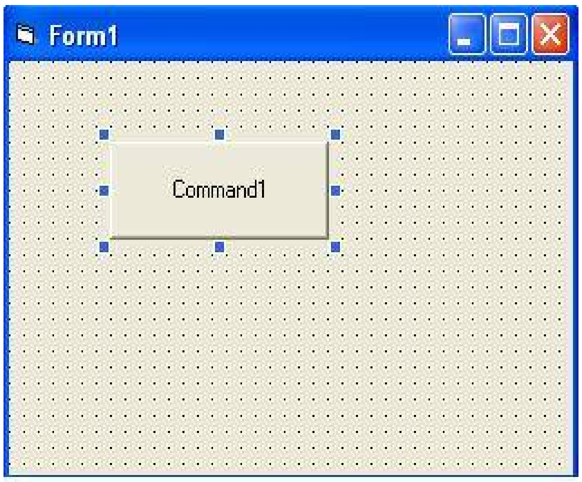
d: adVarWChar

Set the Strup Object to DataReport1.Now run the project to show the report. Output is shown below.

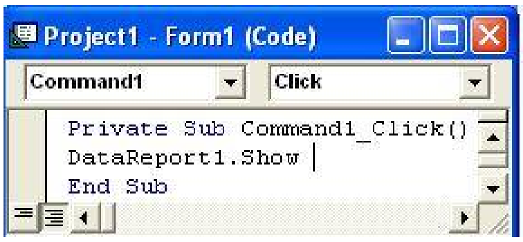


```
NAME:          CLASS:
vini           mca

NAME:          CLASS:
rehaan         bca
```

The second way, by adding a command button to a form and writing some code to open the report at its event click procedure. For this click open the Form1 to add a command button as shown below.

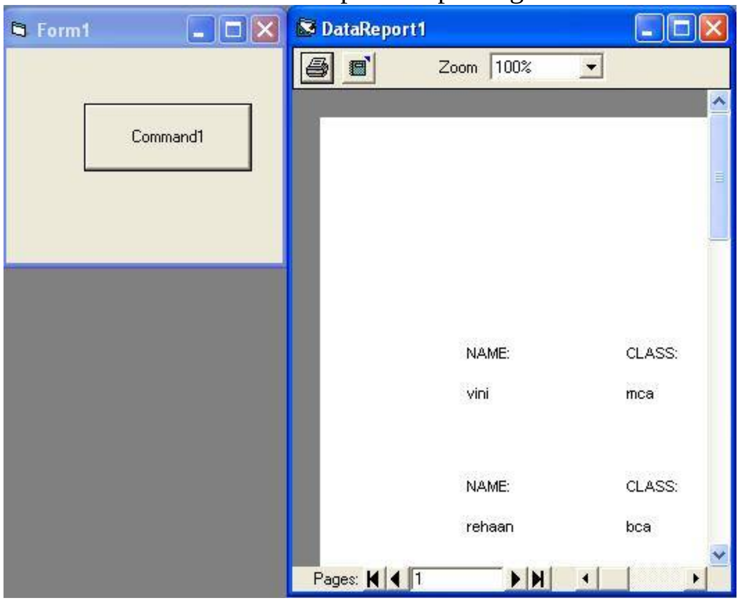


Double click at the command button to open its event procedure. Add the code for opening the report as shown below:



```
Private Sub Command1_Click()
DataReport1.Show
End Sub
```

Click open the project property box from Project menu .Set the Strup Object to Form1.Now run the project From1 should be displayed. Click at the command button to show the report.Output is given below:



```
NAME:      CLASS:
vini       mca

NAME:      CLASS:
rehaan     bca
```

Review & Self Assessment Question

Q1- What is Data Environment?
Q2- Write the steps to add Data Environment?
Q3-Write the steps to generate Data report?

Further Readings

Visual basic by David I Schneider
Visual basic by Steven Holzner
Visual basic by Bill Sheldon
Visual basic by Billy Holls
Visual basic by Rob Windsor

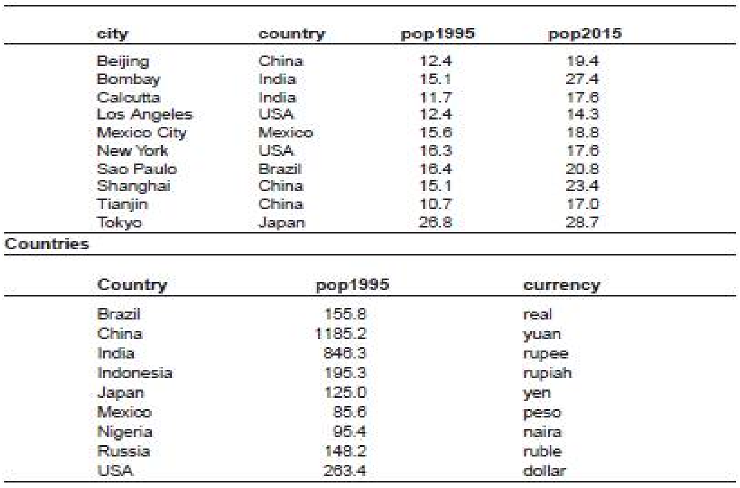
UNIT: 16 - DATABASE MANAGEMENT SYSTEM

Contents

An introduction to Database

A table is a rectangular array of data. Each column of the table, called a field, contains the same type of information. Each row, called a record, contains the same type of information as every other row.

