



SURESH
GYAN VIHAR
UNIVERSITY
Accredited by NAAC with 'A+' Grade

Bachelor of Computer Application

(B.C.A.)

Programming in Visual Basic

Semester-iv

Author- Manish Somavanshi (Bhosale)

SURESH GYAN VIHAR UNIVERSITY
Centre for Distance and Online Education
Mahal, Jagatpura, Jaipur-302025

EDITORIAL BOARD (CDOE, SGVU)

Dr (Prof.) T.K. Jain
Director, CDOE, SGVU

Dr. Dev Brat Gupta
*Associate Professor (SILS) & Academic
Head, CDOE, SGVU*

Ms. Hemlalata Dharendra
Assistant Professor, CDOE, SGVU

Ms. Kapila Bishnoi
Assistant Professor, CDOE, SGVU

Dr. Manish Dwivedi
*Associate Professor & Dy, Director,
CDOE, SGVU*

Mr. Manvendra Narayan Mishra
*Assistant Professor (Deptt. of Mathematics)
SGVU*

Ms. Shreya Mathur
Assistant Professor, CDOE, SGVU

Mr. Ashphaq Ahmad
Assistant Professor, CDOE, SGVU

Published by:

S. B. Prakashan Pvt. Ltd.

WZ-6, Lajwanti Garden, New Delhi: 110046

Tel.: (011) 28520627 | Ph.: 9205476295

Email: info@sbprakashan.com | Web.: www.sbprakashan.com

© SGVU

All rights reserved.

No part of this book may be reproduced or copied in any form or by any means (graphic, electronic or mechanical, including photocopying, recording, taping, or information retrieval system) or reproduced on any disc, tape, perforated media or other information storage device, etc., without the written permission of the publishers.

Every effort has been made to avoid errors or omissions in the publication. In spite of this, some errors might have crept in. Any mistake, error or discrepancy noted may be brought to our notice and it shall be taken care of in the next edition. It is notified that neither the publishers nor the author or seller will be responsible for any damage or loss of any kind, in any manner, therefrom.

For binding mistakes, misprints or for missing pages, etc., the publishers' liability is limited to replacement within one month of purchase by similar edition. All expenses in this connection are to be borne by the purchaser.

Designed & Graphic by : S. B. Prakashan Pvt. Ltd.

Printed at :

Syllabus

Programming in Visual Basic

Learning Objectives

- learn features of Visual Basic and concept of programming.
- Get familiar with VB Interfaces, viz. Menus, Toolbar, Toolbox, different VB windows
- Work with VB controls to place them to VB forms and make executable files.
- Learn to get concept of algorithm and flowchart and write simple programs by using different types of control and loop statements

UNIT-I

Introduction to Visual Basic, Integrated development environment features – Forums – Controls – Events – Methods – Properties - Uses of Property Window – Code Window (Code Behind File) – Variable declaration.

UNIT-II

Scope of Variables – Constant – Array – Loops in Visual Basic: For ... Next, While, Do...While - Select statements: if...end if - if...else if...end if - Select...Case End Case –

UNIT-III

Standard Controls: Form - Text Box – Command Button – Label Box – Check Box – Frame Control – Combo Box – List Box – Radio Button - Image Control - Picture Box – Timer.

UNIT-IV

File System – Drive, DirList, File List Box – Introduction to Built-in-Active X control tool bar – Tree view – Menu Editor – Command dialog control – Rich Text Box.

UNIT-V

Introduction to Database – MS Access – Data Grid (Accessing Data Base data) – Open data base connectivity – Introduction to Dot Net: IDE – Execution Procedures – CLR – CTS. Text and

Reference

- Mastering Visual Basic 6 – BPB Publications, New Delhi.
- Mohammed Azam, Programming with Visual basic 6.0 – Vikas Publishing House.
- Test Your Vb.Net Skills: Language Elements Part 1 Paperback – 1 Dec 2000 by Yashavant P. Kanetkar (Author), Asang Dani, BPB Publications, New Delhi.

Contents

1. Getting Started with V.B.	34
1. Introduction.....	1-1
2. Installing of Visual Basic 6.0.....	1-3
3. Object Oriented Concept.....	1-9
4. Event Driven Programming Language.....	1-10
4.1 Events Related with Mouse and Keyboard	1-11
5. Reviewing the Basics of Forms and Controls.....	1-12
6. Working with Properties.....	1-23
6.1 Studying the Events of a Form	1-25
6.2 Working Code for Events	1-27
6.3 Planning the Design	1-30
2. Constants, Variables, Operators, Control Structure, Looping	70
1. Introduction.....	2-1
2. Data types.....	2-2
2.1 Built in Data Types	2-2
2.2 User Defined Data Types (UDT)	2-4
3. Variables.....	2-6
3.1 Rules to Define Variable	2-6
3.2 Declaring Variable	2-7
3.3 Using Option Explicit Statement	2-7
3.4 Variable Scope	2-8
3.5 Variable Duration	2-9
4. Constant.....	2-10
5. Operators.....	2-11
5.1 Arithmetical Operators	2-12
5.2 Relational Operators	2-13
5.3 Logical Operators	2-13
5.4 Differences between the Two Concatenation Operators	2-14
5.5 Precedence of the Operators	2-15
6. Expression.....	2-16
7. Comments.....	2-17
8. Control Structures.....	2-18
8.1 If	2-18
8.2 Using If...Then...Else Statements	2-20
8.3 Using Nested If Statements	2-24
8.4 Select Case	2-25
9. Looping.....	2-28
9.1 Do loop	2-29
9.2 For - Next	2-36
9.3 While - Wend Loop	2-39
9.4 Nesting Control Structures	2-40
10. Array.....	2-42
10.1 Declaring Array	2-42

- 10.2 Single Dimensional Arrays 2-44
- 10.3 Multidimensional Array 2-45
- 10.4 Dynamic Array 2-47
- 10.5 Control Array 2-48

11. Functions (Built in and user defined)	2-52
12. User Defined Functions	2-59
Solved Programs	2-61

3. Working with Control 38

1. Introduction.....	3-1
2. Adding Controls on Form.....	3-2
3. Working with Properties and Methods of Each Control	3-3
4. Creating MDI Applications	3-22
4.1 Working with Multiple Forms 3-23	
4.2 Loading, Showing and Hiding Forms 3-26	
4.3 Setting the start-up Form 3-27	
4.4 Creating Forms in Code 3-28	
4.5 Arranging MDI Child Window 3-29	
4.6 Opening new MDI Child Window 3-31	
4.7 Creating Properties in a Form 3-32	
4.8 Creating a Method in a form 3-33	
Solved Programs	3-34

Somavanshi, Jain

4. Working with ActiveX Controls and Menus 70

1. Introduction.....	4-1
2. Creating Status Bar for your Program	4-2
3. Working with Progress Bar	4-6
4. Working with Toolbar and Setting up the Image List Controls	4-9
5. Study of Different Dialog Boxes.....	4-22
6. Creating a Menu System	4-32
6.1 Designing the Menu 4-33	
6.2 Creating the Menu with the Menu Editor 4-36	
6.3 Adding Shortcut and Access Keys to Menu Items 4-48	
7. Creating and Accessing Popup Menu	4-49
7.1 Creating Pop-up Menu 4-50	
8. Adding or Modifying Menu at Run Time (Dynamic Menu)	4-57
9. Adding Menu Item for MDI Child Form	4-63
Solved Programs	4-64

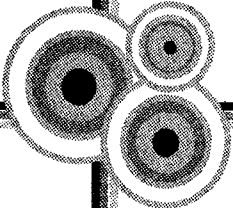
Somavanshi

5. Working with Database 42

1. Introduction.....	5-1
2. Data Control	5-2
2.1 Studying the Properties and Methods of Data Control 5-3	
2.2 Connectivity with MS-Access and Operations of Database through Coding 5-5	
3. ADO Data Control.....	5-8
3.1 Connecting with Oracle 5-13	
3.2 Report Generation 5-24	
4. Developing ADO Application through ADODC and Coding	5-31
Solved Programs	5-35

Somavanshi

GETTING STARTED WITH V.B.



1. Introduction

Visual Basic 6.0 is one of the most popular programming languages in the market today, created by Microsoft for building stand alone Window-based GUI applications. VB is used for multiple purpose in development. We can develop from a lightweight application to a full-fledged enterprise development with VB 6. VB is an object oriented programming development system. It has many predefined controls like menus, listbox, textbox etc. With these controls you can create a user interface for a particular application and can write a code to carry out the actions associated with each control on the form.

VB6 provides MDI i.e. Multiple Document Interface whereas before VB6 there was SDI i.e. Single Document Interface. It means all windows are a part of desktop. We can move them separately anywhere on the screen. There is no parent window for all the windows. From VB 5 there is MDI environment. VB 6 gives you a complete windows application development system in one package. It includes tools you can use to write and compile help files, ActiveX controls and even Internet applications VB itself is a windows application. You will use running VB programs to create other programs. VB includes controls that are tools on the toolbox window that you place on the form, to interact with the user and control the program flow. A program is a set of instructions that makes the computer do some work such as perform; accounting. A project is a collection of files you create to compose your windows application. An application is a collection of one or more files that compile into an executable program.

2

Apr.2013 – 4M

Why visual basic is called as GUI Application.

Apr.2012 – 4M

Write short notes: GUI (Graphical User Interface).

Visual Basic- GUI Application

Graphical User Interface (GUI) is a program interface that takes advantage of the computer's graphics capabilities to make the program easier to use. Well-designed graphical user interfaces can free the user from learning complex command languages. On the other hand, many users find that they work more effectively with a command-driven interface, especially if they already know the command language.

Graphical user interfaces, such as Visual Basic, feature the following basic components:

- i. **Pointer:** A symbol that appears on the display screen and that you move to select objects and commands. Usually, the pointer appears as a small angled arrow. Text -processing applications, however, use an I-beam pointer that is shaped like a capital I.
- ii. **Pointing device:** A device, such as a mouse or trackball, that enables you to select objects on the display screen.
- iii. **Icons:** Small pictures that represent commands, files, or windows. By moving the pointer to the icon and pressing a mouse button, you can execute a command or convert the icon into a window. You can also move the icons around the display screen as if they were real objects on your desk.
- iv. **Desktop:** The area on the display screen where icons are grouped is often referred to as the desktop because the icons are intended to represent real objects on a real desktop.
- v. **Windows:** You can divide the screen into different areas. In each window, you can run a different program or display a different file. You can move windows around the display screen, and change their shape and size at will.
- vi. **Menus:** Most graphical user interfaces let you execute commands by selecting a choice from a menu.

In addition to their visual components, graphical user interfaces also make it easier to move data from one application to another. A true GUI includes standard formats for representing text and graphics. Because the formats are well-defined, different programs that run under a common GUI can share data. This makes it possible, *for example*, to copy a graph created by a spreadsheet program into a document created by a word processor.

2. Installing of Visual Basic 6.0

Before you start working on Visual basic you need to install it on your PC. Lets see the process of installation of Visual Basic 6 on your computer system. Visual basic comes only on CDs. It is a part of visual studio product. There are few steps you have to follow while installing VB.

1. Insert Visual Studio CD. This CD contains automated “*setup.exe*” program. You need to run this setup to install VB.
2. Run *setup.exe*. When you run setup program it will ask your *name* and the *name* of the company.
3. Once you start running the setup it popsup some windows asking few questions. If you are the first user, without making any changes, *click* on the next button in the dialog box.

When you start running setup following window will appear on the screen. If you are the first user *click* on the “NEXT” command button on the screen shown in the *Figure 1.1*

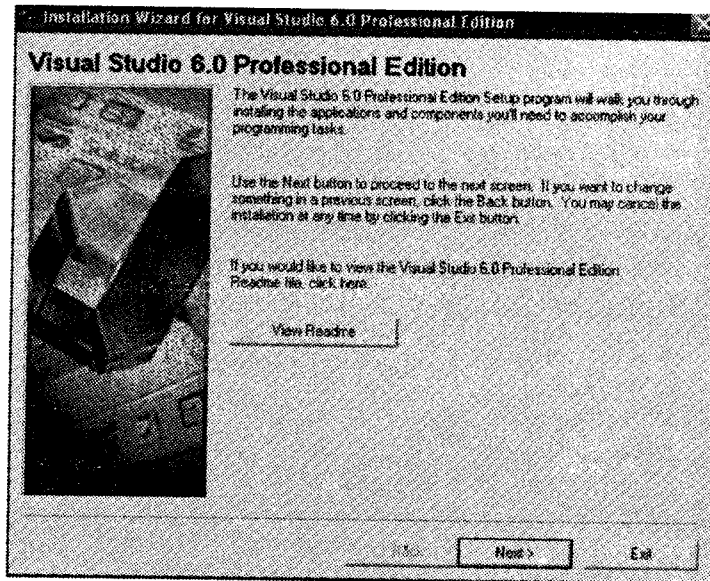


Figure 1.1

4. After clicking NEXT button, now setup will ask you to accept an agreement. There are two option buttons on the screen as shown in the *Figure 1.2*.
 - i. “*I accept the agreement*” and
 - ii. “*I don't accept the agreement*”. If you agree, select the “*I accept the agreement*” option button.

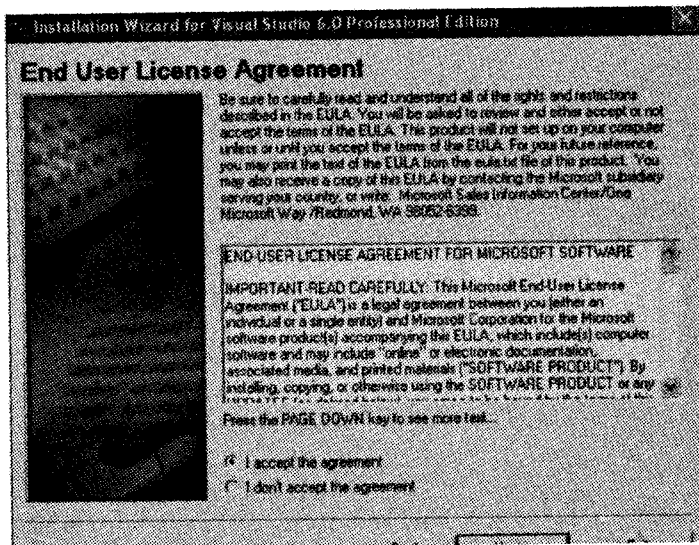


Figure 1.2

5. The next window that appears is the "Product Number and User ID" window. Here you are supposed to enter *Product ID number* in the first text box, *your name* and *your Company name*. After entering this information *click* on the *NEXT* button as shown in *Figure 1.3*.

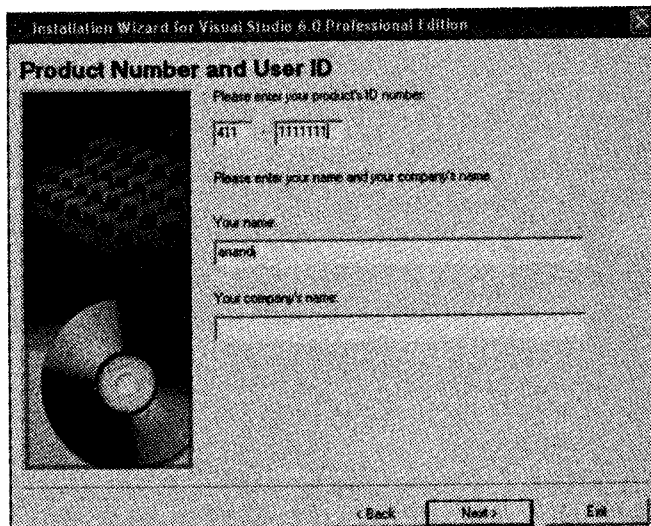


Figure 1.3

- Now installing wizard will ask you to select for edition. By default “*Install Visual Studio 6.0 Professional Edition*” option is selected. If it is not then select accordingly desired edition and go to *NEXT* window. (Refer *Figure 1.4*).

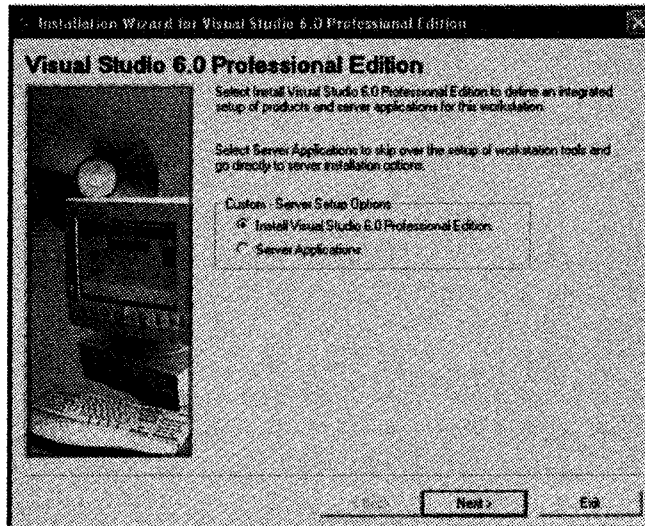


Figure 1.4

- After selecting edition, installation wizard will start installing the files. At this step you need to select the location for those files. You can choose the location of the file that is common among visual studio 6.0 application. The common files should be stored in the folder called “common”. If you wish to change the default location you may *click* the browse button to explore the location from your hard drive disk and choose the new location (try to follow the default settings as shown in *Figure 1.5*) and *click* on the *NEXT* button to proceed further.

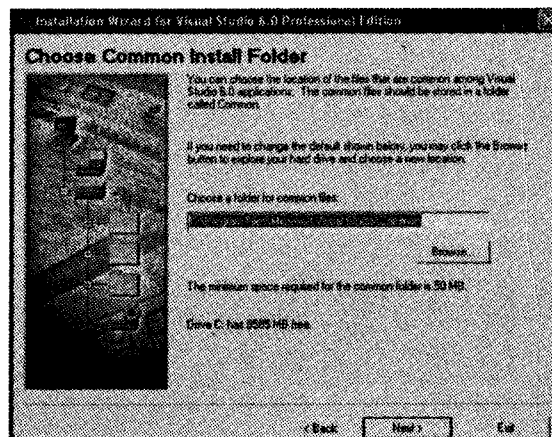


Figure 1.5

8. Read the license agreement and select the “*Continue*” button to start installing Visual Studio 6.0, or select “*Exit Setup*” button to come out from the setup wizard. (Figure 1.6)

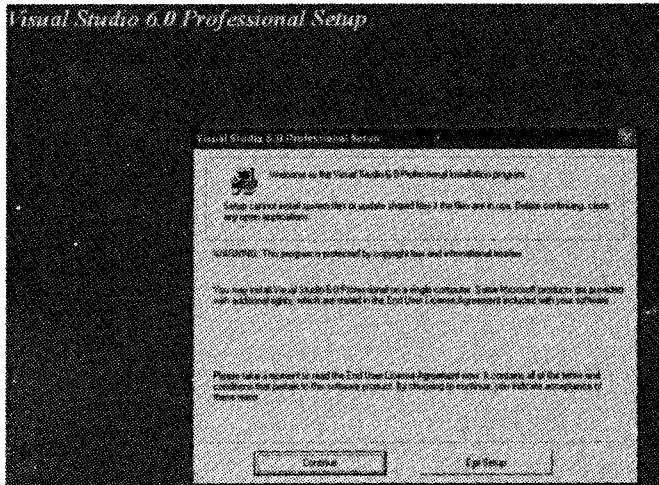


Figure 1.6

9. Note down your product ID if you are using licensed product of the Visual studio 6.0, and then *click* on the “*OK*” button.