Cities



city	country	pop1995	pop2015
Beijing	China	12.4	19.4
Bombay	India	15.1	27.4
Calcutta	India	11.7	17.6
Los Angeles	USA	12.4	14.3
Mexico City	Mexico	15.6	18.8
New York	USA	16.3	17.6
Sao Paulo	Brazil	16.4	20.8
Shanghai	China	15.1	23.4
Tianjin	China	10.7	17.0
Tokyo	Japan	26.8	28.7

Countries

Country	pop1995	currency
Brazil	155.8	real
China	1185.2	yuan
India	846.3	rupee
Indonesia	195.3	rupiah
Japan	125.0	yen
Mexico	85.6	peso
Nigeria	95.4	naira
Russia	148.2	ruble
USA	263.4	dollar

A database (or relational database) is a collection of one or more (usually related) tables that has been created with database management software. The best known dedicated database management products are Access, Btrieve, dBase, FoxPro, and Paradox. Every version of Visual Basic 6.0 can manage, revise, and analyze a database that has been created with one of these products.

The databases used in this chapter can be found in the collection of files accompanying this text. The database files have the extension .MDB. For instance, the file MEGACTY1. MDB is a database file containing the two tables presented on the preceding page.

THE DATA CONTROL

Visual Basic communicates with databases through the data control. Data controls can read, modify, delete, and add records to databases. The following walkthrough uses a data control to connect Visual Basic to the database MEGACTY1.MDB.

A DATA CONTROLWALKTHROUGH

1. Press Alt/File/New Project and double-click on Standard EXE.
2. Double-click on the data control icon. Set its Name property to datCities and its Caption property to Cities.
3. Stretch it horizontally to see the caption Cities.
4. Select the DatabaseName property and set it to the filespec for the file MEGACTY1.MDB.

An Open File dialog box will help you locate the file.

5. Select the Record Source property and click on the down-arrow button at the right of the Settings window.

The names of the two tables in the database, Cities and Countries, are displayed.

6. Select Cities.
7. Place a text box, txtCity, on the form.

Text boxes are said to be **data-aware** because they can be bound to a data control and access its data.

8. In the Properties window, select the DataSource property of txtCity.
9. Click on the down arrow to the right of the Settings box and select datCities.
10. Select the DataField property and click on the down arrow at the right of the Settings box.

You will see the names of the different fields in the table.

11. Select the field city.

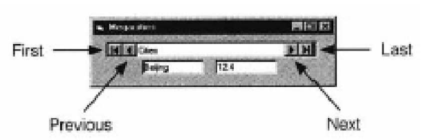
The text box now is said to be **bound** to the data control. It can now display data from the city field of the Cities table.

12. Place another text box, txtPop1995, on the form.
13. Select txtPop1995’s DataSource property.
14. Click on the down arrow to the right of the Settings box and select datCities.
15. Select the DataField property, click on the down arrow at the right of the Settings box, and select pop1995.
16. Run the program.

The form will appear as in Figure 11-1. The arrows on the data control, called **navigation arrows**, look and act like VCR buttons. The arrows have been identified by the tasks they perform.

17. Click on the various navigation arrows on the data control to see the different cities and their populations in the Cities table displayed in the text boxes.
18. Change the name of a city or change its population and then move to another record.

If you look back through the records, you will see that the data have been permanently changed.



USING CODE WITH A DATA CONTROL

Only one record can be accessed at a time; this record is called the **current record**. In this walkthrough,the text boxes bound to the data control showed the contents of the city and pop1995 fields of the current record. The user clicked on the navigation arrows of the data control to select a new current record.

Code can be used to designate another record as the current record. The methods MoveNext, MovePrevious, MoveLast, and MoveFirst select a new current record as suggested by their names. For instance, the statement

```
Data1.Recordset.MoveLast
```

specifies the last record of the table to be the current record. (The word Recordset is inserted in most data-control statements that manipulate records for reasons that needn’t concern us now.)

The entry of the field fieldName of the current record is

```
Data1.Recordset.Fields(“fieldName”).Value
```

For instance, with the status as in from previous, the statement

strVar = datCities.Recordset.Fields(“city”).Value assigns “Beijing” to the variable strVar and the statements

```
datCities.Recordset.Edit
datCities.Recordset.Fields(“city”).Value = “Peking”
datCities.Recordset.Update
```

change the city field of the current record to “Peking”. (The first statement makes a copy of the current record for editing. The second statement alters the copy, and the third statement sends the new copy of the record to the database.)

The number of previously accessed records in the table is given by the RecordCount property. The EOF (End Of File) and BOF (Beginning Of File) run-time properties indicate whether the end or beginning of the file has been reached. For instance, the following two sets of statements each place the cities into a list box.

```
datCities.Recordset.MoveLast ‘Needed to set value of RecordCount
datCities.Recordset.MoveFirst
For i = 1 to datCities.Recordset.RecordCount
lstBox.AddItem datCities.Recordset.Fields(“city”).Value
datCities.Recordset.MoveNext
Next i
datCities.Recordset.MoveFirst
Do
While Not datCities.Recordset.EOF
```

lstBox.AddItem datCities.Recordset.Fields("city").Value
datCities.Recordset.MoveNext
Loop
The current record can be marked for removal with the statement

    Data1.Recordset.Delete

The record will be removed when a data control navigation arrow is clicked or a Move method is executed. A new record can be added to the end of the table with the statement

    Data1.Recordset.AddNew

followed by

    Data1.Recordset.Fields("fieldName").Value = entryForField

statements for each field and a

    Data1.Recordset.Update

statement. Alternately, the AddNew method can be followed by the user typing the information into text boxes bound to the data control and then moving to another record. (**Note:** When you add a record and then click on the MovePrevious arrow, you will not see the next-to-last record, but rather will see the record preceding the record that was current when AddNew was executed.)

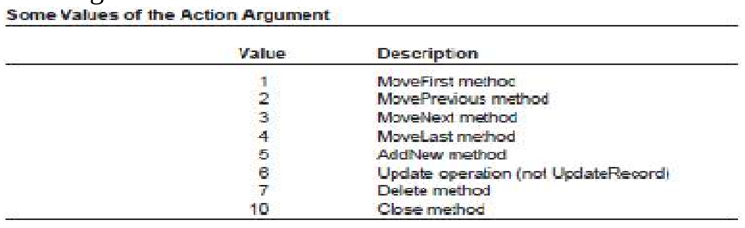
THE VALIDATION EVENT

Visual Basic has a device called **validation** that lets you restrict the values a user can enter into a table. For instance, if the Cities table is only to contain cities with a population of more than 1 million, you can use validation to prevent any record with a number less than 1 in the pop1995 field from being entered. Validation also allows you to catch (and prevent) errors that might cause a program to crash.
Data controls have an event procedure called Validate that is activated whenever the current

1. Record is about to be changed. For instance, it is called when a navigation arrow is clicked or a Move, Update, Delete, or AddNew method is executed. The general form of the Validate event procedure is

Private Sub Data1_Validate (Action As Integer, Save As Integer)
statement(s)
End Sub
The value of Action identifies the specific operation that triggered the event and the value of Save specifies whether data bound to the control has changed. You can change the value of the

1. Action argument to perform a different action. Some values of the Action argument are shown in table


Some Values of the Action Argument

Value	Description
1	MoveFirst method
2	MovePrevious method
3	MoveNext method
4	MoveLast method
5	AddNew method
6	Update operation (not UpdateRecord)
7	Delete method
10	Close method

If you assign 0 to the Action argument, the operation will be canceled when the Validate event procedure is exited.
The value of Save is –1 (True) if the data in any control attached to the data control have changed and is 0 (False) otherwise. If you set the value of Save to 0 in the Validate event procedure, any changes will not be saved.
Consider the form created in the walkthrough. Suppose the contents of txtPop1995, the 1995 population text box, is changed to .8 and then a navigator arrow is clicked in an attempt to move to another record. The following code prevents the move.

```
Private Sub datCities_Validate (Action As Integer, Save As Integer)
Dim strMsg As String
If val(txtPop1995) < 1 then
strMsg = “We only allow cities having a population ” & _ “at least one
million.”
MsgBox strMsg“City too small!”
Action = 0
End If
End Sub
```

If the statement
Action = 0 is changed to
Save = 0
the move will take place, but the previous record will retain its original values. That is, the number .8 will not appear in the table.