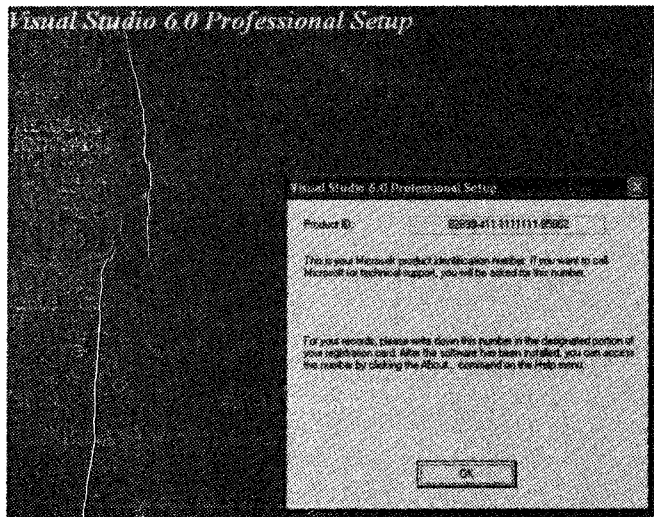


Figure 1.7

10. Now in the Figure 1.8 you can see three buttons. The *large icon* or *button* is to start the installation; the “*Change Folder*” button is to change the path of visual basic. By default wizard will take “C:\Program File\...” path. If you wish to change the path you can do that by

clicking “Change Folder” button and if you want to exit the setup click the button “Exit Setup” at the bottom of the screen.

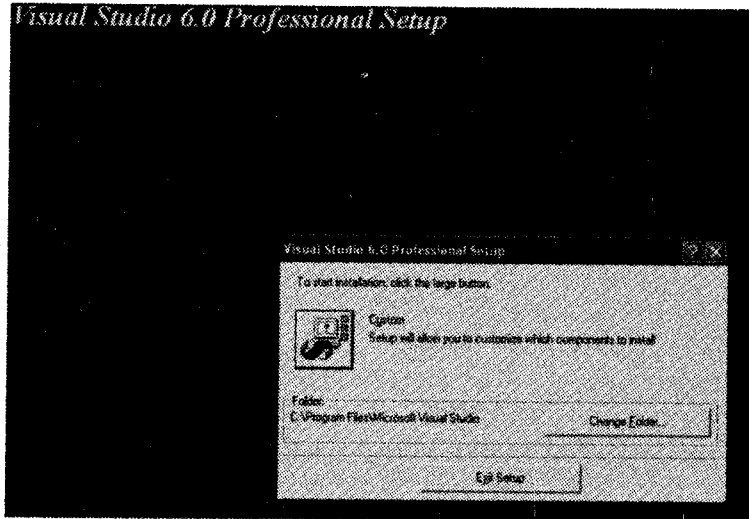


Figure 1.8

11. At this stage wizard will give you the list of options which you can install. You need to select the Item(s) you want to install. You can choose multiple items to install. Along with this you can see different buttons on the screen. You can select all the options at a time by clicking “Select All” button. With “change folder” button you can change the default folder where Visual studio will install all selected items. “Continue” will continue the installation process and “Cancel” will cancel the process of installation. (Figure 1.9)

In our case make sure that the first option in the list i.e. “Microsoft Visual Basic 6.0” is selected. And then click on the “Continue” button to continue the installation of VB.

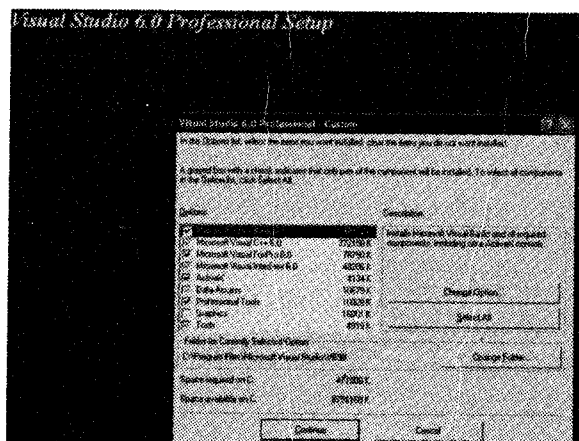


Figure 1.9

12. At this step select “*Register Environment Variables*” check box. Click “**OK**” and go ahead. (Figure 1.10)

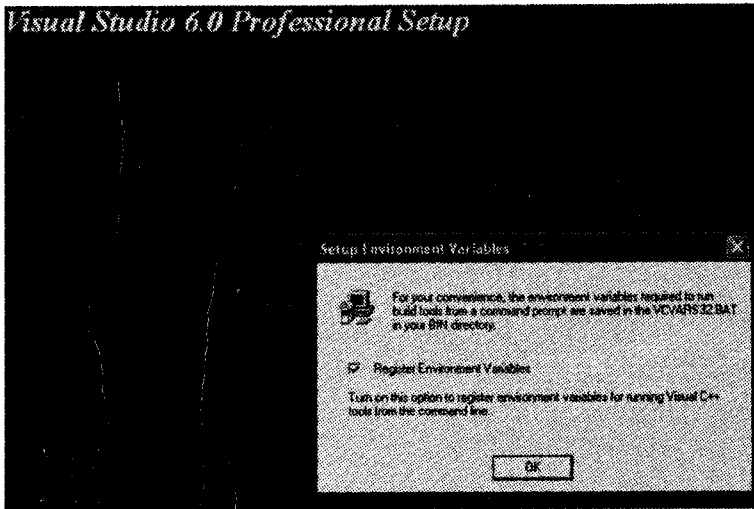


Figure 1.10

13. This is the last step of installation. In Figure 1.11 you can see the progress bar of the installation process. Once it reaches 100% you will get Microsoft Visual Basic 6.0 installed on your PC. If required restart the PC. Now you are ready to use VB.

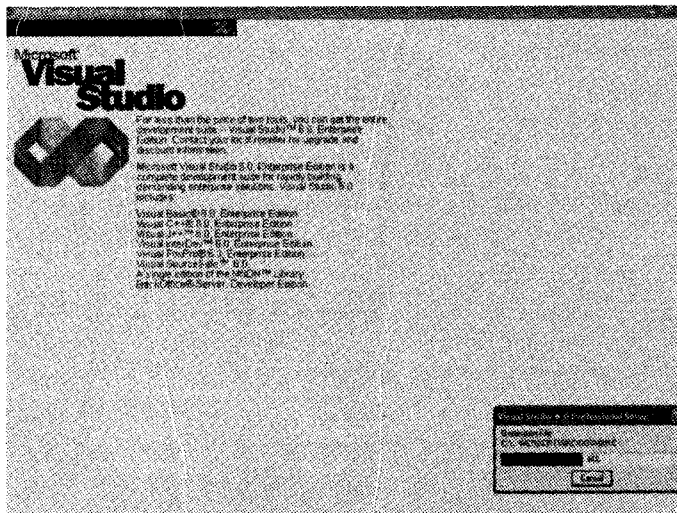


Figure 1.11

14. To open VB, select *Start* → *Programs* → *Microsoft Visual Studio* → *Microsoft Visual Basic 6.0*

3. Object Oriented Concept

1

Apr.2011 – 4M
Write short note on:
Object Oriented
Programming.

Object Oriented Programming (OOP) is a more advanced aspect of visual basic. Object Oriented Programming is quite simple, in fact it is probably simpler for those who have never programmed before than for those with long experience of traditional structure programming language like FORTRAN / PASCAL.

In Object Oriented Programming style you break a problem down into small parts and solve them individually. Here everything is considered as an object and every variable or function is a property of an object.

Object Oriented Programming (OOP) is more than just a programming concept. It is a way of thinking about applications. It is learning to think of applications not as procedures, but objects and real entities. In programming, an object is a run time instance of code and data that comprise some sort of logical grouping, usually referred to as a business entity. Another great advantage is improved code readability, reliability, and adaptability. *For example*, You don't have to know how a remote control works, you just need to know the number of channel you want to watch. Like wise in case of OOPs you can directly use an object without knowing the structure of that object.

In Visual Basic, we use classes to define components. Once created, and populated with data, a class becomes an object with properties and methods. So far, so good, Visual Basic can do classes with properties and methods.

Object oriented design concepts

1. **Inheritance:** Inheritance is the process by which one object can acquire the properties of another object without rewriting the code. It means that we can add additional features to an existing class without modifying it. This is possible by deriving a new class from the existing one. This is important because it supports the concept of classification or reusability.

Inheritance allows one class to inherit the functionality of another without having to rewrite the code. This is similar to having some qualities of its parent, while still functioning and appearing quite unique. Inheritance as a programming concept works the same way.

2. **Polymorphism:** Polymorphism is a Greek term, *poly* means 'many' and *morph* means 'forms/bodies'; means the ability to take more than one form. An operation may exhibit different behaviors in different instances. The behaviors depend upon the types of data used in the operation. The polymorphism is implemented using overloading. We implement polymorphism in Visual Basic, by overriding methods.
3. **Aggregation:** Aggregation is a sort of symbiotic relationship between objects. In aggregation, a host object acts as a liaison between the outside world and an inner object. This can be accomplished in several ways. One is by having both objects implement the same interface.

Since they contain the same interface, when a host object is called, it can in turn make a call to the inner component's same method (delegation), in effect, forwarding the call. Another technique allows the method of the inner component to be directly exposed to the outside world (aggregation).

4. **Encapsulation:** Encapsulation is a method of data abstraction that allows us to change data through a representation, not the real thing. In Visual Basic, we can accomplish this using properties and methods.
5. **Overloading:** Overloading allows us to provide multiple procedures that have the same *name* and do similar things. Overloading allows varying functionality of the same method by providing more than one signature. *For example*, with both methods, you could create several objects that have the same interface and different implementation, but with overloading, you can accomplish this in totally different ways. Instead of overriding, thus rewriting the method for each object, you could provide several methods named the same, but with different parameters in the same interface. Visual Basic doesn't support overloading methods.

4. Event Driven Programming Language

3

Apr.2013 – 8M

What is Event Driven Programming in VB?
Explain with example.

Apr.2010 – 4M

What is Event Driven Programming Language?

Oct.2010 – 4M

Write a short note on:
Event Driven Programming

Visual basic uses an event-driven architecture. *For example*, whenever the user clicks one of the mouse buttons, or types in a key on the keyboard, a signal called an event is created. The Windows system then checks event details such as event location, where the mouse *click* has occurred and then notifies the appropriate widget to take appropriate action.

VB supports an event driven programming style. You will experience this in this book step by step. In event-driven programming you will think according to the user action. i.e. "how and what should happen when user do this".

Then you would write the program to perform that task. This is called event-driven programming. You program according to what events the user would generate. Moving mouse, dragging a picture, clicking a button, and typing text are all events:

Consider the following example: Let's create a simple button called 'CmdClick' and write the code that will be executed when the button is clicked. The code below simply changes the *caption* of the button, when the button is clicked.

```
Private Sub CmdClick_Click ()
    CmdClick.Caption = "This is Button"
End Sub
```

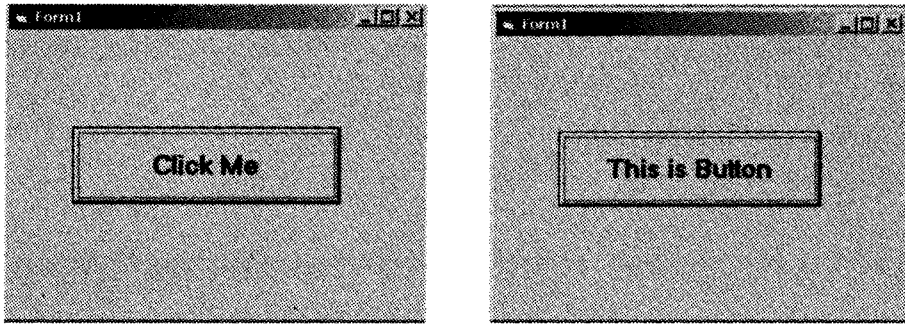


Figure 1.12

4.1 Events Related with Mouse and Keyboard

There are a number of event that we can use, *for example*, double-click, mouse move, mouse up and down events, got focus and lost focus events, key up and down events, drag and drop events, form load etc.

The program’s response to an action taken by the user is referred to as an event. The event is initiated by user and it is responded by the program. This complete process is called an event, and the code written for this event to happen is called as event procedure.

2

Apr.2012 – 8M
Discuss various events related with Mouse and Keyboard.

Apr.2010 – 4M
Write short note on: Keyboard events

The MouseDown, MouseUp, and MouseMove events enable the applications to respond to both the location and the state of the mouse. These mouse events are recognized by most controls.

Event	Description
MouseDown	Occurs when the user presses any mouse button.
MouseUp	Occurs when the user releases any mouse button.
MouseMove	Occurs each time the mouse pointer is moved to a new point on the screen.

A form can recognize a mouse event when the pointer is over a part of the form where there are no controls. A control can recognize a mouse event when the pointer is over the control.

When the user holds down a mouse button, the object continues to recognize all mouse events until the user releases the button. This is true even when the pointer is moved off the object.

Keyboard events: Visual Basic is an event driven programming language. It supports various mouse and keyboard events. The key event occurs when the user presses any key that corresponds to a

certain alphanumeric value or an action such as enter, spacing, backspace or so on. Each of those values or actions are represented by a set of codes known as the ASCII. Using the keyboard events, user can program different controls and forms to respond to various key actions.

*For example:*KeyUp, KeyDown, KeyPress

5. Reviewing the Basics of Forms and Controls

VB is popular for its simplicity. The most attractive thing about VB is its Graphical User Interface (GUI). It uses all the powerful features of the windows and provides powerful user interface. VB has an IDE i.e. Integrated Development Environment in which you can develop, run and debug your application. The components of an IDE are tool box, property window, form layout window, code window etc.

In this section, we will study all about the VB environment. VB6 has MDI i.e. Multiple Document Interface environment. That means all the windows are part of the large window called as parent window and others are child window. We cannot move child window outside of the parent window.

Let's have a tour to Visual Basic IDE

After installing Visual Basic, when you start it a dialog box asking about New Project type will appear. In that you can see different Tabs. Like *New*, *Existing* and *Recent*. *New* tab displays list of different types of projects. *Existing* tab will show the list of already existing project list on your disk. *Recent* will show list of recently used project. The screen will be as follows.

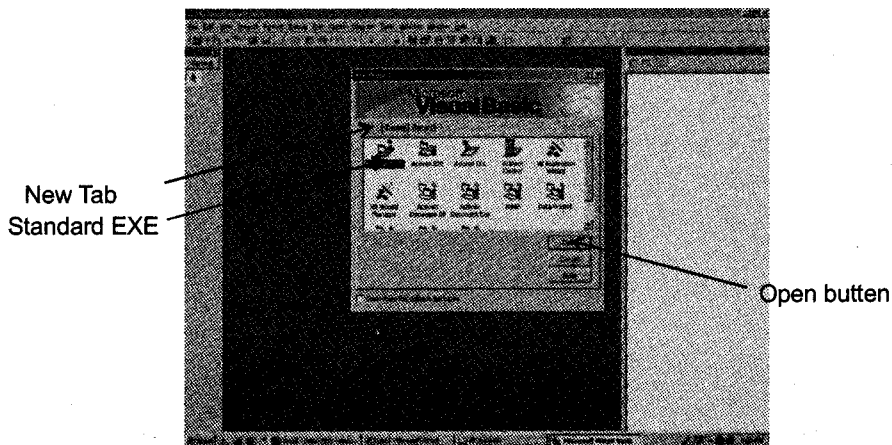


Figure 1.13

In this New Project dialog box you have two options i.e. either you open an existing project or you can create new one. Here we will create new project therefore select *New* → *Standard EXE* and click

on Open button. Standard EXE creates stand alone programs having .exe extension. This can be used like an independent application. After selecting open option, now you have VB's User interface with all required windows opened. This is called as IDE (Integrated Development Environment). Let's have a look on the IDE. Now your screen will look like this *Figure 1.14*.

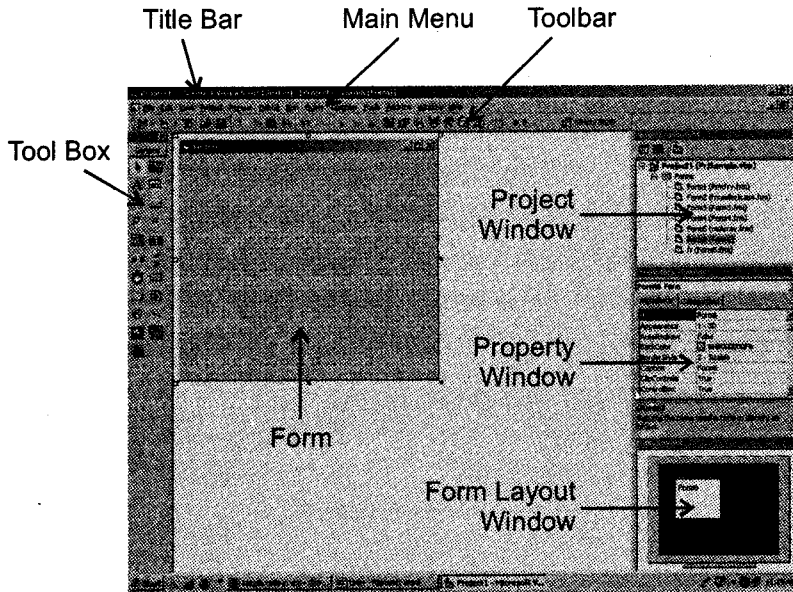


Figure 1.14

IDE is the screen that appears first when you start visual basic. All the necessary windows you need to develop any application are integrated under single window therefore it is called as Integrated Development Environment. The different components of IDE are Menu Bar, Title Bar, Tool Bar, Tool Box Form (placeholder of all the controls), property window, project window, form layout window etc. which are arranged in a elegant way. You can change this IDE arrangement up to some extent to work according to your requirement. i.e. you can close or keep open the windows as per your preference. All the windows in this IDE are dockable at certain location of the screen and can be interlocked with each other. As you can see in the above screen. One window is docked with another window. *For example*, the project explorer window and the properties window are interlocked with each other. This default arrangement of the components is comfortable for the user to work. Or you can change this style and customize the IDE as you wish.