Relational Databases and SQL

Primary and Foreign Keys

A well-designed table should have a field (or set of fields) that can be used to uniquely identify each record. Such a field (or set of fields) is called a **primary key**. For instance, in the Countries table , the country field is a primary key. In the Cities table, because

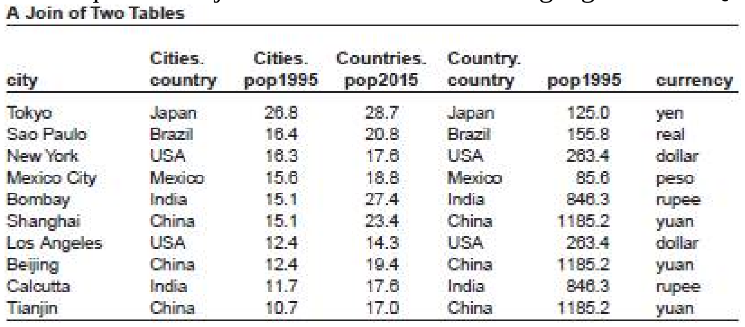
1. we are only considering very large cities (of over 1 million population), the city field is a primary key. Databases of student enrollments in a college usually use a field of social security numbers as the primary key. Names would not be a good choice because there could easily be two students having the same name.

When a database is created, a field can be specified as a primary key. If so, Visual Basic will insist that every record have an entry in the primary key field and that the same entry does not appear in two different records. If the user tries to enter a record with no data in the primary key, the error message “Index or primary key can’t contain a null record.” will be generated. If the user tries to enter a record with the same primary key data as another record, the error message “Duplicate value in index, primary key, or relationship. Changes were unsuccessful.” will be displayed.

When a database contains two or more tables, the tables are usually related. For instance, the two tables Cities and Countries are related by their country field. Let’s refer to these two fields as Cities. Country and

Countries. country. Notice that every entry in Cities. country appears uniquely in Countries. country and Countries. Country is a primary key. We say that Cities. country
is a **foreign key** of Countries.country. Foreign keys can be specified when a table is first created.
If so, Visual Basic will insist on the **Rule of Referential Integrity**, namely, that each
value in the foreign key must also appear in the primary key of the other table.
The CD accompanying this book contains a database named MEGACTY2.MDB. It has the same information as MEGACTY1.MDB except that Cities. city and Countries. country have been specified as primary keys for their respective tables, and Cities. Country has been specified as a foreign key of Countries. country. If the user tries to add a city to the Cities table whose country does not appear in the Countries table, then the error message "Can't add or change record. Referential integrity rules require a related record in table 'Countries'." will be displayed. The message will also be generated if the user tries to delete a country from the Countries. Country field that appears in the Cities. Country field. Due to the interdependence of the two tables in MEGACTY2.MDB, this database is called a **relational database**.

A foreign key allows Visual Basic to link (or **join**) together two tables from a relational database in a meaningful way. For instance, when the two tables Cities and Countries from MEGACTY2.MDB are joined based on the foreign key Cities. country, the result is Table.
The record for each city is expanded to show its country's population and its currency. This joined table is very handy if, say, we wanted to click on navigation arrows and display a city's name and currency. We only have to create the original two tables; Visual Basic creates the joined table as needed. The request for a joined table is made in a language called SQL.



A Join of Two Tables

city	Cities. country	Cities. pop1995	Countries. pop2015	Country. country	pop1995	currency
Tokyo	Japan	26.8	28.7	Japan	125.0	yen
Sao Paulo	Brazil	16.4	20.8	Brazil	155.8	real
New York	USA	16.3	17.6	USA	263.4	dollar
Mexico City	Mexico	15.6	18.8	Mexico	85.6	peso
Bombay	India	15.1	27.4	India	846.3	rupee
Shanghai	China	15.1	23.4	China	1185.2	yuan
Los Angeles	USA	12.4	14.3	USA	263.4	dollar
Beijing	China	12.4	19.4	China	1185.2	yuan
Calcutta	India	11.7	17.6	India	846.3	rupee
Tianjin	China	10.7	17.0	China	1185.2	yuan

SQL

Structured Query Language (SQL) was developed in the early 1970s at IBM for use with relational databases. The language was standardized in 1986 by ANSI (American National Standards Institute). Visual Basic uses a version of SQL that is compliant with ANSI-89 SQL.

SQL is a very powerful language. One use of SQL is to request specialized information from an existing database and/or to have the information presented in a specified order.

FOUR SQL REQUESTS

We will focus on four basic types of requests that can be made with SQL.
Request I: Show the records of a table in a specified order.
Some examples of orders with MEGACTY2.MDB are
(a) Alphabetical order based on the name of the city.
(b) Alphabetical order based on the name of the country, and within each country group, the name of the city.
(c) In descending order based on the projected 2015 population.
Request II: Show just the records that meet certain criteria.
Some examples of criteria with MEGACTY2.MDB are
(a) Cities that are in China.
(b) Cities whose 2015 population is projected to be at least 20 million.
(c) Cities whose name begins with the letter S.
Request III: Join the tables together, connected by a foreign key, and present the records as in Requests I and II.
Some examples with MEGACTY2.MDB are
(a) Show the cities in descending order of the populations of their countries.
(b) Show the cities whose currency has “u” as its second letter.
Request IV: Make available just some of the fields of either the basic tables or the joined table. (For now, this type of request just conserves space and effort by Visual Basic. However, it will be very useful in Section 11.3 when used with a FlexGrid control.)
Some examples with MEGACTY2.MDB are
(a) Make available just the city and country fields of the table Cities.
(b) Make available just the city and currency fields of the joined table.
Normally, we set the RecordSource property of a data control to an entire table. Also, the records of the table are normally presented in the order they are physically stored in the table.
We make the requests discussed above by specifying the RecordSource property as one of the following kinds of settings.
Request I: SELECT * FROM Table1 ORDER BY field1 ASC
or SELECT * FROM Table1 ORDER BY field1 DESC
Request II: SELECT * FROM Table1 WHERE criteria
Request III: SELECT * FROM Table1 INNER JOIN Table2 ON foreign field= primary field WHERE criteria
Request IV: SELECT field1, field2, . . . fieldN FROM Table1 WHERE criteria
The words ASC and DESC specify ASCending and DESCending orders, respectively. A criteria clause is a string containing a condition of the type used with If blocks. In addition to the standard operators <, >, and =, criteria strings frequently contain the operator Like.

Essentially, Like uses the wildcard characters? and * to compare a string to a pattern. A question mark stands for a single character in the same position as the question mark. For instance, the pattern "B?d" is matched by "Bid", "Bud", and "Bad". An asterisk stands for any number of characters in the same position as the asterisk. For instance, the pattern "C*r" is matched by "Computer", "Chair", and "Car". See Comments 3 through 5 for further information about Like.

In the sentence

SELECT fields FROM clause

fields is either * (to indicate all fields) or a sequence of the fields to be available (separated by commas), and clause is either a single table or a join of two tables. A join of two tables is indicated by a clause of the form table1 INNER JOIN table2 ON foreign key of table1=primary key of table2

Appending

WHERE criteria

to the end of the sentence restricts the records to those satisfying criteria. Appending

ORDER BY field(s) ASC (or DESC)

presents the records ordered by the specified field or fields.

In general, the SQL statements we consider will look like

SELECT www FROM xxx WHERE yyy ORDER BY zzz

where SELECT www FROM xxx is always present and accompanied by one or both of WHERE yyy and ORDER BY zzz. In addition, the xxx portion might contain an INNER JOIN phrase.

The settings for the examples mentioned earlier are as follows:

I (a) Show the records from Cities in alphabetical order based on the name of the city.

SELECT * FROM Cities ORDER BY city ASC

I (b) Show the records from Cities in alphabetical order based first on the name of the country and,, within each country group,, the name of the city.

SELECT * FROM Cities ORDER BY country,, city ASC

I (c) Show the records from Cities in descending order based on the projected 2015 population.

SELECT * FROM Cities ORDER BY pop2015 DESC

II (a) Show the records for the Cities in China.

SELECT * FROM Cities WHERE country = 'China'

II (b) Show the records from Cities whose 2015 population is projected to be at least 20 million.

SELECT * FROM Cities WHERE pop2015 >= 20

II (c) Show the records from Cities whose name begins with the letter S.

SELECT * FROM Cities WHERE city Like 'S*'

III (a) Show the records from the joined table in descending order of the populations of their countries.

SELECT * FROM Cities INNER JOIN Countries ON
Cities.country =

1. Countries.country ORDER BY Countries.pop1995 DESC

III (b) Show the records from the joined table whose currency has “u” as its second letter.

SELECT * FROM Cities INNER JOIN Countries ON Cities.country =
Countries.country WHERE currency Like ‘?u*’

IV (a) Make available just the city and country fields of the table Cities.

SELECT city, country FROM Cities

IV (b) Make available just the city and currency fields of the joined table.

SELECT city,, currency FROM Cities INNER JOIN Countries ON Cities.country
= Countries.country

Note: In several of the statements, the single quote, rather than the normal double quote was
used to surround strings. This is standard practice with SQL statements.
We can think of an SQL statement as creating in essence a new “virtual” table from existing tables. For instance, we might regard the statement

SELECT city, pop2015 FROM Cities WHERE pop2015 >= 20

as creating the “virtual” table

city	pop2015
Tokyo	28.7
Sao Paulo	20.8
Bombay	27.4
Shanghai	23.4

This table is a subtable of the original table Cities, that is, it consists of what is left after certain columns and rows are deleted.
As another example, the statement

SELECT Cities.city, Cities.Country, Country.currency FROM Cities INNER JOIN
Countries ON Cities.country = Countries.country WHERE Countries.country>’K’

creates in essence the “virtual” table

Cities.city	Cities.country	currency
New York	USA	dollar
Mexico City	Mexico	peso
Los Angeles	USA	dollar

which is a subtable of a join of the two tables Cities and Countries.
These “virtual” tables don’t really exist physically. However, for all practical purposes, Visual Basic acts as if they did. In Visual Basic terminology, a “virtual” table is called a **recordset** and SQL statements are said to create a recordset. In standard relational database books, a “virtual” table is called a **view**.