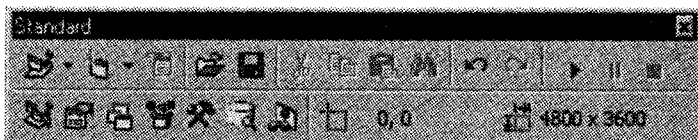
Now we will discuss all the components in details

1. **Title Bar:** At the top of the screen is the Title Bar. The title bar gives us information about what program we're using and what Visual Basic program we are working with.
2. **Main Menu:** Below the title bar is the Main Menu. This menu is very much similar to the menu in MS Word, Excel etc. If you have experience of working with these tools you must be

familiar with use of Menu and its different options. You can perform all types of operations using this menu.

- 3. Tool Bar:** Under the main menu is the Toolbar. You can see small icons/buttons with pictures. These buttons are the alternate to the options in main menu. To understand the meaning and use of that button you just put the cursor over the button for a while; a little help called 'tool tip text' will pop up and tell you what that particular button does. Like many windows application, Visual Basic has multiple toolbars available. You can easily customize your own toolbar to suit your needs. There are four built-in toolbars available in VB. They are Standard, Edit, Debug, and Form Editor. (Select View → Toolbar Menu to open any tool bar) By default, the standard toolbar appears immediately below the menu bar. All the tools bars looks like this

Standard Tool Bar



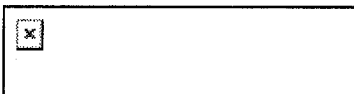
Edit Tool Bar



Debug Tool Bar



Form Editor



- 4. Project Explorer Window:** The project explorer window also called the project window gives you a tree-structured or list view of all the files which make the project. Through this project window you will get a bird eye view of your entire application. This window displays forms, modules, classes etc. When you want to work with a particular part of the loaded application, double *click* the component in the project window to activate and bring that component into foreground. You can also add or remove the items like forms, class modules from the project window by right-clicking in the window. In the project window (*Figure 1.15*) you can see three buttons named as View Code, View Object, Toggle Folder below the title bar. Using these buttons you can switch between different views. The left most button–View code button opens objects code window. Middle button–View object button opens the object itself and the

last button– Toggle Folder opens and closes folders in the project window. You can toggle on the folder view to separate different components in different folders. This is useful in case of very big application where you have many modules, forms and classes.

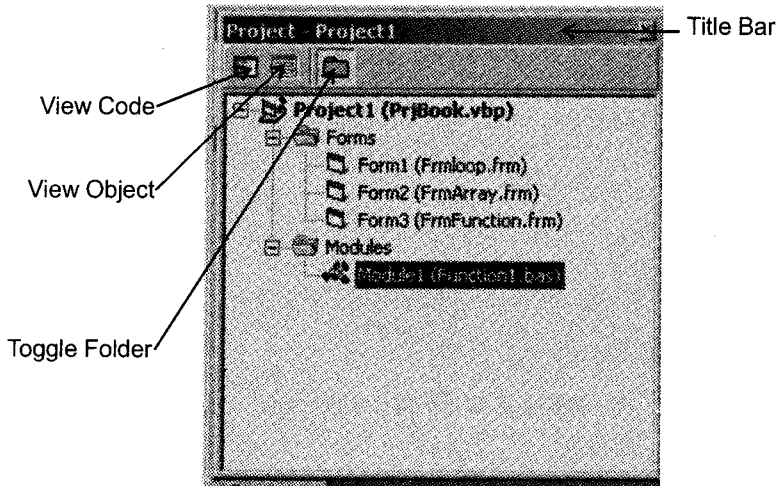


Figure 1.15

5. **Properties Window:** In VB every control has properties. Some of the properties are common for all controls whereas all the controls have some unique properties which makes that control different from other controls. Properties define the *appearance* and the behavior of the control to which they are associated. The properties of the control can be changed from this property window. The property window shows all the property and detailed descriptive information of the control, which is selected. (Figure 1.16) The *title bar* shows the *name* of the control which is currently selected and which properties have been shown in the property window. Below the title bar is *Control list* box which shows the control *name* and the type of control (e.g. Form3 Form). You can see two tabs below the control list box that are *Alphabetic* and *Categorized*. Both tabs contain same properties but in different manner. Alphabetic tab provides properties in alphanumeric ascending arrangement and categorized tab groups the properties on the basis of *appearance* and the behavior of the control to which they are associated. But I will suggest you to use alphabetic arrangement. Because all the properties are arranged alphabetically so it is easy to search or locate any property in this tab. The properties are displayed in two columns. The first column shows the *property name* and second column shows the current *value of the property*. This is called setting box. The properties of the control can be changed from the property window. Changing properties through property window are called as design time setting. To change the property just select the right hand side value of the property. Either you can type the value or VB will popup a list box to select the value. You can select any of the value from that list. At the bottom of the property window there is a note which shows the description of the currently selected property.

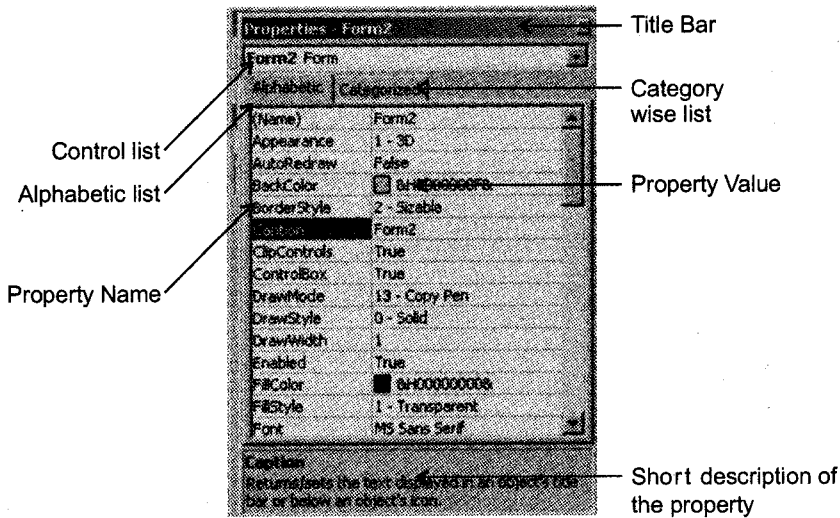


Figure 1.16: Property Window

6. **Form Layout Window:** The Form Layout Window (*Figure 1.17*) is used to position your form as you want them appear on the screen when you run your application. To open form layout window on the screen, *click* View Menu → Form Layout Window option. Now, in the Form Layout Window you can see your forms. You can position any of the form anywhere on the screen. *Click* on the form in the little screen and drag it to the desired position. This establishes the location of the form on your computer monitor when you run the application. In the following *Figure 1.17* you can see three forms. These forms are from same project. You can set different location for each form.

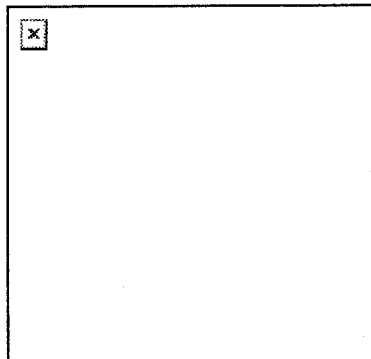


Figure 1.17: Form Layout Window

7. **Tool Box:** Tool box is located at the left of the screen. It contains all the controls which are used to build your application. The controls you see on the tool box are default controls. They

are installed at the time of installation of VB. If the toolbox window is not present on the screen, *click* View on the main menu, then Toolbox. The toolbox is simply a library of controls which you can place on your application. Once you've placed all the controls you need onto your applications forms, you can hide the toolbox to make room for working in the other elements of IDE. The Toolbox window is probably the first window you'll become familiar with because it lets you visually create the user interface for your applications. If you are working first time in Visual Basic then it is necessary that you should spend more time to understand all these controls and their properties. Properties are shown in the property window. As we have just now discussed in the above section. You can add more controls on the tool box.

To add more controls on tool box follow these steps

- Select Project Menu → Components. (Figure 1.18)
- From the dialog box select the control which you want to add.
- *Click* on the OK button.

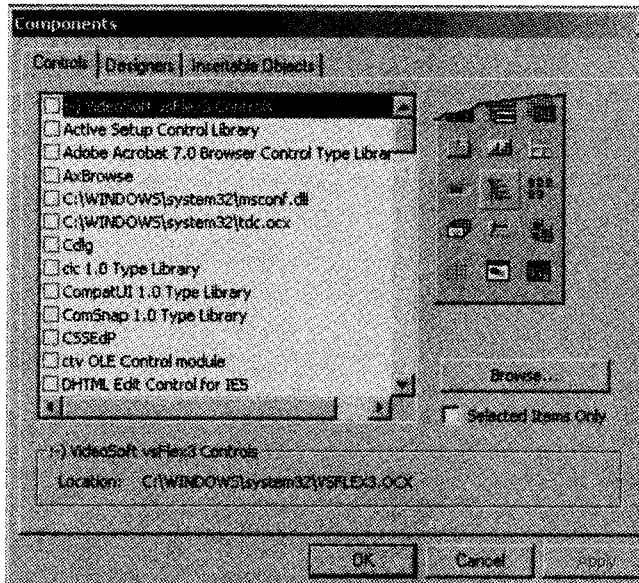


Figure 1.18: Components Window

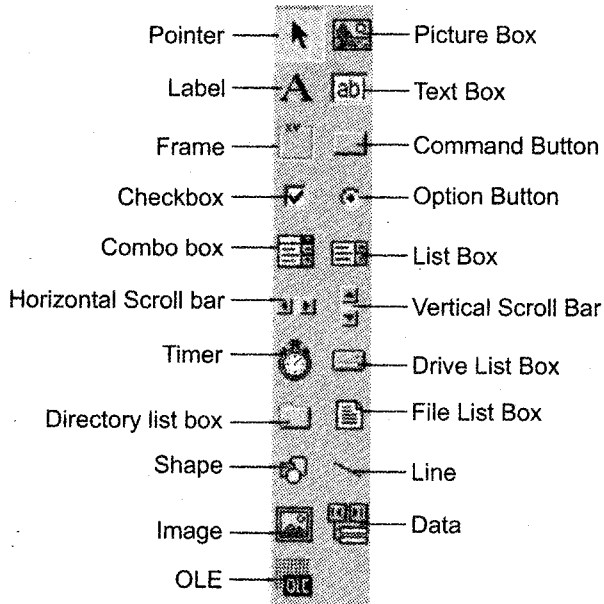











Figure 1.19: Tool Box

The Visual Basic Controls refer Figure 1.19 while you read the descriptions for the control.

Icon	Control	Icon	Control
	The Pointer isn't a control; <i>click</i> this icon when you want to select controls already on the form rather than create new ones.		Option Button controls are always used in groups, and you can select only one control in the group at a time.
	The Label control is used to display static text or text that shouldn't be edited by the user; it's often used to <i>label</i> other controls.		The ComboBox control is a combination of a TextBox and a ListBox control; ComboBox controls don't support multiple selections.
	The Text Box control is a field that contains a string of characters that can be edited by the user. It can be single-line or multiline. This is probably the most widely used control of any Windows application and is also one of the richest controls in terms of properties and events.		The ListBox control contains a number of items, and the user can select one or more of them. Instead of placing all the items on the form you can bind it in list box it saves the space on the form.
	The Command Button control is present in almost every form, e.g. OK and Cancel buttons.	 	The HScrollBar and VScrollBar is used to create scroll bars. But most of the controls enabled with their own scroll bars whenever necessary so these controls are used very rarely.

	<p>The specialty of the Timer control is it isn't visible at run time. Its only purpose is to regularly raise an event in its parent form. By writing code in the corresponding event procedure, you can perform a task in the background. This is the control having very less properties.</p>		<p>The DriveListBox, DirListBox, and FileListBox controls are often used together to create file-oriented dialog boxes. DriveListBox is a ComboBox-like control filled automatically with the names of all the drives in the system. DirListBox is a variant of the ListBox control; it shows all the subdirectories of a given directory. FileListBox is another special ListBox control; this control fills automatically with names of the files in a specified directory. While these three controls offer a lot of functionality, in a sense they have been superseded by the Common Dialog control, which displays a more modern user interface.</p>
	<p>The Picture Box control is used to display images in any of the following formats: BMP (bitmap), DIB, ICO (icon), CUR (cursor), WMF (metafile), EMF (enhanced metafile), GIF, and JPEG.</p>		
	<p>The Frame control is typically used as a container for other controls. You rarely write code that reacts to events raised by this control.</p>		<p>The Data control is the key to <i>data binding</i>, a Visual Basic feature that lets you connect one or more controls on a form to fields in a database table. The Data control works with Jet databases even though you can also use attached tables to connect to data stored in databases stored in other formats. But it can't work with ActiveX Data Objects (ADO) sources and is therefore not suitable for exploiting the most interesting database-oriented Visual Basic 6 features.</p>
	<p>The Check Box control is used when the user has to make a yes/no, true/false selection.</p>		<p>The OLE control stands for Object Linking and Embedding means making available the program which is not belonging to the visual basic. Such as a Microsoft Excel spreadsheet.</p>
	<p>The Shape and Line controls are used only to draw lines, rectangles, circles, and ovals on forms. They are generally used for design purpose. These controls never raise any events.</p>		

Actually these controls are used to make your application. But we can't use these controls alone. We need placeholder to put all these controls. The VB Forms works as placeholder for these controls. You have to put the controls on the form and then write appropriate code for the same. Controls should be arranged in elegant way on the form which gives meaning to your application. And then you can write events for the controls.

To display the controls on the form follow the steps given below

1. Create a new project. (*Refer Section 4*)
2. Now you have one default Form.

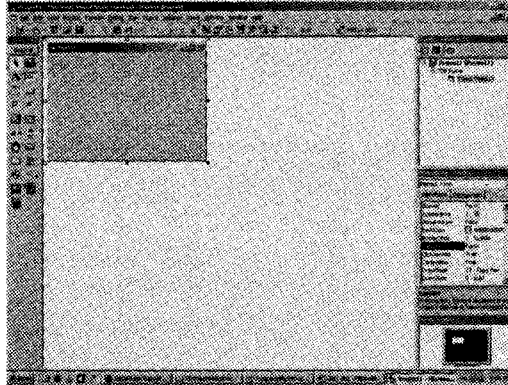


Figure 1.20: Blank Form

3. To display the controls on the form *click* the mouse on the control which you want to display on the form and then draw that control on the form. You can see the mouse pointer has been changed to plus sign. Drag and release the mouse pointer and you will get your controls displayed on the form. (The screen will look like *Figure 1.21*)

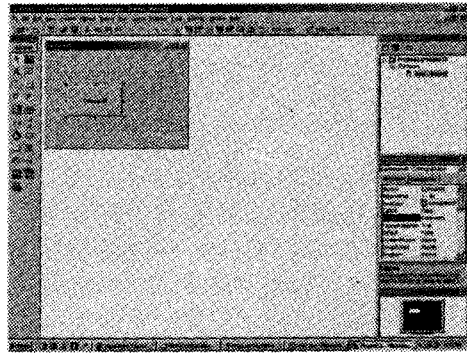
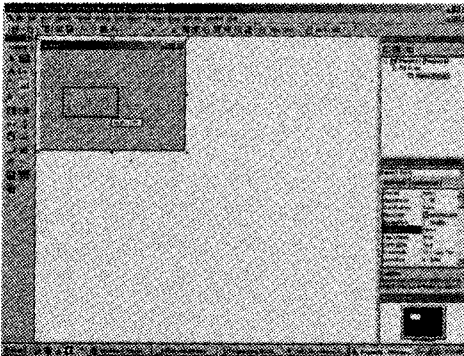


Figure 1.21: Drawing command button on the Form

4. Now you have command button displayed on the form. Follow the same procedure for other controls.

5.

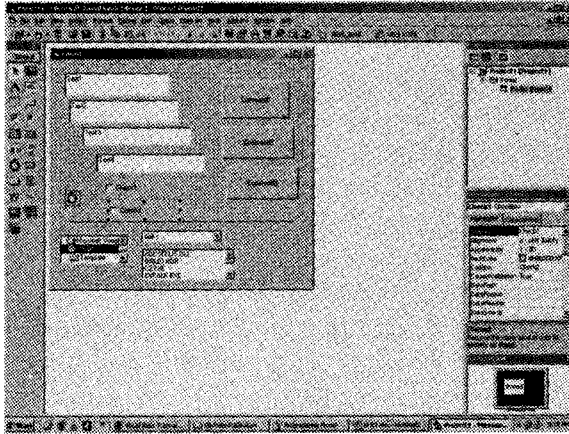


Figure 1.22: Form after designing

Once you finish designing your application you must be eager to start or run your application and want to see the result. In the following section we will discuss about how to start and stop the application.

Running a Visual Basic Project

There are different ways to start or run the project. They are: (Figure 1.19)

- Click on the triangle button on the standard toolbar as shown in the following Figure 1.23. This button looks like the Play button on a VCR, CD player, or cassette tape player. Or
- Press F5 button on the keyboard. Or
- Select Run Menu → Start option.

This starts the application and now you can see the result. You can now interact with your application. The program will perform events.

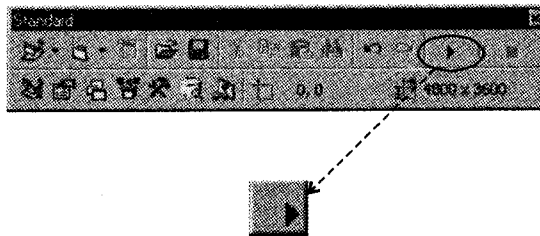


Figure 1.23: Start button

After running the form shown in the *Figure 1.22* it will look like the form shown in the *Figure 1.24*.