SQL also can be used in code with a statement of the form

Data1.RecordSource = “ SELECT ... FROM ...”

to alter the order and kinds of records presented from a database. However, such a statement must be followed by the statement

Data1.Refresh

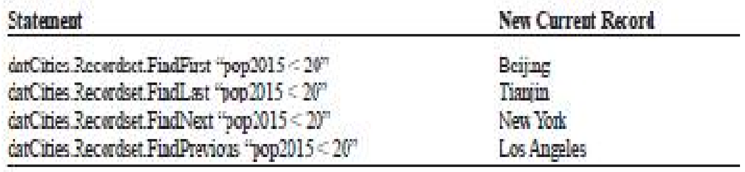
to reset the information processed by the data control.

FIND METHODS

Suppose a table has been attached to the data control Data1, and an SQL statement has been
used to create and order a recordset. Then a statement of the form

Data1.RecordSet.FindFirst criteria

starts at the beginning of the recordset, searches for the first record in the recordset that satisfies the criteria, and makes that record the current record. (Here, criteria is a string just like a criteria phrase that follows WHERE in an SQL statement.) The related methods Find- Last, FindNext, and FindPrevious function as their names suggest. (FindLast starts at the end of the recordset. FindNext and FindPrevious start at the current record.) For instance, suppose an SQL statement ordered the cities alphabetically by name. The following statements and their outcomes show the effects of the various Find methods. (In each case, assume that the current record is Mexico City.)



Statement	New Current Record
datCities.Recordset.FindFirst "pop2015 < 20"	Beijing
datCities.Recordset.FindLast "pop2015 < 20"	Tianjin
datCities.Recordset.FindNext "pop2015 < 20"	New York
datCities.Recordset.FindPrevious "pop2015 < 20"	Los Angeles

Visual Basic has two properties, NoMatch and Bookmark that help when a Find method fails to locate a suitable record.
If BkMk is a string variable, a statement of the form

BkMk = Data1.Recordset.Bookmark

assigns the location of the current record to the variable BkMk. When desired, the statement

Data1.Recordset.Bookmark = BkMk

will return to the original location in the table.
If a Find method does not locate a record matching the criteria, the first or last record (depending on where the search is heading) in the recordset becomes the current record and the NoMatch property is set to True. Therefore, the following lines of code can be used to keep the current record current whenever the Find method is unsuccessful.

```
BkMk = Data1.Recordset.Bookmark
Data1.Recordset.FindNext criteria
If Data1.Recordset.NoMatch = True Then
Data1.Recordset.Bookmark = BkMk
End If
```

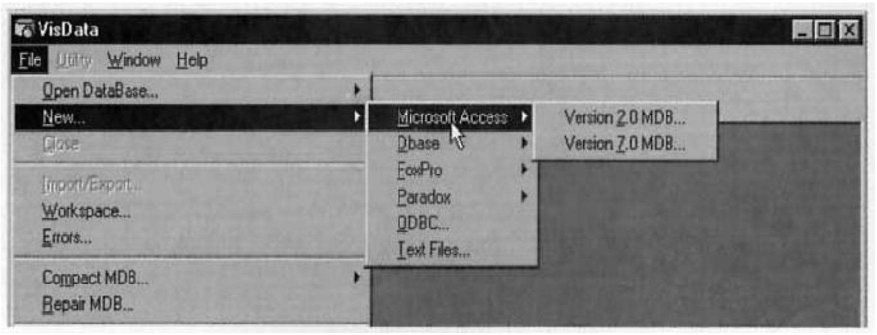
USING THE DATA-BOUND LIST BOX AND COMBO BOX CONTROLS

The data-bound list box and data-bound combo box controls look like the standard list box and combo box controls. They are often used with data-entry forms to speed data entry and ensure that valid data is entered. These controls automatically fill with a column from a table or recordset after you set a few properties. Note that the data-bound controls display data from a table or a recordset. Therefore, three methods (AddItem, Clear, and RemoveItem) and two properties (ListCount and Sorted) used with the regular list box and combo box are not available with the data-bound controls. The count of items displayed in a data-bound control is determined by the RecordCount property of the recordset.

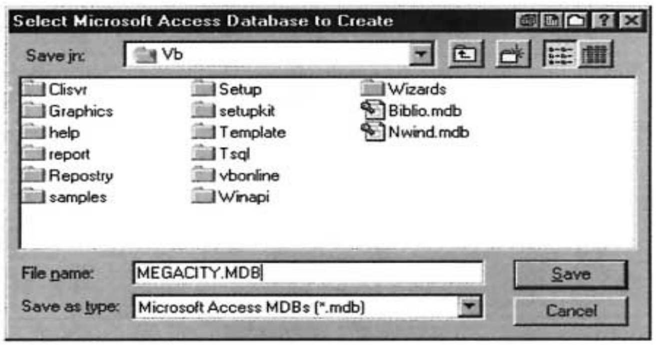
Two key properties determine the entries of a data-bound list or combo box—**Row- Source** and **ListField**. RowSource specifies a data control, and ListField specifies a field from the data control's table or recordset. That field will be used to fill the list.

CREATING A DATABASE WITH VISUAL DATA MANAGER

You invoke Visual Data Manager (VisData) from Visual Basic by pressing Alt/Add-Ins/Visual Data Manager. The first two entries of the File menu of VisData are Open Database, used to view and alter an existing database, and New, used to create a new database.



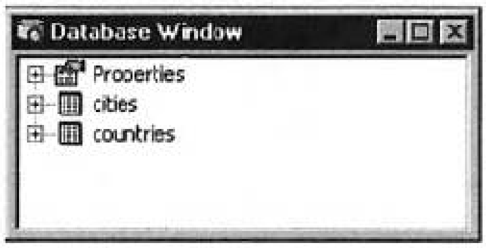
Let's focus on creating a new database. After you select New, you are presented with a drop-down menu used to choose the type of database as shown in figure. Choose Microsoft Access and then specify a version. (Version 7.0 is the latest version of Access, the one that comes with the Professional Edition of Office 97.) Then a standard file-naming dialog box titled "Select Microsoft Access Database to Create" appears



File name: MEGACITY.MDB

Save as type: Microsoft Access MDBs (*.mdb)

After you name the database, say as MEGACITY.MDB, and click Save, the Database window and SQL Statement box appear. We will work solely with the Database window.



Suppose you want to create a database with two tables. Starting from the Database window, the basic steps are as follows:
1. Create the database and the two tables.
(a) Click on the right mouse button within the Database window. Click on New
Table and use the Table Structure window to name the first table and to specify the fields and their data types.
(b) Repeat Step 1 for the second table.
2. (Optional) Specify a primary key for a table.
(a)Highlight the table name in the Database window.
(b) Press the right mouse button and choose Design to invoke the Table Structure window.
(c) Press the Add Index button to invoke the Add Index window and follow the steps listed after the figure.
3. Place records into a table.
(a) Double-click on the table in the Database window to invoke the Table window
(b) Follow the directions listed after Figure.

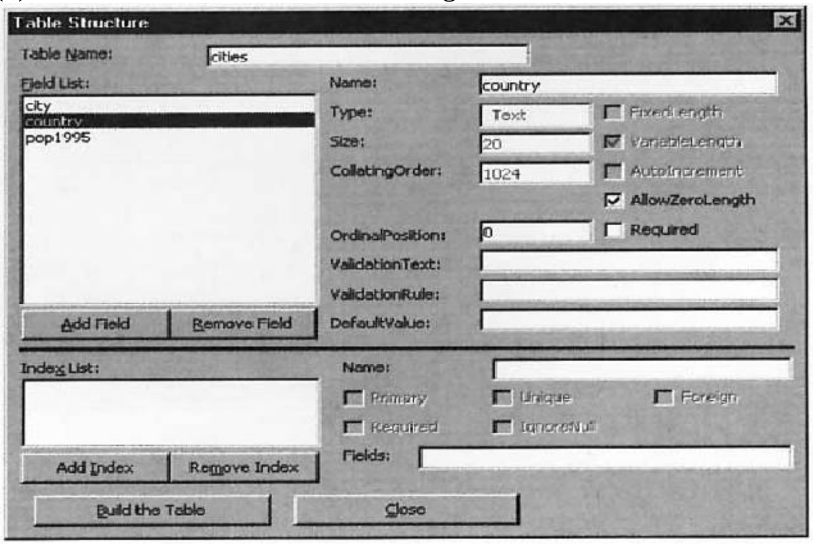


Table Structure

Table Name: cities

Field List:
city
country
pop1995

Name: country
Type: Text
Size: 20
CollatingOrder: 1024
OrdinalPosition: 0
ValidationText:
ValidationRule:
DefaultValue:

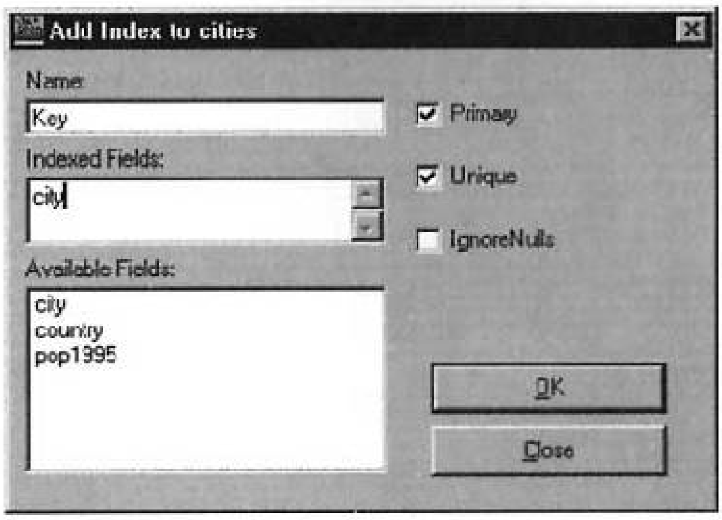
[ ] FixedLength
[✓] VariableLength
[ ] AutoIncrement
[✓] AllowZeroLength
[ ] Required

Index List:

Name:
[ ] Primary
[ ] Unique
[ ] Foreign
[ ] Required
[ ] IgnoreNull
Fields:

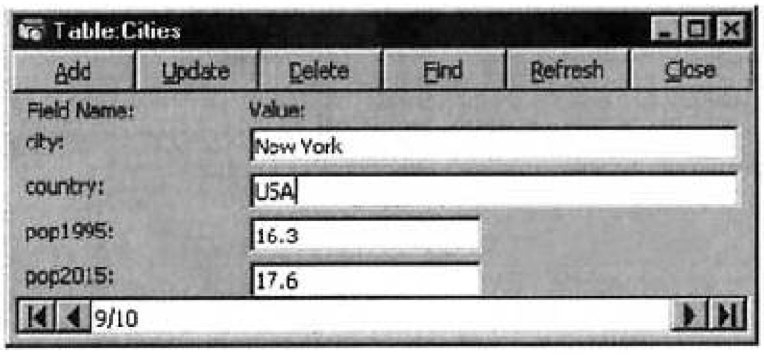
How to use the Table Structure window
1. Type the name of the table in the Table Name text box.
2. Click on the Add Field button. (An Add Field window will be displayed.)
3. Type the name of a field in the Name text box.

4. Click on the down arrow of the Type combo box and select a type from the dropdown list. (We use primarily the type “Text” for string, and “Single” or “Integer” for numbers.)
5. If the data type is “Text,” type the length of the largest possible string needed in the Size box. (For instance, length 20 should suffice for names of cities. The length must be no longer than 255 characters.)
6. Press OK to add the field to the table.
7. Repeat Steps 3 through 6 until you have added all the fields to the table. When you are through, click on the Close button to return to the Database window.
8. To delete a field, highlight the field by clicking on it in the list box and then press the Remove Field button.
9. When all fields have been specified, press the “Build the Table” button to save the table and return to the Database window. (If we later decide to add a new field or delete an existing field, we can return to the Table Structure window by highlighting the table in the Database window, clicking the right mouse button, and choosing Design.)



How to use the Add Index window to specify a primary key

1. Type in a name for an index, such as principal, click on the field which is to be the primary field, and place check marks in the Primary and Unique check boxes.
2. Click OK and then click Close.
3. Press “Build the Table” or Close to return to the Database window.



New York
USA
16.3
17.6

How to use the Table window

1. If the table has no records, the Value text boxes will be blank. To add a record, type data into the fields and press the Update button.
2. To add a new record to the end of a table that already contains records, click the Add button, type data into the fields, and press the Update button
3. To view an existing record, use the navigator buttons (identical to those of the data control) or the Find button to move to a record of a specified value.
4. To remove the current record from the table, click the Delete button. The record is immediately deleted and cannot be recovered.
5. To edit an existing record, make changes in the data and click on the Update button.
6. Press the Close button to return to the Database window.

At any time, the main section of the Database window contains a list of the tables you have specified. Initially the main section is blank except for Properties.
(This is a list of properties pertaining to the database.) If you click the right mouse button while highlighting a table, a menu will appear with the following options:

Menu Item	Use
Open	Open the table to allow records to be added
Design	Specify the design (number and types of fields).
Rename	Rename the given table.
Delete	Remove the given table from the database.
Copy Structure	Copies the structure of the given database with or without the data currently contained in the database.
Refresh List	Redisplay the list in the Database window.
New Table	Open the Table Structure window.
New Query	Open the Query Builder window.

Review & Self Assessment Question

Q1-What is Database?
Q2- What do you mean by Validation Event?
Q3-What is Primary Key?
Q4-Describe Foreign Key?
Q5-What do you mean by Structure Query Language?

Further Readings

Visual basic by David I Schneider
Visual basic by Steven Holzner
Visual basic by Bill Sheldon
Visual basic by Billy Holls
Visual basic by Rob Windsor

BIBLIOGRAPHY

Visual basic by David I Schneider
Visual basic by Steven Holzner
Visual basic by David McCarter
Visual basic by Gaston C Hillor
Visual basic by Todd Herman