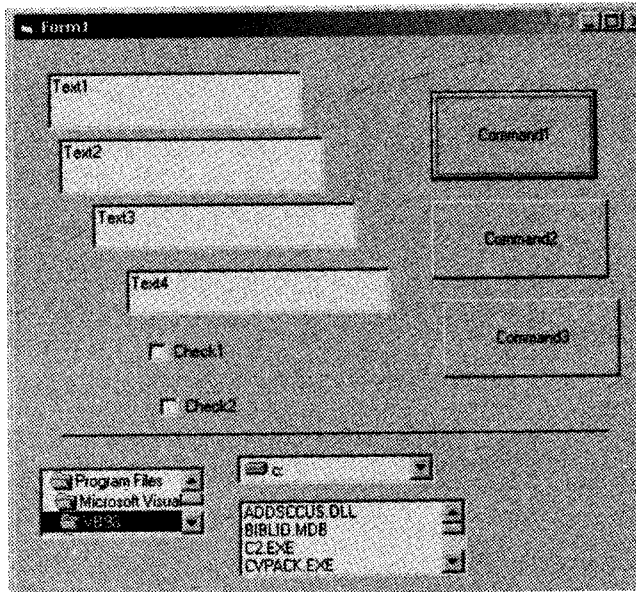


Figure 1.24: Run Form

Stopping a Visual Basic Project

If you want to stop there are many ways to stop a Visual Basic application. They are

1. By using the toolbar. (*Figure 1.25*) Look for a button that looks like the *Stop* button on a VCR, CD player, or cassette tape player. *Click* on this button. The project will stop and Visual Basic will return to design mode.
2. An alternate way to stop the project is to use the *Close* button found on the form. It is the little button that looks like an "X" in the upper right corner of the form.
3. Otherwise you can have your own button written code to stop an application on your form.

When you stop any project VB returns again in the design mode.

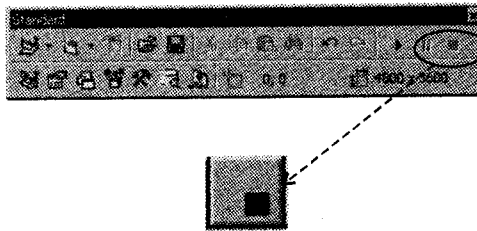


Figure 1.25: Stop button

6. Working with Properties

Properties are characteristics or attributes of any control. All VB controls have properties. All these properties are displayed in the property window. This window lists lots of different properties that you can use to change how a control looks and behaves. Before writing an event procedure for the control to respond to a user's input, you have to set certain properties for the control to determine its appearance and how it will work with the event procedure. You can set the properties of the controls in the properties window (i.e. design time) or at runtime.

For example, you have displayed one *Label* control on the form. You can change its properties like the *Appearance* property; it sets whether a control should look 'Flat' or 3D.

Another property is *Name*. We should give a meaningful name to the control so that when we start writing the code, we can refer to this control, and remember which it is. To change the name property, we modify the value next to the Name box at the top of the properties list.

There are some properties that are common for all the controls

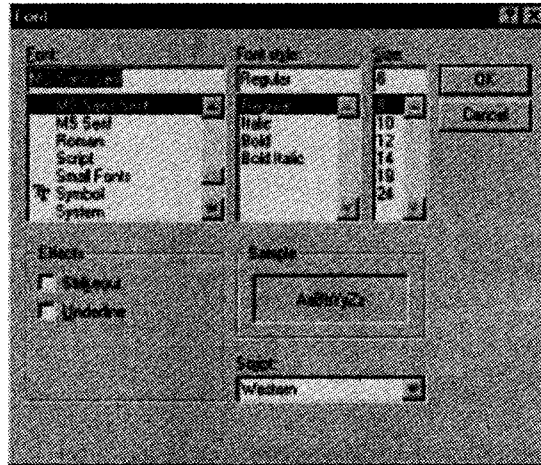
1. **Name:** Every control has *name* property. *Name* property is used in code to refer to the control when you want to manipulate its properties or methods. *For example*, if you *name* a command button cmdSave, you can write code for save. If there are multiple command buttons in the application it will be easy for programmer to write code for the specific command button according to its use. Otherwise you have to refer form every time.

Changing *name* property is not compulsory but it is a good programming practice to rename a control by a meaningful *name*. It is the VB style that prefixes the *name* with control type. Because of this it becomes easy to identify that control type when you write any event.

For example, you can prefix a TextBox control with the letters "txt", command button with "cmd" as shown in the above example cmdSave. *Label* with "lbl", option button with "opt" and so on.

2. **Enabled:** The enabled property of a control is a true/false property that you can set to determine whether or not the control can receive focus or respond to user-generated events such as the *Click* event. Many controls' appear "grayed" when you set their enabled property to false. *Label* control never gets focus, so its enabled property has no effect on whether the user can set focus to the *Label*. You can set a control's enabled property at both design time and runtime.
3. **Visible:** This property is true by default. If you set it to false the control will not be visible to the user when you start the application.
4. **Font:** This is the property that contains many properties within itself. Double-click the word "Font" in the control's properties window; *Click* the ellipsis button (...) to the right of the word

"Font" in the properties window. You will get a Font dialog box. Where you can set all the properties of font like size, face, style etc. The font dialog box looks like this



5. **Left:** The position of the left side of a control with respect to its container.
6. **Top:** The position of the top of a control with respect to its container.
7. **Height:** A control's height.
8. **Width:** A control's width.
9. **BackColor:** Set the background color of the control.
10. **ToolTipText:** Except form all the controls have this property. When mouse is paused over the control, help appears with the control it is called as tool tip text. Here you can set the *name*, or purpose or any help related to that control.

Inspite of all these properties every control has some unique properties. You need to understand these properties to understand the use of that control. Some of them are given below:

1. **Text:** Textbox and combo box control has text property. This property allows you to input text at run time. This text may be string, number, special character or any other thing.
2. **Caption:** *Label* control has *caption* property. Text box do not have *caption* property. This is used to display text. You can set this property design time as well as at run time.
3. **Style:** Command button, option button, list box, combo box all these controls have style property. In case of command button it decides the *appearance* of the button either as standard or graphical. If it is set to graphical you can display image on the button. In case of list and combo box it decides the style of the list box. It is either a standard or list box or simple combo, dropdown combo or drop down list.

6.1 Studying the Events of a Form

Forms are the container which hold all the controls of your application. You add these to your VB application as they are needed. Form has many properties and different events and methods. They are discussed as follows:

1
Oct.2012 – 8M
What is an Event?
Explain various Events of
a Form.

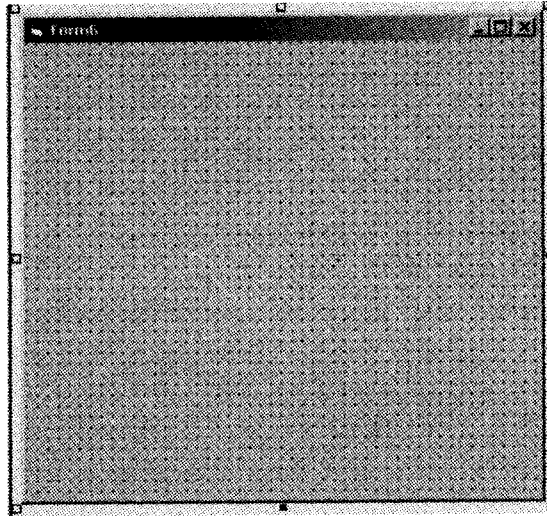


Figure 1.26

The Show and Hide Methods

The *Show* method of a form displays that form on the screen. If the form to be shown is not already loaded into memory, the Show method will load it before showing it. The Show method is typically used to transfer control from one form to another.

The syntax is

```
formname.Show
```

For example, if Form1 is open and want to display Form2, the code will be *Form2.Show*

To suspend execution of the first form until after the second form is done with, add the keyword constant *vbModal* as an argument to the Show method.

The syntax is

```
Form2.Show vbModal
```

The *Hide* method of a form removes the form from the screen i.e. makes it invisible, but the form still remains in memory.

The syntax is

```
formname.Hide
```

For example,

```
Form1.Hide
```

As an alternative, you can use the keyword *Me*. The keyword "Me" refers to the form in which code is currently running: **Me.Hide**

The Load and Unload Statements

The *Load* statement loads a form into memory, but does not display it. When you code the Load statement for a form, the *Form_Load* event of that form will be triggered.

The syntax is

```
Load formname
```

The *Unload* statement removes a form from memory and from the screen. When you code the Unload statement for a form, the *Form_Unload* event of that form will be triggered.

The syntax is

```
Unload formname
```

A form can unload itself, as in

Unload Me

The Unload event is also triggered when the user clicks the Windows, "close" ("X") button on the form.

You can also have your own close button, place a command button named "cmdClose" with the *caption* "Close" on a form. In the *Click* event for the command button, instead of coding "End", code:

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

When unloaded, a form is removed from the Forms collection. It is not actually destroyed, however, until all references to it are set to *Nothing*. Before this is done, it remains in the created but not loaded state. Once all references are set to *Nothing*, then the form is destroyed, the *Terminate* event fires, and the memory and resources the form used are released.

End statement: The *End* statement automatically unloads all forms in a project. Ending an application with "End" does not give the user a second chance to keep working with the program. The End statement ends the program quickly.

Form_Initialize: This event will occur as soon as the form is referenced in any way. Normally, any related objects are initialized in this event.

Form_Resize: This event resizes the form appropriately.

Form_Activate: Any code that must be run after the form has been loaded (such as a SetFocus call) can be put into the Activate event. The form will get additional Activate events whenever it becomes the active form in the application.

Form_GotFocus: This event will only occur if no other control on the form can get focus. Otherwise, the specific control (the first enabled control in the TabOrder) will get the focus.

Form_Paint: This event will occur whenever the form needs to be repainted.

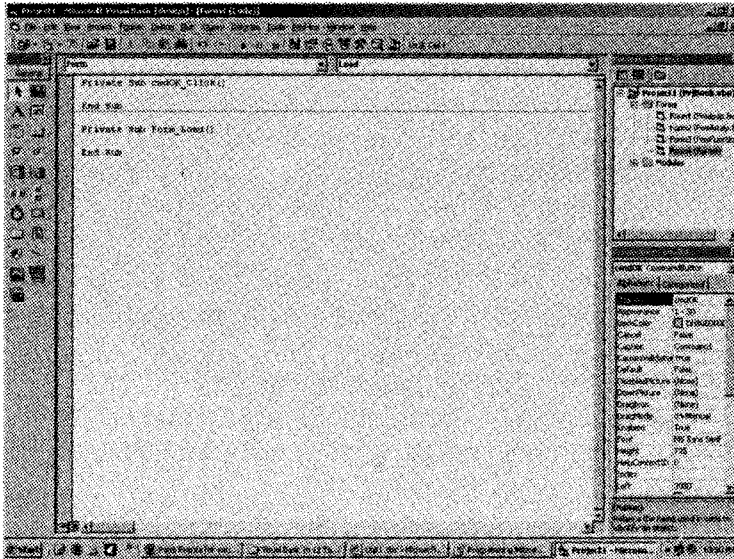
Form_Terminate: This event will occur when all references to the form are terminated by going out of scope or with the Set frmXXX = Nothing syntax. To ensure you are cleaning up memory after you unload the form, you should set the form reference to *Nothing*. If you don't need access to any methods or properties, you can add this line with unload event.

6.2 Working Code for Events

Code Window

Like its name implies, this is where you type in the code that VB executes. Notice that the heading of the window indicates with which event the code is associated. This is the window where you will write your code. It is a word processor with full of facilities where you can write your code easily. Instead of using notepad or any other editor to write VB code, VB provides its own code editor. This code editor includes facilities like color coding for keywords, automatic syntax checking. It gives auto list and auto quick info features. Auto list pop ups list of properties and methods when you type the code and auto quick info provides the information related to syntax.

When you double *click* any item on the form, the form itself opens the code window, and you will see the following window:



You can see some lines of code appearing in the window, VB automatically inserts that lines every time you open the code window. Let's try to understand the code VB has inserted for you: (Refer *Figure*)

```
Private Sub cmdOK_Click()
End sub
```

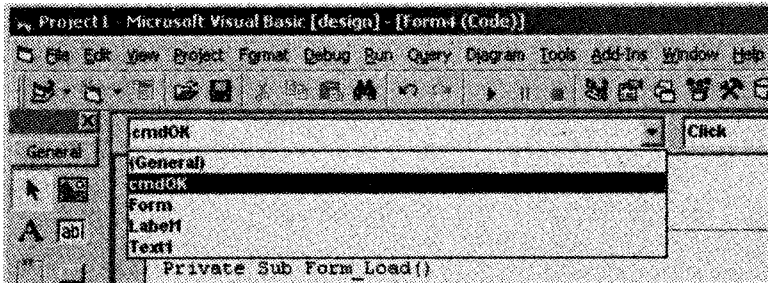
The words *Sub* and *End Sub* mean that this is a *Procedure*. Procedures are blocks of code that can be *executed* by Visual Basic. The word *Private* defines the scope of procedures. The most important part is *cmdOK_Click ()*. As we have discussed *name* property in the above section, “cmdOK” is the name of the button that you added to your form. The text after the *_* is known as the *event*. Any action you take can be an event in Visual basic. Like mouse paused over any object, *click*, double *click*, or when the user enters some text into a textbox, when check box is selected or deselected all of these trigger events.

In this case, the event is *Click*, and occurs, when cmdOK, command button, is clicked. Therefore, any code you enter into this procedure will be run when the OK button is clicked.

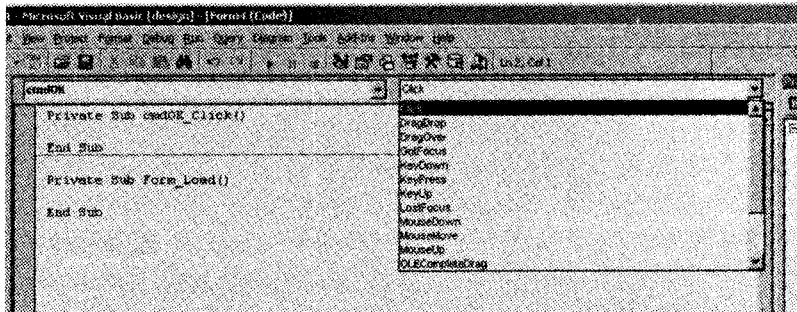
Now take a look at the two drop-down boxes at the top of the window:



The box on the left lists all the controls that are on your form. If you *click* it, you will see the names of the three controls that you added to the Form. The screen will look like *Figure*.



The right hand box lists all the events that can occur for this control. If you *click* it, you will see a long list of possible events for the button; the one that you are currently editing is displayed in bold (*Click*). If you select another item in the list, Visual Basic will automatically add a new procedure. As you can see in the following screen there are two procedures that are `cmdOK_Click` and `Form_Load`.



Some Common events

Events are what happen in and around your program. *For example*, when a user clicks a button, many events occur: The mouse button is pressed, the command button in your program is clicked, and then the mouse button is released. These three things correspond to the mouse down event, the *Click* event, and the mouse up event. During this process, the *GotFocus* event for the command button and the *LostFocus* event for whichever object previously held the focus also occur.

Not all controls have the same events, but some events are shared by many controls. These events occur as a result of some specific user action, such as moving the mouse, pressing a key on the keyboard, or clicking a text box.

1. **Move:** Changes an object's position in response to a code request.
2. **Drag:** Handles the execution of a drag-and-drop operation by the user.
3. **SetFocus:** Gives focus to the object specified in the method call.

4. **Using GotFocus and LostFocus:** The GotFocus and LostFocus events relate to most other events because they occur whenever a new control becomes active to receive user input. This makes GotFocus and LostFocus useful for data validation, the process of making sure that data is in the proper format for your program.
5. **Change:** The user modifies text in a combo box or text box.
6. **Click:** The user clicks the primary mouse button on an object.
7. **DbClick:** The user double-clicks the primary mouse button on an object.
8. **DragDrop:** The user drags an object to another location.
9. **DragOver:** The user drags an object over another control.
10. **GotFocus:** An object receives focus.
11. **KeyDown:** The user presses a keyboard key while an object has focus.
12. **KeyPress:** The user presses and releases a keyboard key while an object has focus.
13. **KeyUp:** The user releases a keyboard key while an object has focus.
14. **LostFocus:** An object loses focus.
15. **MouseDown:** The user presses any mouse button while the mouse pointer is over an object.
16. **MouseMove:** The user moves the mouse pointer over an object.
17. **MouseUp:** The user releases any mouse button while the mouse pointer is over an object.

6.3 Planning the Design

There are three important steps which you need to design any of your VB application that design the interface, set the properties and write the event.

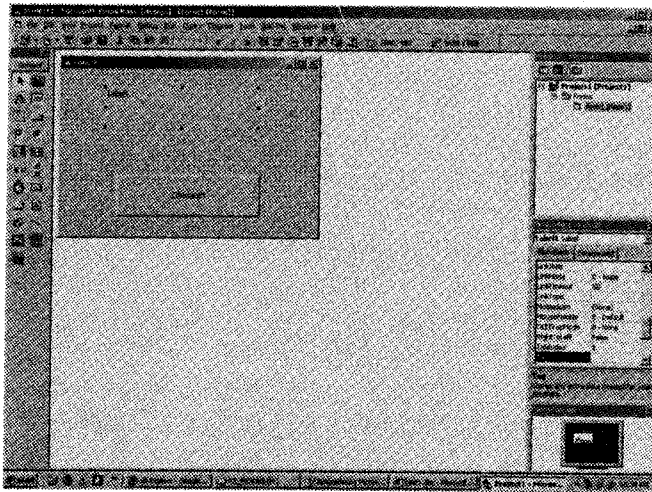
These steps are discussed in details below:

1. First step in designing your application is to plan what the user actually wants to see on the screen. For this pick up a pen and pencil and start designing your application on the paper. You can directly start working in VB IDE. But paper practice is always best to get a command on VB.
2. Plan what controls you require on the screen to perform your task. i.e. display text box, labels, menu and the options in menu, command buttons, option buttons, list control etc.
3. Change the properties of the control. Spend more time to understand properties. Properties are very important in designing any application in VB.

4. Next step is to arrange the controls in a proper way so that the user can easily understand the form and then change *label* of the controls. Give meaningful *caption* to the control so that when the user reads the form he/she can easily understand the meaning and use of the control. *For example*, save command button should have caption “Save” so that the user can understand the meaning of that control.
5. Identify the events for the control on your screen.
6. Write the code, procedure or functions for the events so that the control works.
7. Save the project alongwith all the forms and code written over there.
8. Run the application.

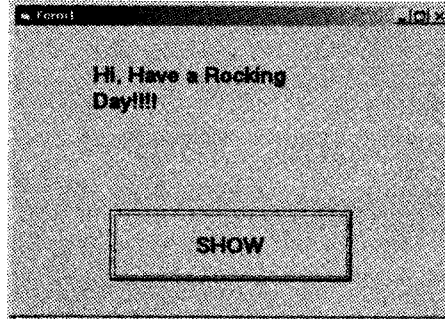
Now its time to do something. Let us start with first application

- Step 1:** Start VB. As VB comes up, it automatically creates a NEW application consisting of only one form which has no controls.
- Step 2:** Using the mouse, move the cursor over the toolbox and select a command button control. Now move the cursor to the form and while pressing the left mouse button draw out a rectangular shape with the mouse. Once the shape is drawn and the left mouse is released, a command button appears on the form. Same way select *Label* control and draw on the form, now your form should look like this:



- Step 3:** Change the properties of the command button. To change the properties you have properties window open on the screen or press F4 key to open the window. And then the property window will show the properties of the selected control.

Step 6: Your program is now complete and ready to run. Press F5 to run the program or *click* on the start button. Your application will appear as a single window in which there is a single button *labeled* 'show'. Press the button with mouse and you will see the word 'Hi, Have a Rocking Day!!!!' appear on the *label*. Now your form should look like this.



That's it! You've just successfully done your first VB application. To return to the IDE, *click* on the close button at the top right of the form.



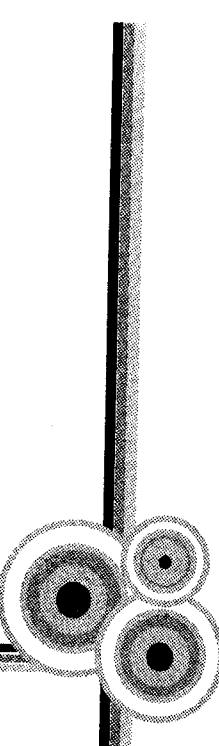
PU Questions

4 Marks

- [Apr.2013 – 4M] 1. Why visual basic is called as GUI Application?
- [Apr.2012 – 4M] 2. Write short notes: GUI (Graphical User Interface).
- [Oct.2011 – 4M] 3. Explain the structure of MDI.
- [Apr.2011 – 4M] 4. Write short note on: Object Oriented Programming .
- [Oct.2010 – 4M] 5. Write short note on: Even Driven Programming
- [Apr.2010 – 4M] 6. What is Event Driven Programming Language?

8 Marks

- [Apr.2013 – 8M] 1. What is Event Driven Programming in VB? Explain with example.
- [Oct.2012 – 8M] 2. What is an Event? Explain various Events of a Form.
- [Apr.2012 – 8M] 3. Discuss various Events related with Mouse and Keyboard.
- [Oct.2011 – 8M] 4. Write a short note on:
- Message box
 - Keyboard events
- [Oct.2010 – 8M] 5. Write short note:
- Keyboard Events.
 - Even Driven Programming



Chapter 2
**CONSTANTS,
VARIABLES,
OPERATORS, CONTROL
STRUCTURE, LOOPING
AND ARRAY**

1. Introduction

Designing form and displaying controls on the form and setting properties of the control is not sufficient to create an application or software. To develop any application we need to perform some arithmetic or logical operations, require some entity which will hold our data. To process any task we need to execute a sequence of instructions, this can be done using some programming tools like variables, data types, constants etc.

Like other programming language visual basic also has all these fundamental programming tools which is called as the grammar or syntax rules of the programming language.

In this chapter we will cover some fundamental features of Visual Basic like operators, data types, variables, constants, expressions etc. These fundamental things help us to perform some numerical operations, assigning data and manipulating strings in program. Visual basic supports rich set of built-in operators.

2. Data types

Visual Basic supports several data types. The default data type is variant.

3

Oct.10, Apr.12 – 4M

Describe data types in VB.

Apr. 2013 – 4M

Explain data types in VB.

The variant data type can store any type of data like numeric, date/time or string data. The data type specifies that what kind of data the variable will hold. The fundamental data types in Visual Basic are variant, integer, long, single, double, string, currency, byte and boolean. Each data type has the minimum and maximum range of values it can hold. In addition, some types can modify to extend their size. There are two categories of data types in VB6.0 that are built-in types and user defined data types. Both the types are discussed in detail in the following section.

1

Apr. 2011 – 4M

Explain any four built-in data types in VB.

2.1 Built in Data Types

Following table shows the list of all data types available in the Visual Basic

Sr. No.	Data Type	Description	Size in bytes	Value Range
1.	Byte	Store integer values	1	0 – 255
2.	Boolean	<p>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.</p> <p>The name is given from the name Boole, a famous mathematician.</p> <p><i>For example:</i> An option button property is FALSE if it is deselected and TRUE if it is selected.</p>	2	True or False
3.	String (fixed length)	Used to store alphanumeric values. A variable length string can store approximately 4 billion characters. Set of characters (fixed length) can be combination of alphabets and characters white spaces, special characters, and numbers.	length of string	1 to about 65400 characters.

	String String (variable)	<p>For example: name and caption property of any control is string. To identify variable of type string you can put dollar (\$) sign at the end of the variable name. For example: StrName\$</p> <p>This means the variable will only hold string. Strings are generally used to collect the data from text boxes.</p>	NA length +10 bytes	0 to 2 billion characters.
4.	Integer	<p>This data type can hold only whole or non decimal number. Percent (%) sign is used to make the variable compulsory to hold integers. For example: height/width properties of any control are of integer type. For example: IntNumber% = 40</p>	2	-32,768 to + 32,767
5.	Long	Store integer values.	4	-2,147,483,468- +2,147,483,467
6.	Single	<p>Store floating point values. For single the identifier we can use is exclamation sign (!) to make it compulsory to hold single number. For example: number! = 12.43</p>	4	-3.402823 E+38 to -1.401298 E-45 for -ve values and from 1.401298 E-45 to 3.402823 E38 for +ve values.
7.	Double	Store large floating value which exceeds the single data type value. The identifier used with double is pound (#)	8	-1.79769313486232E+308 through-4.94065645841247 E-324 for negative values. And from 4.94065645841247E-324 to 1.79769313486232E+308 for positive values.
8.	Currency	Stores monetary values. It supports 4 digits to the right of decimal point and 15 digits to the left	8	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
9.	Date	Used 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	8	Jan 1st 100 to December 31st 9999
10.	Variant (numeric) Variant (text)	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 variant.	16 bytes length +22 bytes	0 to 2 billion characters NA
11.	Object	Any type can be stored in a variable of type Object	4	NA
12.	Decimal	Not fully supported by VB.	12	NA

2.2 User Defined Data Types (UDT)

The built-in data types we have examined so far have been simple. They can also be used to form User Defined Types (UDT). In this section, we will discuss this concept in detail.

Like 'Structure' in C or C++ 'TYPE' is used to define UDT in VB. A User-Defined Type (UDT) is a compound data structure that encapsulates many variables of different data types. UDTs can be declared as Private or Public. It can only be private in form while in standard modules can be public or private.

Now we will see how to define the type:

1. You must first define its structure, using a *Type* directive in the declaration section of a module:

Syntax

```
Type [private/public]Type typename
Member1 As datatype1
Member2 As datatype1
:
:
End Type
```

For example,

```
Private Type Item
Srno As Integer
ItemName As String
Qty As Single
Price As Currency
End type
```

2. Once you have defined a Type structure, you can create variables of that type as you would do with any Visual Basic built-in type. This can be done with the *Dim* statement as in any other variable declaration statement.

Syntax

```
Dim Variable_name as Type_Name
```

For example,

```
Dim obl As Item
```

3. You can then access its individual member of type using the dot operator.

Syntax

```
Variable_name.Member_Name
```

For example,

```
Ob1.srno = 100
Ob1.ItemName = "Sugar"
Ob1.Qty = 4.5
Ob1.Price = 40
```

4. You can also define an array of these user-defined data types.

Syntax

```
Dim ArrayName(size) As Type_Name
```

For example,

```
Dim Item_Array(10) as Item
```

5. This User-Defined data type can be referenced in an application by using the variable name in the procedure along with the item name.

For example, if the Caption property of a Label namely Label1 is to be assigned the name of the Item, the statement can be written as given below.

```
Label1.Caption = ob1.ItemName
```

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

```
Label1.Caption = ob1(i).ItemName
```

6. User-defined data types can also be passed to procedures.

For example,

```
Sub ItemInfo (ob1 as Item)
    Label1.Caption = ob1.ItemName
    Label1.Caption = ob1.Price
End Sub
```

7. You can nest UDTs. This is helpful users to create more complicated type structure.

For example,

```
Public Type BirthDate
    MM As Integer
    DD As Integer
    YY As Integer
End Type
Public Type Student
    DOB as BirthDate
    Name As String
    Std As String
    Fees As Integer
End Type
```

8. When UDTs are nested, you use the same dot operator to access the member (VariableMember) but with an extra level, like this:

```
Dim StudInfo As Student
StudInfo.DOB.DD = 07
StudInfo.DOB.MM = 09
StudInfo.DOB.YY = 1976
```

9. UDTs can also be helpful when storing data on disk because VB's Random file type is specifically designed to work with UDTs.

3. Variables

Variables are the words which are used to store information during program execution. A variable has a name and a value. Variables must be declared before they are used in the program. To declare a variable use the dim statement followed by variables name. There are some rules to define variable name.

3.1 Rules to Define Variable

1. Variable name can be a combination of alphabets and numbers. But the name must begin with a letter.
2. Special characters like (@, #, \$, %, &, etc) are not allowed in variable name. except underscore (_).
3. A variable name cannot exceed 255 characters. But practically it is not possible to write long variable names. It is a good programming practice to write small and meaningful variable name.

For example,

```
Dim Name as String
Dim Rollno As Integer
Dim ProdName as String
```

4. Uppercase letters and lowercase letters are same for visual basic. VB does not differentiate between uppercase and lowercase.

For example,

```
Dim A as Integer
Dim a as Integer
```

Both the variable A and a are treated same in Visual Basic.

5. Reserve words are not allowed in variable names. There are number of reserve words (*For example:* Dim, If, Else, Case etc which represent commands, functions, and keywords. These reserve words have some predefined meaning in VB. User cannot override the meaning of these reserve words by using them in variable name.

Syntax

```
Dim Variable_Name As Data Type
```

For example,

```
Dim x As Integer
Dim y As Double
Dim name As String
```

3.2 Declaring Variable

Declaring variable is optional in Visual Basic. On the basis of that VB can declare variable in two ways i.e. implicit and explicit. But it is a good programming practice to declare variables explicitly to avoid ambiguity.

1. **Implicit declaration:** In VB it is not required to declare a variable. If VB meets an undeclared variable name it creates a new variable on the spot and uses it. The new variable's data type is variant, the generic data type that can accommodate all other data types. But, it is not convenient in programming.
2. **Explicit declaration:** Explicit declaration means declaring variable before they are used. Dim statement is used to declare the variable explicitly.

Syntax

```
Dim varname as datatype
```

In code window, write a statement *option explicit* (it is discussed in the following section 2.3) in general declaration. After inserting this statement if you try to run program without declaring variable it will give an error message. This avoids all the problems created by implicit declaration. Option explicit forces the user to declare the variable explicitly.

For example,

```
Option explicit  
Dim X as Integer
```

3.3 Using Option Explicit Statement

In Visual Basic explicit variable declaration is optional i.e. VB does not force variable declaration. It may be convenient to declare variables implicitly, but it can lead to errors that may not be recognized at run time.

For example, a variable named as “count” is used implicitly and is assigned to a value. In the next step, this variable is incremented by 2 by the following statement.

```
count = cont + 2
```

This calculation will result in count yielding a value of 2 as cont would have been initialized to zero. This is because the count variable has been typed as cont 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.

Let us see one more *example*. Suppose you are using variable X implicitly.

```
X = 100
```

Initially compiler will consider this X as an Integer variable. But later in the program if you change the value of the X and initialized it as string variable, as shown in the following statement.

```
X = "Visual Basic"
```

This will create an ambiguity and compiler will throw an error. Because previously it is considered as integer variable and then if there is any change in the data type it creates problem. Therefore, 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.

1. Select Options item in the Tools menu.
2. Select the Editor tab in the Options dialog box.
3. Check Require Variable Declaration option.
4. Finally click the OK button.

Or you can write "*Option Explicit*" statement in the general declaration section and then start writing code.

For example,

```
Option Explicit
Dim X As Integer
X = 10
```

'Because X is declared, this statement does not generate an error.

'The following assignment produces a COMPILER ERROR because, 'the variable is not declared and Option Explicit is on.

```
Y = 10 ' causes ERROR
```

3.4 Variable Scope

Scope defines the visibility of a variable. Variables can have scope ranging from global where any procedure in the application can access the variable to local to a single procedure. The scope of a variable determines where you can access that variable in your code. If a variable is in scope you can read or set its' value. If it is out of scope you will not be able to access it.

There are three types of scope for variables in Visual Basic

- 1. Global Scope:** Global variables are available anywhere in your program. Any line of code in any procedure can access the value of the variable. But Global variables are preferred when they are truly needed. It is not a good programming practice to make every variable global. To create a global variable, declare it in the declarations section of a standard module using the Global or Public keyword.
- 2. Module Scope:** Module level variables are available to any code within the module where they are declared. Module level variables allow you to share data between procedures without exposing that data to every procedure in the application. Therefore using module level variable is good programming practice. To create a module level variable, declare it in the declarations section of a module using either the Dim or Private keyword.
- 3. Local Scope:** Local variables have very restricted access. They are only available to the procedure in which they are created. They can be read or modified only in the same module. You create local variables by declaring them with the Dim or Static keyword within the body of a Sub, Function, or Property procedure.

Apr.2010 – 4M

1
What do you mean by Variable? Explain Scope of Variables.

3.5 Variable Duration

Duration defines the lifetime of a variable. Variables can exist for the life of an application or can be created and destroyed in a single procedure. Variable duration, or lifetime, indicates how long a variable exists in the life of a program. A variable may have a duration ranging from as long as the lifetime of an application or as short as the lifetime of a single procedure. Duration is closely related to variable scope because the location where a variable is declared can affect both.

Normally, all variables declared at the module level have duration of the lifetime of an application, and variables declared within a procedure only exist while the procedure is executing.

There are some exceptions to this rule

- 1. Static Variables:** Static variables, when declared in a procedure, keep their value even when that procedure is over. However, because they were declared in the procedure they are not available to other procedures and functions. You can declare local variables or even an entire procedure as static. Static variables retain their values between procedure calls. A common use of static variables is as control flags, for write-once property settings, and so on.

You can make variable static by using keyword *Static*.

For example,

```
Static x As Integer
```

The following code shows an event procedure for a Command Button (CmdClickMe) that counts and displays the number of clicks made.

```
Private Sub CmdClickMe_Click ( )  
Static Count As Integer  
Count = Count + 1  
Print Count  
End Sub
```

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

- 2. Class Modules:** Module level variables declared in class modules exist for the lifetime of the class objects. Remember that with classes you cannot directly access code or data within the module without first creating an instance of an object defined by the class.
- 3. Form Modules:** Form modules, like class modules, require that an instance of the form be created before its code and data can be used. However, VB treats form modules somewhat differently and will automatically create an instance of a form if any property of the form is referenced in your code. Module level variables in forms are not destroyed until the reference to the form object is released by setting the form to nothing. Unloading a form does not clear the value of the form's module level variables.

4. Constant

Sometimes in programs we need variables which do not change their values, this can be done by making constant variable. Visual Basic allows you to do so. The named constant feature in Visual Basic allows you to create value that will never change. Once a variable is made constant it means we cannot change its value throughout the program. Constants are similar to variables, except that you provide a value for the constant when you declare it, and its value can never change.

1

Oct. 2012 – 4M
Explain constants with an example.

For example, if you want to keep rate of interest same for the program you can make rate of interest constant, Or the value of PI is always same so PI can be a constant variable.

Keyword *Const* is used to declare constant.

Syntax

```
[Public | Global | Private] Const constant_name [As data type] = Value
```

Where,

- *Global* (project-level) constant can only be declared in a standard module and not in a form, using "Public Const" or "Global Const".
- Module-level and Local-Level constants can be declared in the General Declarations Section of either a standard or form code module using "Private Const" or only "Const" because by default it is "Private".
- If [*As Data type*] is omitted then the type of the variable depends on the *value*.

For example,

```
Const pi As Double = 3.14
Public Const RATE As Single = 0.06
Private Const JDATE As Date = #1/1/1980#
Const StrMSG = "Good Morning!!!" 'String data type assumed
```

Note: In Const declarations, string literals are delimited with double quotes ("), date literals are delimited with pound signs (#), and numeric literals are not delimited.

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 = vbMaximized
```

5. Operators

An operator is a symbol that tells the computer to perform certain mathematical, logical and relational operations. Operators are used in programs to manipulate data and variables. All the operators can be classified as binary or unary operators. Binary operators are the operators which require two operands to perform the operation and unary operators are the operators which take only one operand to perform the operation.

For example, the positive sign and the negative sign are each unary operators. They accept only one value when they do arithmetic operations. Generally we do not use + operator to show positive sign, by default the number is considered as positive, if there is no any negative sign with the operand. But to show negation we use - sign. *For example,* -a or -5 means negative 'a' or negative '5'. Remaining operators work as binary operators. There are different categories of operators in Visual Basic.

5.1 Arithmetical Operators

Arithmetic operators are used to perform basic arithmetic operations like addition, multiplication, division and exponentiation. These operators can be operated on numeric data types like integer, double. Arithmetic operators also involve the calculation of numeric values represented by literals, variables, other expressions, function, property calls, and constants. An arithmetic expression can be composed of a single numerical constant or variable or combination of variable constant and operators.

Following table explains all the arithmetic operators with the sample code

Operators	Description	Example	Result	Sample code of VB
+	Addition (Add two values in an expression together with the + Operator)	2+125	127	Dim x As Integer x = 2 + 125
-	Subtraction (subtract one from another with the - Operator) Negation also uses the - Operator (Visual Basic), but with only one operand,	100-50	50	Dim x As Integer x = 100 - 50 Dim x As Integer = 55 Dim y As Integer y = -x
/	Divide (Divide two numbers and return a decimal number)	26/5	5.2	Dim y As Double y = 26 / 5
\	Integer Division (Divide two numbers and return an integer)	26\5	5	Dim y As Integer y = 26 \ 5
*	Multiply	5*6	30	Dim y As Double y = 45 * 55.23
^	Exponent (power of)	3^3	27	Dim z As Double z = 23 ^ 3
Mod	Remainder of division	20 Mod 6	2	Dim x As Integer = 20 Dim y As Integer = 6 Dim z As Integer z = x Mod y
&	String concatenation (& is not really arithmetic but is considered to be in this category for precedence purposes.)	"This is "& "&"Fun"	"This is Fun"	Print "This is" & " " & "Fun"

5.2 Relational Operators

Relational operators are used to do comparisons between two items. It compares values to one another. Therefore relational operators are also called as comparison operators. All of the relational operators result in a *Boolean* value, it is either zero or one.

The list of relational operators is as follows:

Operators	Description	Example	Result
>	Greater than	45>10	True
<	Less than	40<18	False
>=	Greater than or equal to	200>=10	True
<=	Less than or equal to	100<=200	True
<>	Not Equal to	3<>8	True
=	Equal to	5=7	False

5.3 Logical Operators

Relational operators are not sufficient to make more complicated decisions. In addition to relational operators VB also supports logical operators like AND, OR, XOR, and NOT which are used to test more complicated conditions and make decisions. Logical operators compare *Boolean* expressions and return a *Boolean* result. The *And*, *Or*, and *Xor* operators are *binary* because they take two operands, while the *Not* operator is *unary* because it takes a single operand. Some of these operators can also perform bitwise logical operations on integral values.

The Not Operator (Visual Basic) performs logical *negation* on a *Boolean* expression. It yields the logical opposite of its operand. If the expression evaluates to *True*, then *Not* returns *False*; if the expression evaluates to *False*, then *Not* returns *True*.

The following table shows the list of logical operators in VB

Operators	Description
OR	It results TRUE if any one expression is TRUE. Otherwise it is FALSE.
AND	It results TRUE if both the expressions are true. If either of the expression is FALSE the AND operation results FALSE. And if both the expressions are FALSE in that case also AND results FALSE.
XOR	Results FALSE only if one of the expression is true and the other is false.
NOT	This operator is used to negate the value of the variable. i.e. it changes the TRUE value to FALSE and FALSE to TRUE.

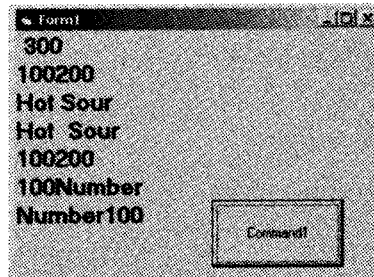
5.4 Differences between the Two Concatenation Operators

Concatenation operators join multiple strings into a single string. There are two concatenation operators, “+” and “&”. The “+” Operator is primarily used as an arithmetic operator that adds two numbers. However, it can also concatenate numeric operands with string operands. The + operator has a complex set of rules that determine whether to add, concatenate, signal a compiler error, or throw a run-time exception.

The “&” Operator is defined only for *String* operands, and it makes sure that both operands are string. Regardless of the setting of *Option String*, the “&” operator is exclusively defined for concatenating strings. Therefore it is recommended that to concatenate strings we should use & operator.

For example,

```
Dim x As Integer
Dim y As Integer
Private Sub Command1_Click()
    x = 100
    y = 200
    Print x + y
    Print x & y
    Print "Hot" + " Sour"
    Print "Hot " & " Sour"
    Print x & y
    Print x & "Number"
    Print "Number" & x
    'Print x + "Number" 'Conversion Error
    'Print "Number" + x 'Conversion Error
End Sub
```



In the above example you can see the different uses of “+” and “&” operators. If “+” is used with both numeric operands it adds both the values. When it is used with one numeric and one string operand then VB throws conversion error. If both the operands are string then + is overloaded by its

default meaning of addition and it performs the concatenation operation on both the strings. Whereas “&” is used to concatenate the strings. & operators concatenate the operands inspite the fact that, they are both strings; or either of them are string.

5.5 Precedence of the Operators

Imagine what will be the situation if we crowd all the operators together in the single expression?

For example: $\text{Ans} = 2 + 4 / 5 - 6 * 10 / 5$

If there are multiple operators in a single expression you need to know the order in which Visual Basic will evaluate them. Otherwise, you will get different results other than the expected one.

For example: $\text{ANS} = 6 + 5 * 8$

Could be interpreted as

$\text{Ans} = (6 + 5) * 8 \Rightarrow 88$

OR as

$\text{Ans} = 6 + (5 * 8) \Rightarrow 46$

Visual Basic calculates such expressions according to the priority given to the operators that is called *operator precedence*. When multiple operations occur in an expression, each operator is evaluated and resolved according to the operator precedence. All the operators have some predefined precedence in VB.

You can change the order of the operators by using parenthesis. In case of equal precedence, operations are performed left to right in the order in which they appear in the expression. Indeed, complex expressions should include parentheses to avoid any compiler misinterpretation and make the expression easier to read. You can use one or more pairs of parentheses in an expression to clarify the order of precedence. The parenthesis directs Visual Basic to evaluate the expression. The inner set of parentheses is evaluated first then performs the operation in the outer parentheses, and then it's outer and so on.

The order of the precedence from first to last is given in the following list

1. Parentheses ()
2. Exponentiation (^).
3. Negation (-).
4. Multiplication and division (*, /).
5. Integer division (\).
6. Modulo operator (Mod).
7. Addition/concatenation and subtraction (+, -).

8. String concatenation (&).
9. Arithmetic bit shift (<<, >>).
10. All the comparison operators (=, <>, <, <=, >, >=) have equal precedence and all have higher precedence than the logical and bitwise operators, but lower precedence than the arithmetic and concatenation operators.
11. Logical and bitwise negation (NOT). The logical and bitwise operators have lower precedence than the arithmetic, concatenation, and comparison operators.

6. Expression

An expression is a combination of operands and operators that represent a value. The operands indicate some values and operator is a symbol that indicates the action.

There are different types of expression in VB6. Like Boolean expression, logical expression, string expression etc.

Syntax

<Variable> = <Expression>

Where,

The assignment statement causes the value of the *expression* on the right side of the equal sign to be stored in the *variable* specified on the left side of the equal sign.

An *Expression* can be a constant, variable, or any valid combination of constants and/or variables.

Following *example* shows different forms of an expression in VB6

1. <Variable> = <constant>

i.e. constant is an expression

```
strSub = "Visual Basic"
```

here "Visual Basic" is constant and strSub is a string type variable.

2. <Variable> = <Variable>

i.e. variable on the right side of equal sign is an expression

```
Dim no1 As Integer
```

```
Dim no2 As Integer
```

```
no2 = 100
```

```
no1 = no2
```

Here value of no2 will be assigned to no1.

3. <Variable> = <arithmetic expression>

```
Dim no1 As Integer, no2 As Integer, no3 As Integer
```

```
no1 = 100.
```

```
no2 = 200
no3 = no1 + no2
no1 + no2 is an arithmetic expression.
```

Note: You cannot specify expression on the left side of the equal to sign i.e.
<Expression> = <expression>

is invalid

For example,

```
no1 + no2 = no3 + no4
```

is invalid, because, the role of the assignment statement is to assign value to the variable specified on the left of the equal sign therefore there can never be an expression on the left side of equal to sign.

7. Comments

Comments are the written remarks to document the program. It is the part of the program which is not considered at the time of execution. It is not shown in the output. Compiler ignores that line at the time of execution. Thus it helps the programmer to write some tips, explanation of some complicated logic of the code, to provide program heading, programmer details, to provide date, time or any other details related to that code, to mark separators within the program.

An apostrophe (') sign is used to start the comment in VB. The sign is followed with the text message. The comment can be inserted anywhere in the program. The comment ends with the end of the line. Therefore it is called as single line comment. There is no any special symbol used to end the comment.

For example,

```
'-----
'Program showing use of & and + operators.
Private Sub Command1_Click()
    x = 100
    y = 200
    Print x + y
    Print x & y
    Print "Hot" + " Sour"
    Print "Hot " & " Sour"
    Print x & y
    Print x & "Number"
    Print "Number" & x
    'Print x + "Number" 'Conversion Error
    'Print "Number" + x 'Conversion Error
End Sub
'End of the Program
'-----
```

8. Control Structures

In previous units we discussed fundamental concepts of Visual Basic. In Unit I we have seen that how different tools are used to design and develop any application. In Unit II we understood the grammar of the language i.e. how data types, variables and operators are necessary to write the program.

Now you must be familiar with the Visual Basic Integrated Development Environment. Remember that now you will need to write some logical expressions with the help of relational (<, >, <=, >=, !=, ==) and logical (&&, || and !) operators. Therefore it is necessary that you must have studied all those operators thoroughly. If not go to the Unit II and read it again. Otherwise if you have understood all those operators and tools very well then let's get ready to write some advanced but interesting application in the Visual Basic.

1**Apr.2010 – 4M**

Explain different control structures used in VB with examples.

Like other programming languages Visual Basic also supports different control structures. Control structures are used to control the flow of program's execution. Visual Basic supports different control structures such as decision making statement like if... Then, if...Then ...Else, Select...Case, and Loop structures such as Do While...Loop, While...Wend, For...Next etc. We will discuss all these control structure in this unit.

8.1 If

Some time in the program you need to perform action on the basis of some condition. This condition is generally a logical test. Such statements are referred as branching statement. The *IF-END IF* statement is used to perform such job in visual basic.

Syntax

```
IF <condition> THEN
    Statements
END IF
```

where,

- *IF – THEN* and *END IF* are the Visual Basic keywords. *IF-THEN* indicates the starting of the If block and *END IF* shows the end of IF block. Here in VB we don't use any brackets therefore these statements work like brackets.

- *<Condition>* is either simple or compound logical test. This is written using relational or logical operators. The result of this condition or logical expression is either TRUE or FALSE.
- "*Statements*" is a sequence of valid VB statements.

The *IF-THEN-END IF* statement works as follows

- The logical-expression is evaluated first. Its result is either *TRUE* or *FALSE*.
- If the result is *TRUE*, the *statements* after *IF-THEN* statement are executed.
- If the result is *FALSE*, there is no action. The controls are transferred to the statement following the *IF-THEN-END IF*.

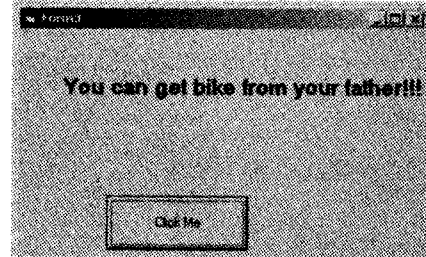
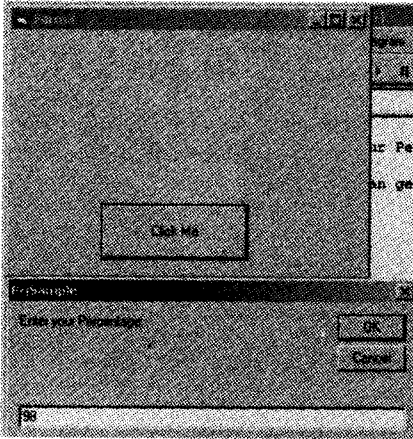
Example 1

Example to understand the use of IF-End If statement.

1. Add a new form in your project.
2. Display one Command Button and Label on the form.
3. Change the following properties of Label1
 - a. Name: lblMessage
 - b. Caption: <blank>
 - c. Autosize: True
 - d. Font: Bold, 14
4. Change the properties of the command button
 - a. Caption: Click Me
 - b. Height: 750
 - c. Width: 1935
5. Double click on the command button. This will open the code window.
6. Write the following code on the Command1_Click event

```
Private Sub Command1_Click ()
    Percentage = InputBox ("Enter your Percentage: ")
    If Percentage >= 90 Then
        lblMessage.Caption = "You can get bike from your father!!!"
    End If
End Sub
```

7. Run the Form and you will get the result as follows:



In the above *example* $Percentage \geq 90$ is the logical expression or condition. If the value of the *Percentage* variable is greater than or equal to 90 then only this condition will be true and then `lblMessage.Caption = "You can get bike from your father!!!"` this part will get executed. Otherwise no action will be taken and control will be shifted to the next statement after End if or if there is no any statement after end if there is end of the program.

There are different forms of IF-THEN-END IF statement. Because in the above format if block will be getting executed if the condition is true otherwise there is no action. This will not work in every situation. Therefore Visual Basic provides different forms of IF-END IF statements. Like If-Then-Else statement, If-Then-Else If and nested If statement. They are discussed in the sections that follow

8.2 Using If...Then...Else Statements

This may be the situation where you will take one action if condition is true or take other action if the condition is false. *For example*, you go to E-Square Theater, you will watch movie which you want or if movie has been changed you will go for shopping. This can be done with the help of If...Then...Else statement. If...Then...Else takes the following format:

Syntax

```
If <condition> Then
    'If condition is true this part Statements will get executed
Else
    'If condition is false this part Statements will get executed
End If
```

Where,

- *IF – THEN* and *END IF* statement are the same as before.
- *<Condition>* also same as before. It is either simple or compound logical test. This is written using relational or logical operators. The result of this condition or logical expression is either *TRUE* or *FALSE*.
- The new thing in this format is *ELSE* key word.
- Here in this format you will see two parts of block of code. One is before *ELSE* statement and second is after *ELSE*.

2

Oct. 11, 10 – 4M
 Explain IF-then-else statement in Visual Basic, with syntax and example.

The *IF-THEN-ELSE-END IF* statement works as follows

- The logical-expression is evaluated first. It results in either *TRUE* or *FALSE*
- If the result is *TRUE*, the *statements* after *IF-THEN* statement are executed upto *ELSE* statement.
- If the result is *FALSE*, the program skips the If-Then block and executes the *ELSE* block i.e. the statements after else statement.

Example 2,

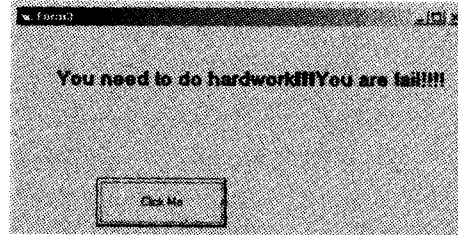
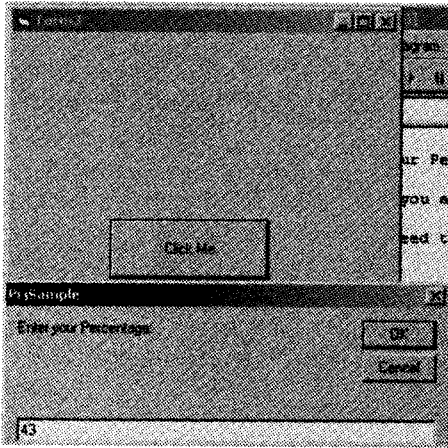
Example to understand the use of If-Then-Else

1. Add a new form in your project.
2. Display one Command Button and Label on the form.
3. Change the following properties of Label1
 - a. Name: lblMessage
 - b. Caption: <blank>
 - c. Autosize: True
4. Change the properties of the command button
 - a. Caption: Click Me
 - b. Height: 750
 - c. Width: 1935
5. Double click on the command button. This will open the code window.
6. Write the following code on the Command1_Click event

```
Private Sub Command1_Click ()
    Percentage = InputBox ("Enter your Percentage: ")
    If Percentage >= 50 Then
        lblMessage.Caption = "Good you are pass!!!"
    Else
        lblMessage.Caption = "You need to do hardwork.....You are fail!!!!"
    End If
```


End Sub

7. Run the Form and you will get the result as follows.



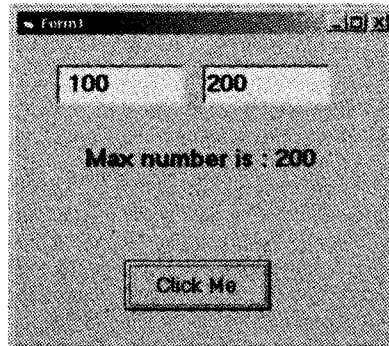
In the above example $Percentage \geq 50$ is the logical expression or *condition*. If the value of the *Percentage* variable is greater than or equal to 50 then only this condition will be true and then `lblMessage.Caption = "Good you are pass!!!"` this part will get executed. Otherwise the else part i.e. `LblMessage.Caption = "You need to do hardwork.....You are fail!!!!"` gets executed. If the condition is false the program will not check IF block. It will directly shift to else part and after executing else part control will hand over to the code after end if.

Example 3

1. Add a new form in your project.
2. Display two text boxes, one label and one command button on the form as you can see in the following output form. From that output form you will get an idea how to design that form and what properties should be set.
3. Change the following properties of Label1
 - a. Name: lblAnswer
 - b. Caption: <blank>
 - c. Autosize: True
 - d. Font Size: 14, Bold
4. Change the properties of the command button:
 - a. Caption: Click Me
 - b. Name: CmdClickMe
 - c. Height: as per your requirement
 - d. Width: as per your requirement
5. Change the properties of both text boxes:

- a. Name: txtno1 and txtno2
 - b. Font Size: 14 Bold
6. Double click on the command button. This will open the code window.
 7. Write the following code on the CmdClickMe_Click event

```
Dim no1 As Integer
Dim no2 As Integer
Dim ans As Integer
Private Sub CmdClickMe_Click()
    no1 = Val(txtno1.Text)
    no2 = Val(txtno2.Text)
    If no1 > no2 Then
        lblAnswer.Caption = "Max number is: " & no1
    Else
        lblAnswer.Caption = "Max number is: " & no2
    End If
End Sub
```



In this *example* you can read values in both text boxes i.e. txtno1 and txtno2. By default the value in text boxes are text. You need to convert it into number by using VB function VAL(). This function converts text to number. But that text should be only digits and not any character. Next if condition checks the number, if condition is true no1 is big else no2 is bigger.

You can skip the code block after If-then statement and directly write Else block. Because of this you can directly execute the else block i.e. only the False portion of the statement, you can place code statements after the Else statement; you aren't required to place any statements between the If and Else statements, as shown in the following line of code:

```
Dim CH as Integer
If CH <= 1 then
Else
    MsgBox "You are here in Else."
End If
```

8.3 Using Nested If Statements

You can nest if statement. i.e. if statement within another if statement. If you want to execute set of statement depending upon the test for a condition that depends on whether another condition is already True, use nested If statements. You can put as many if within another if as you wish. There is no limit on the number of if – else statement to nest. The format for a nested if statement is as follows

```
If condition Then
    If another_condition Then
        Statement
    Else
        Another statement
    End If
End If
```

OR

```
If condition1 Then
    If condition 2 Then
        Statement
    Else
        If Condition3 then
            Another statement
        Else
            Statements
        End if
    End If
End If
```

This nested form of if statement tests multiple conditions by placing another if statement within if statement. The problem with such nesting is that you need to write as many end if as there are ifs in the code. If you miss any single end if, the code will throw an error. The solution for this is instead of writing separate if after else statement you can connect that if with else. That is called ladder if. If you use ladder you don't need to write end if for that Elseif statement. The format for ladder is as follows:

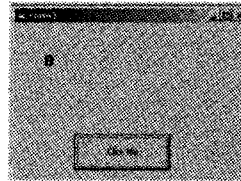
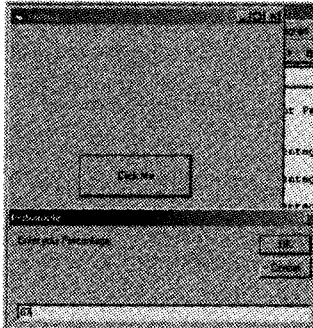
```
If condition1 Then
    If condition2 Then
        Statement
    ElseIf Condition3 then
        Another statement
    ElseIf
        Statements
    End if
End If
```

Example 4

```
Private Sub Command1_Click()  
    Percentage = InputBox("Enter your Percentage: ")  
    If Percentage > 80 Then  
        Grade = "A"  
    ElseIf Percentage < 80 And Percentage > 59 Then  
        Grade = "B"  
    ElseIf Percentage < 60 And Percentage > 49 Then  
        Grade = "C"  
    ElseIf Percentage < 50 And Percentage > 39 Then  
        Grade = "D"  
    Else  
        Grade = "E"  
    End If  
    lblMessage.Caption = Grade  
End Sub
```

This is the best *example* which shows the use of nested if. In the above example we are calculating Grade. There are various Grades given on the basis of percentage.

- You can try this *example*.
- Repeat the same procedure as we have done in *example 2*.
- Here in this example you just need to change the code.
- In the above example (*example 2*) we are checking only 2 situations i.e. true or false. But in this example we are checking multiple situations by using multiple if statements.



8.4 Select Case

In the previous section, we have learned how to use *If... ElseIf* control structure to control the program flow. Now we will learn another way to control the program flow, that is, the *Select Case* control structure. In some situations we need to select one option among multiple. In such situations

we can use if-else statement. But it may increase the complexity of the program and the programmer may get confused. Because programmer needs to write many nested if statement which will be complicated to read and modify the code. In such situation it is better to use multiway decision making statement i.e. select – case statement. Select...case is more convenient to use than the If...Else...End If.

The format of the select case control structure is shown below:

```
Select Case expression
  Case value1
    VB statements1
  Case value2
    VB Statements2
  Case valueN
    VB Statements
:
  Case Else
    VB Statements
End Select
```

Where,

- The *Select Case* is the VB keywords. The *Expression* after the select case statement is called select condition. When the program starts executing the value of expression is compared with the *case value* statement one by one. It compares with first case statements and then moves to the next till the match is found. If the match is found the program executes the code written in *VB statement*. The value of expression can be string or number. But if the expression is string the value of case statement must be string or if it is number the value of case statements is also number i.e. the data type of expression and case value must be same. You can write as many case statements as you wish. There is no limit on the number of case statements.
- *VB Statement* is the valid VB statements. If any case matches with the value of select expression, program executes VB statements of that particular Case Value.
- You can write multiple values with case. All the values should be separated with commas. *For example, Case 10,12,34,60.*
- You can also give the range of values. While giving range the start and end value must be connected with TO keyword. *For example, case 10 To 100.*
- You can also write logical expression with Case. Use *IS* keyword with case and then specify the logical expression. *For example, Case IS <> 100.*
- The *case else* part is optional. If match could not be found among available case values then case else is executed.
- Select case is easy to use for menu entries.

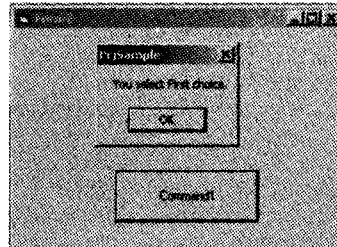
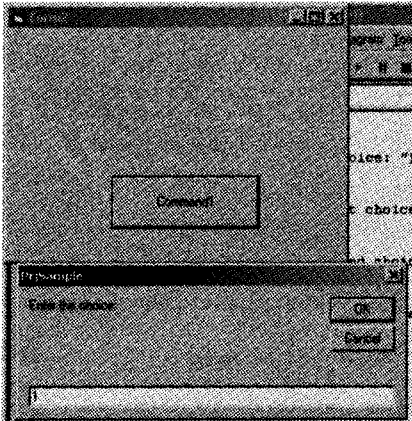
Examples,

1. In this example simple select case statement is used.

```
Private Sub Command1_Click ()
    Dim choice As String
    Choice = InputBox ("Enter the choice :")
    Select Case choice
    Case 1
        MsgBox "You select First choice."
    Case 2
        MsgBox "You select Second choice."
    Case 3
        MsgBox "You select Third choice."
    End Select
End Sub
```

Oct. 2012 - 4M
Explain SELECT Case in
VB with example.

1

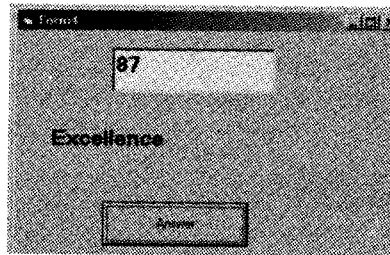


2. If you are using string value with case and expression then the value should be enclosed within the double quotes, as shown in the following *example*.

```
Dim grade As String
Private Sub Compute_Click()
    Grade = txtgrade.Text
    Select Case Grade
    Case "A"
        lblResult.Caption = "Congratulations!! You have  
Distinction."
    Case "B"
        lblResult.Caption = "Good!! You have Firs class."
    Case "C"
        lblResult.Caption = "Oops!!! Only Pass Class. Hard Work"
    Case Else
        lblResult.Caption="Very Bad!!! Fail."
    End Select
End Sub
```

3. Using Range of the values with case.

```
Dim mark As Integer
Private Sub Command1_Click ()
    Mark = txtmarks.Text
    Select Case mark
        Case 0 To 49
            lblResult.Caption = "Need lot to work harder"
        Case 50 To 59
            lblResult.Caption = "Improve!!!"
        Case 60 To 69
            lblResult.Caption = "Near to Good!! Try some more."
        Case 70 To 84
            lblResult.Caption = "It's Good"
        Case Is >= 85
            lblResult.Caption = "Excellence"
        Case Else
            lblResult.Caption = "You have entered some wrong Value."
    End Select
End Sub
```



9. Looping

Sometimes in programs we need to execute some code a number of times. In such cases one option is; write the code that many times. But is it the right solution? If you want to repeat any step 1000 times then what? VB gives solution for this. We have loop statements in VB which helps us to execute the block of code repetitively.

Loops tells the computer when and how often to perform the set of instructions. By using loops, you can simplify the complicated code. Loops can be used to check or process arrays, change properties of a program's controls, and execute or skip a set of tasks until a certain condition exists or is met. Visual basic supports different forms of loop statements.

While dealing with loops three things are very important:

1. **Initialization of the loop:** The loop must be initialized properly. It should be done before loop starts.
2. **Test or condition:** The condition is tested before or after the loop. If condition is at the time of the entry of the loop such loops are called as entry controlled loop because it checks condition at the time of the entry of the loop. Such loops are executed only if condition is true. Otherwise they never get executed. Where as if condition is at the end of the loop such loops are called as exit controlled loop. Because the condition is tested while exiting the loop. Such loops are executed at least once. Because program first enters the loop and then checks the condition. On the basis of the place of this condition there are four different forms of do loops. They are discussed in next section.
3. **Increment or decrement of the loop counter:** If this is not set properly you may go into infinite loop and will not be able to come out of the loop. This is done inside the loop. This is necessary to reach upto the condition of the loop and terminate the loop.

9.1 Do loop

On the basis of the place of the condition the Do loop has several different formats, as follows:

1. **Do while condition**
 'VB statements
Loop
2. **Do**
 'VB statements
Loop While condition
3. **Do Until condition**
 'VB statements
Loop
4. **Do**
 'VB statements
Loop Until condition

1

Apr. 2011 – 4M

Explain DO WHILE ...
LOOP Statement in brief.**1. Do while condition**

'VB statements

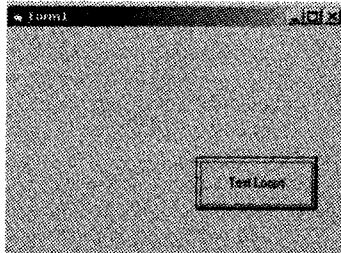
Loop

- This is the most commonly used format of do loop.
- Do, Loop and While are VB keywords.
- Condition is the logical expression (Just like if statement) which is used to check the iterations of the loop. It continues to loop as long as the condition is true.
- In this format, the condition is tested first; if condition is TRUE, then the VB statements are executed. When it reaches to the Loop it goes back to the Do and tests condition again. When the condition goes False it comes out of the loop. If the condition is false at the first pass the program never enters the loop.
- The Do indicates the beginning of the loop where as Loop indicates the end of loop.

Examples,

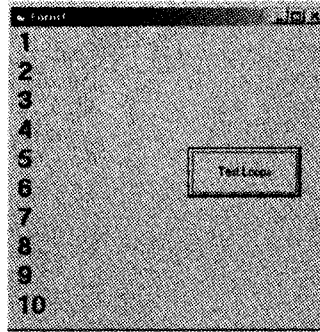
- i. This program will not print anything because the condition is FALSE and it does not enter in the loop.

```
Dim i As Integer
Private Sub Command1_Click ()
    i = 20
    Do While i <= 10 'Value of I is greater than 10
    Print i
        i = i + 1
    Loop
End Sub
```



- ii. This program will print 1 to 10 numbers because the loop executes till the condition is TRUE.

```
Dim i As Integer
Private Sub Command1_Click()
    i = 1
    Do While i <= 10
        Print i
        i = i + 1
    Loop
End Sub
```



2. Do

‘VB statements

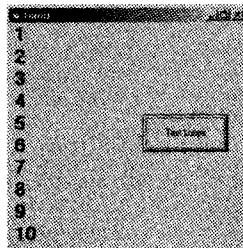
Loop While condition

- *Do*, *Loop* and *While* are VB keywords. The *Do* indicates the beginning of the loop where as *Loop* indicates the end of loop.
- Condition is the logical expression which is used to check the iterations of the loop. In this case it continues to loop as long as the condition is true.
- In this format, the *condition* is tested at the end of the loop; therefore the program first enters the loop and after first pass the condition is tested. If condition is *TRUE*, again it goes back to *Do* statement and executes the code. When it reaches the *Loop* again it checks the condition. The step is repeated till the condition is true. When the condition goes *False* it comes out of the loop. However the condition is false still at least one pass through the loop is carried out.

Example,

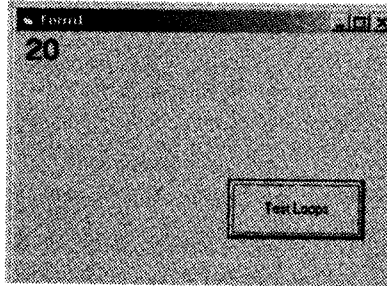
- This program will print 1 to 10 numbers.

```
Dim i As Integer
Private Sub Command1_Click ()
i = 1
Do
    Print i
    i = i + 1
Loop While i <= 10
End Sub
```



- ii. In this code the condition is false. Still the program executes loop once. Because the condition is tested at the end of the loop.

```
Dim i as Integer
Private Sub Command1_Click ()
i = 20
Do
    Print i
    i = i + 1
Loop While i <= 10
End Sub
```



3. Do Until condition

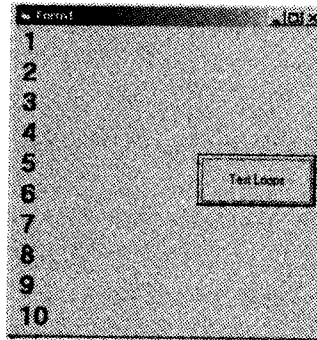
'VB statements

Loop

- *Do*, *Loop* and *until* are the VB keywords.
- Condition is the logical expression which is used to check the iterations of the loop. It continues to loop as long as the condition is not true.
- In this format, the *condition* is tested first; if condition is *FALSE*, then the VB statements are executed. When it reaches to the *Loop* it goes back to the *Do* and tests condition again. When the condition goes *true* it comes out of the loop.
- The *Do* indicates the beginning of the loop where as *Loop* indicates the end of loop.
- The Do...Until loop is basically the same as the Do...While loop, except that the Do...Until loop runs as long as the condition is false. When the condition becomes true, the loop terminates.

Examples,

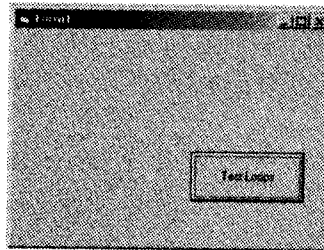
```
i. Dim i As Integer
Private Sub Command1_Click ()
i = 1
Do Until i > 10
    Print i
    i = i + 1
Loop
End Sub
```



```

ii. Dim i As Integer
     Private Sub Command1_Click()
         i = 100
         Do Until i > 10
             Print i
             i = i + 1
         Loop
     End Sub

```



4. Do

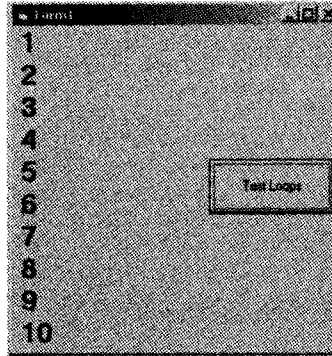
'VB statements

Loop Until condition

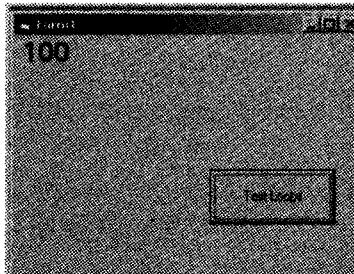
- *Do*, *Loop* and *until* are the VB keywords. The *Do* indicates the beginning of the loop whereas *Loop* indicates the end of loop.
- Condition is the logical expression which is used to check the iterations of the loop. It continues to loop as long as the condition is not true.
- In this format, the *condition* is tested at the end of the loop; therefore the program first enters the loop and after first pass the condition is tested. If condition is *not TRUE*, again it goes back to *Do* statement and executes the code. When it reaches to the *Loop* again it checks the condition. The step is repeated till the condition goes false. When the condition goes *true* it comes out of the loop. Like *Do-Loop-While* this loop also executes at least once.

Examples,

- i. Dim i As Integer
 Private Sub Command1_Click ()
 i = 1
 Do
 Print i
 i = i + 1
 Loop Until i > 10
 End Sub



- ii. Dim i As Integer
 Private Sub Command1_Click ()
 i = 100
 Do
 Print i
 i = i + 1
 Loop Until i > 10
 End Sub



Following table shows the comparison among all the 4 types of Do – Loop

Do-While Loop	Do-Loop While	Do-Until Loop	Do-Loop Until
Do While condition 'VB statements Loop	Do 'VB statements Loop While condition	Do Until condition 'VB statements Loop	Do 'VB statements Loop Until condition
This loop continues executing the body of the loop as long as the relational test is True.		This loop continues executing the body of the loop as long as the relational test is False	
The placement of condition is at the beginning. The condition is tested first.	The placement of condition is at the End. Therefore the loop is executed at least once.	The placement of condition is at the beginning. The condition is tested first.	The placement of condition is at the End. Therefore the loop is executed at least once.
Do While no <20 No = no + 1 Loop	Do No = no + 1 Loop While no <20	Do Until no <20 No = no + 1 Loop	Do No = no + 1 Loop Until no <20

5. Exit Do statement

Sometimes we need to exit a loop prematurely because a certain condition is satisfied. The solution for this is Exit Do statement. Whenever your program reaches an Exit Do statement within a loop, it will exit the current loop. The Exit Do command can be used with all types of Do loops.

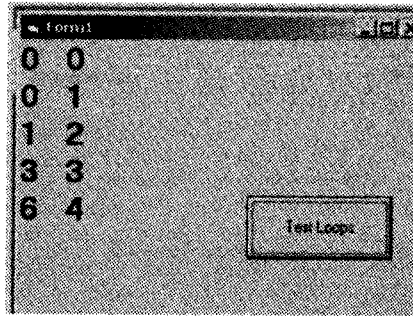
Syntax

Exit Do.

Example

This program shows the use of Exit Do command. In this program the loop should terminate if the value of i becomes greater than or equal to 10.

```
Dim i As Integer
Dim j As Integer
Private Sub Command1_Click ()
i = 0
j = 0
Do Until j > 10
    Print i & " " & j
    i = i + j
    If i >= 10 Then
        Exit Do
    End If
    j = j + 1
Loop
End Sub
```



9.2 For - Next

For loop is used when the number of iterations of the loop is known. Some time it is better to use for-next loop instead of using do-loop.

3

Oct.10,12 - 4M

Apr.12 - 8M

Explain any two looping structures used in VB with syntax and examples.

The format of for loop is bit complicated than that of DO-LOOP statements but once you understand that there is no problem in using for loop.

The format of the For-next loop is

```
For counter = START To END [STEP increment by]
    VB Statements
Next [Counter]
```

Where,

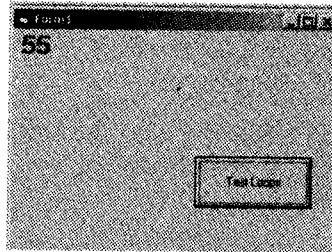
- For, to, Next, Step are VB keywords.
- The statement written in square brackets are optional.
- Counter, START, END, increment by are number variables.
- *VB statement* is block of code to be executed. Here you can write any valid VB statement.
- The *counter* is initialized by the value of *START*
- The loop is executed till the Counter reaches to the value of *END*.
- Every time Counter is checked to see if it is greater than *END*; if yes, it comes out from the loop and control passes to the statement after the Next; if not the VB statements are executed.
- At *Next*, by default counter is incremented by one. If you want to increase the counter by different number you need to specify the number with *STEP*.

- *[STEP increment by]* is used to increment the loop counter by the number different than the default value i.e.1. You can specify positive as well as negative value with the STEP. If you want to run the loop in decreasing order the value of the START should be large than END and STEP value is negative number. If START is smaller than the END then STEP must be positive number. This step is optional.

Examples,

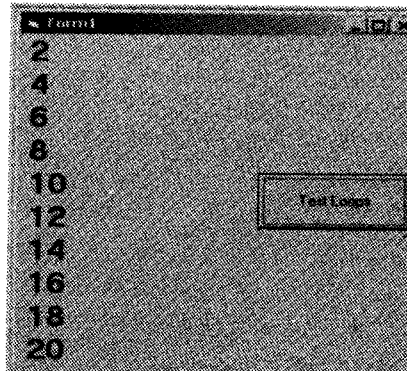
1. This program shows use of for loop. Here the code will print sum of 1 to 10 numbers.

```
Dim sum As Integer
Dim no As Integer
Private Sub Command1_Click ()
    Sum = 0
    For no = 1 To 10
        Sum = Sum + no
    Next
    Print Sum
End Sub
```



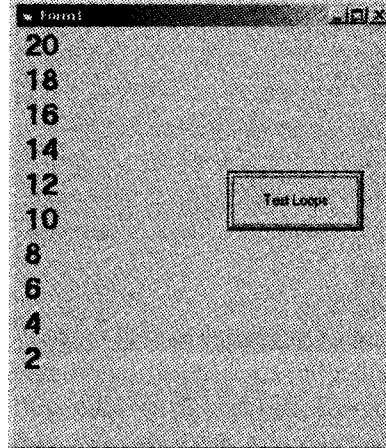
2. Program show use of for loop. Here the code will print all even numbers from 1 to 20.

```
Dim i As Integer
Dim j As Integer
Private Sub Command1_Click ()
    For i = 2 To 20 Step 2
        Print i
    Next
End Sub
```



3. For loop in decreasing order.

```
Dim i As Integer
Private Sub Command1_Click ()
    For i = 20 To 2 Step -2
        Print i
    Next
End Sub
```



Exit for Statement

Like Do Loops, you can exit from the for-loop before it terminates naturally.

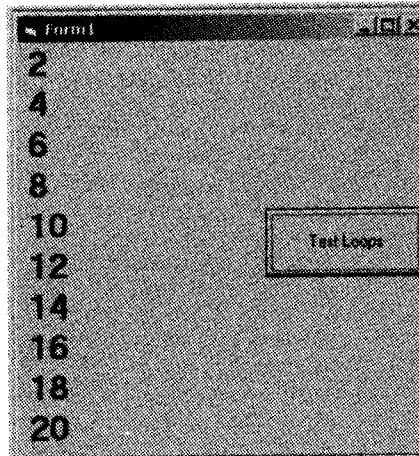
Syntax

```
Exit For
```

When this statement is encountered in the code the controls are exited from current for loop. You can write this statement anywhere within a For-Next loop.

Example,

```
1. Dim i As Integer
   Private Sub Command1_Click()
       For i = 2 To 200 Step 2
           Print i
           If i >= 20 Then
               Exit For
           End If
       Next
   End Sub
```



9.3 While – Wend Loop

A *While...Wend* is similar to the *Do While...Loop* statement. It executes a block of statements while a condition is TRUE.

The format of the While – wend is

```
While Condition
  VB Statements
Wend
```

Where,

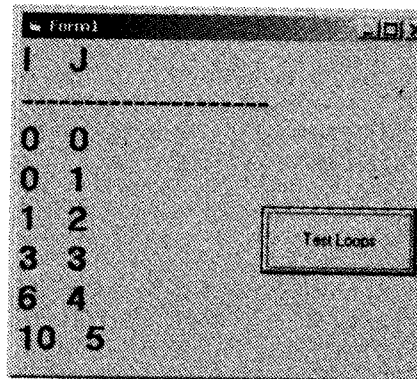
- *While* and *wend* are the VB keywords.
- *While* indicates the beginning of the loop where as *Wend* is the end of the loop. *Wend* stands for End of *While*.(W-End)
- Condition is the logical expression.
- VB Statement is block of code to execute.
- If the condition is true the VB statement is executed. If it is false the controls are shifted to the statement after *wend*.

In the following example *while...Wend* counts from 1 to 100

```
Dim n as Integer
n = 1
While n <=100
  n = n + 1
Wend
```

For example,

```
Dim i As Integer
Dim j As Integer
Private Sub Command1_Click()
i = 0
j = 0
Print "I" & " " & "J"
Print "-----"
While j <= 5
    Print i & " " & j
    i = i + j
    j = j + 1
Wend
End Sub
```



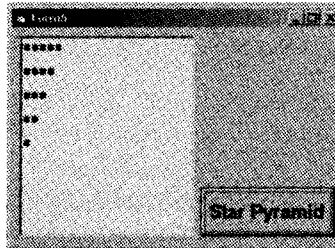
9.4 Nesting Control Structures

We have discussed various control structures in this unit. To make your program more easy and flexible you can place any control structure within any other control structure as many time as you need. This is called as nesting control structures. For example you can nest do – while loop with for loop, or you can have if statement within select case statement. You can also nest one loop within another loop. In case of nesting loops what happens is that the first pass of the outer loop starts the inner loop, which executes to completion. Again the second pass of the outer loop triggers the inner loop again. This repeats until the outer loop finishes.

Examples,

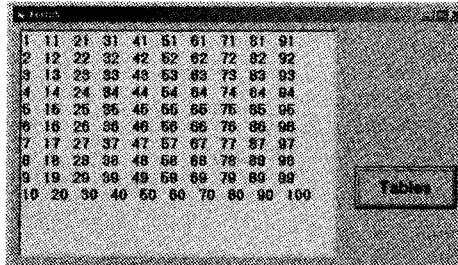
1. This Program shows nested loops. VBNewLine is VB's constant used to go on new line.

```
Option Explicit
Dim i As Integer
Dim j As Integer
Private Sub CmdStar_Click()
    For j = 5 To 1 Step -1
        For i = 1 To j
            txtstar.Text = txtstar & "*"
        Next i
        txtstar.Text = txtstar.Text & vbNewLine
    Next j
End Sub
```



2. Program to print number from 1 to 100 using nested loops.

```
Option Explicit
Dim i As Integer
Dim j As Integer
Private Sub Cmdtable_Click()
    For i = 1 To 10
        For j = 0 To 9
            txttable.Text = txttable & i + (10 * j) & " "
        Next j
        txttable.Text = txttable.Text & vbNewLine
    Next i
End Sub
```



10. Array

1

Oct. 2011 – 4M

What are Arrays in Visual Basic?

Many times we need to process multiple items at a time. An individual variable can hold only one item. In such cases if we require many items we have to declare that many variables; now imagine you have to manage with all these variables in program. How tedious it will be! You need to remember the names of all these variables. Such programs are not readable and not even easy to modify.

Therefore to overcome this problem we have arrays in VB. With arrays we can store multiple items of same characteristics under single name. Therefore we can define array as: "An Array is a set of elements having same data type."

That means all the elements in the array must have same data type. However you can have any type of elements. In other words we can say, arrays is nothing but, set of elements of same data type sharing single name. There is no restriction on the number of elements in the array. The size of the array can differ. One array might have 5 elements, another might have 100 elements, and it is also possible that an array do not have any elements at all.

An individual member of an array is called as subscript or an element. This is identified by specifying the arrayname followed by index number in parenthesis. This index number is an integer number which indicates the position of that element in an array. This index number determines the dimensions of the array. Generally this index starts from 0; therefore if we consider n number of array and if first subscript starts from 0 then the size of that array will be (n-1).

10.1 Declaring Array

Like variables, arrays are also declared before they are used. The Dim statement is used to declare the arrays. Dim statement specifies the dimensions of the array. The format to declare the array is as follows

```
Dim | Public | Private ArrayName(Size) As DataType
```

Where,

- *Dim*, *Public*, and *Private* are Visual Basic keywords that declare the array and its scope.
- If an array is *Dim*, the array is private to the procedure in which it is declared. If you use *Dim* within a module's procedure, the array will be available to only that procedure, but if you use *Dim* in the module's declarations section, the array will be available to all procedures within the module.

- If an array is *Public*, the array is visible from anywhere in the program,
- *Private* makes the array visible only to the form or module in which it is declared.
- *ArrayName* is the name of the Array. Like any variable, array also has some name.
- (*Size*) shows the number of element that array will hold. This size is always a positive integer number. This is also referred as index number. By default the first element of an array starts from zero. The number that you write in the parenthesis shows the upper limit of that array. It is the total capacity to hold the element.
- *Data type* is the type of the element in the array. All the elements are of same type. You can have all the elements of different types, such as one element of type integer, and other of type double and so on. Such arrays are called variant arrays. But using variant arrays is bad programming practice.

For example, Dim Deptname (10) As String

```
Public Deptname (10) As String
Private Deptname (10) As String
```

In the above example *Deptname* is the name of the array. Total number of elements stored in the array is *10*. *Deptname (0)* will be the first element; *Deptname (1)* is second, and, so on. The last element will be *Deptname (9)*. The scope of the array will be according to the prefix written with it i.e. *Dim, Public, Private*.

In VB it is not compulsory that array index should start from zero. You can specify the lower limit along with the upper limit.

The format of that array is as follows:

- *Dim City (1 to 10) as String*
- *Dim Books (10 to 100) as String*
- *Dim Price (11 to 20) as Double*

You can assign the element to the array by specifying the array name with index number in parenthesis. It is shown as follows: (consider the above *example*).

Dim Deptname (10) As String

```
Deptname (0) = "Computer"
Deptname (1) = "Sales"
Deptname (2) = "Purchase"
```

.....

```
Deptname (10) = "Finance"
```

If the array index starts from some other value except zero.

Dim Books (10 to 100) as String

```
Books (10) = "Visual Basic"
Books (11) = "Java"
Books (12) = "C"
```

.....

```
Books (100) = "C++"
```

You can use *option Base* statement to force the program to define the lower limit of the array. It works like *Option Explicit* statement. You need to specify the value with *Option Base*. This statement must appear at the form or module level. Once you declare *Option Base* statement it is applied to all arrays in the current form or module.

Option Base 1

For example,

Option Base 1

Dim X (100) As Integer

This is similar to

Dim X (1 To 100) As Integer

On the basis of dimension array can be divided into 2 types that are – single dimensional and two or multidimensional array.

10.2 Single Dimensional Arrays

1-D array looks like a simple list. It is either in the form of row or column. While declaring this array you need to specify only one dimension of the array. By default the array starts from the zero index number. But in VB we have the facility to change this index and you can start indexing from 1 or any number you wish.

For example, Dim BookName (8)

Where, BookName is the name of the array that holds the list of (7)8 book names which are called as elements of that array. BookName (0) will be the first element of that array.

BookName (0) = "Visual Basic"First element of the array

BookName (1) = "Data Structure"Second element of the array

.....
BookName (7) = "C programming"Third element of the array

Thus instead of having (7) 8 different variables for 8 (7) books this is easy to share one name with different index number.

The single dimensional arrays look like this

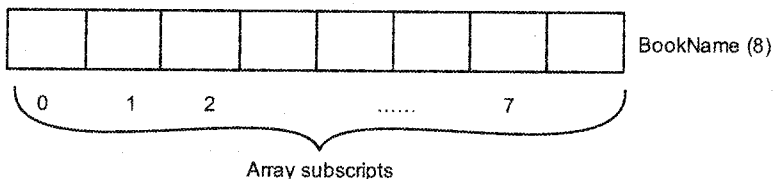


Figure 2.1

For example, this program shows the use of one dimensional array.

```
Option Explicit
Dim name(1 To 5)As String
Dim i As Integer
Private Sub Command1_Click()
    For i = 1 To 5
        name(i) = InputBox("Enter your name:" & i)
        Print name(i)
    Next
End Sub
```

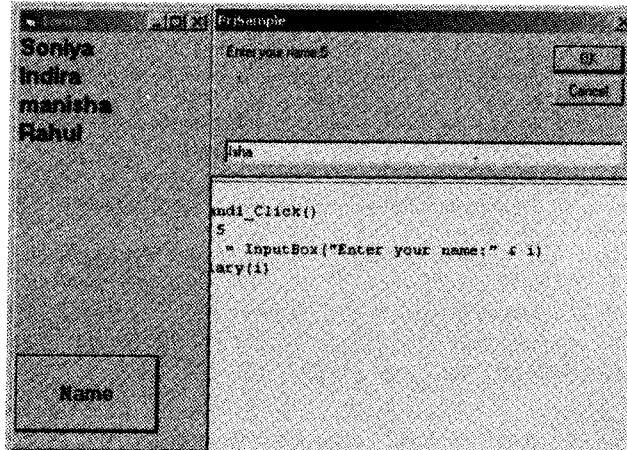


Figure 2.2

10.3 Multidimensional Array

An array can have multiple dimensions. The array having rows and columns are called as 2-D array. While declaring these arrays specify number of rows and columns along with array name. First subscript indicates the number of rows and second shows number of columns. 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 identifies the element's row and the second identifies the element's column.

Syntax

```
Dim ArrayName(Rows, Cols)As DataType
```

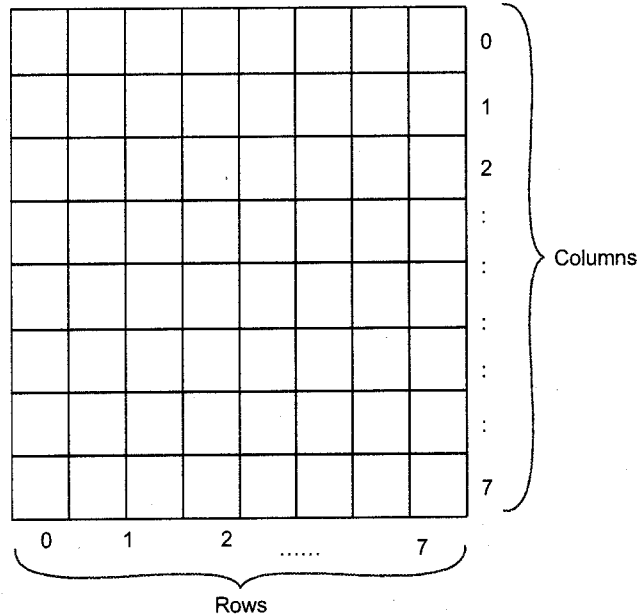



Figure 2.3

The following statement declares a two-dimensional array of 10 by 10:

For example, `Dim StudInfo (10, 10) As String`

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

Dim Result (100 To 200, 1 To 100)

You can have higher dimensional arrays i.e. the array having more than two dimensions. While declaring this array you need to specify one more additional subscript along with row and column. Because of this multidimensional array requires more space. Therefore we generally avoid use of such arrays. If all the spaces are not utilized there is unnecessary wastage of memory space. Visual Basic supports at least 60 array dimensions, but most people will need to use more than two or three dimensional-arrays.

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

`Dim Details (101 To 200, 1 To 100, 201 To 300)`

For example,

`Option Explicit`

`Dim salary (1 To 2, 1 To 2) As Integer`

`Dim i As Integer, j As Integer`

```
Private Sub Command1_Click()  
    For i = 1 To 2  
        For j = 1 To 2  
            Salary(i, j) = InputBox("Enter your salary:" & i)  
            Print salary (i, j)  
        Next j  
    Next i  
End Sub
```

10.4 Dynamic Array

Sometimes we cannot predict the size of the array in advance. For this we have dynamic arrays. Dynamic arrays are flexible and have capability of changing the size of the array at run time. The array we discussed in the above section is fixed or static array i.e. the arrays of which size is known to the compiler and compiler allocates memory space for that elements. But the problem with this is if all the locations are not used there is wastage of memory space. These problems can be overcome by using dynamic array.

'This is a static array.

```
Dim AuthorList (100) As String
```

Following are the steps to create dynamic array

1. Declare it like general array using Dim statement but do not specify the size. Keep the parenthesis empty.

Syntax

```
Dim | Public | Private ArrayName () As DataType
```

For example,

```
Dim MyArray () As Integer
```

2. Then you create the array when you actually need it, using a ReDim statement. ReDim statement is used to redimension the array. It can be written only in a procedure. ReDim is executable statement. It forces the program to carry out an action at run time.

Syntax

```
ReDim ArrayName (UpperLimit To LowerLimit) As DataType
```

Example,

```
ReDim MyArray(10) OR  
ReDim MyArray(1 to 10)
```

You can also have dynamic multidimensional arrays.

```
Dim MyArray() As Double
ReDim MyArray(1 TO 10, 1 TO 10)
```

Example

'An array defined in a module

```
Dim Student() As String
...
Sub Info()
'Now if you need you create the array.
ReDim Student (200) As String
End Sub
```

By default index starts from 0 unless an Option Base 1 statement is placed at the beginning of the module. But do not use Option Base statement here because it makes code reuse more difficult. If you want to explicitly use a lower index different from 0, use this syntax instead.

ReDim Student (1 To 200) As String

Dynamic arrays can be resized each time with a different number of items. When you re-create a dynamic array, its contents are reset to 0 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 Student (200) As String

10.5 Control Array

Similar to arrays of variables; you can group a set of controls together as an array. Such a grouping of the same types of control is called as *control array*. Creating control array is a flexible feature of VB. Here same type of control means group of text boxes, labels, option buttons, and command buttons etc. You can identify individual control in the control array by its index which is the subscript value. Just like other array has. The index is non negative value. Control arrays can only be one-dimensional. The size of the array extends as you add control to the array.

Not all the properties of the control array elements are same. It may vary i.e., some members can be visible and some not, they can be sized differently, they can have different fonts or colors, they can have different captions etc.

To refer to a member of a control array, the syntax is

```
ControlName(Index) [.Property]
```

3

Apr.2010 – 4M

Write short note on Control Array.

Oct.11, Apr.12 – 8M/4M

What are Control Arrays? Explain with the help of a suitable example.

For example, to refer to the caption property of the second element of an array of labels called Label1, you would use:

```
Label1 (1).Caption = "Visual Basic"
```

The benefit of using control array is under some situations they can share the common name type, event & procedures. This reduces the overload of code and makes the program dynamic. Adding controls with control arrays uses fewer resources than adding multiple control of same type at design time.

We can create control array only at design time, and before creating control array minimum one control must be there on the form.

There are different methods of creating a control array:

1. You create a control and then assign a numeric, non-negative value to its Index property; you have thus created a control array with just one element.

For example,

- Place 2/3 text boxes on the form.
- Set *Index* property of all the text boxes. Like for First Textbox Index = 0; for second Index = 1 and so on. This will create control array of text box.

2. You create two controls of the same class and assign them an identical Name property. Visual Basic shows a dialog box warning you that there's already a control with that name and asks whether you want to create a control array. Click on the Yes button.

For example,

- Place a command button on the form. Check its *Name* property. It is Command1.
- Now place second command button. Its Name Property is Command2. Try to change the Name of Command2 to Command1.
- VB will show you the dialog box asking you to create control array.
- Click on Yes Button this will create control array for these command buttons.

3. You select a control on the form, press Ctrl+C to copy it to the clipboard, and then press Ctrl+V to paste a new instance of the control, which has the same Name property as the original one. Visual Basic shows the warning mentioned in the previous bullet.

For example,

- Place one Command button on the form. Change its name property to CmdArray.
- Click on the command to select it.
- Put pointer on the command and press right-click and select *Copy* option.
- Not on the free space of the form again press right mouse button and select *Paste* button.

- Now VB will ask you to create control array. Select *Yes* button.
- This will create control array of command buttons with same properties of the first command button.

To build a sample application that uses a control array, perform the following steps:

1. Start a new VB project. Place an option button on the form on right hand side and set its properties as follows:
 - a. Name: optColor
 - b. Caption: Red
 - c. Font: size 14, Bold

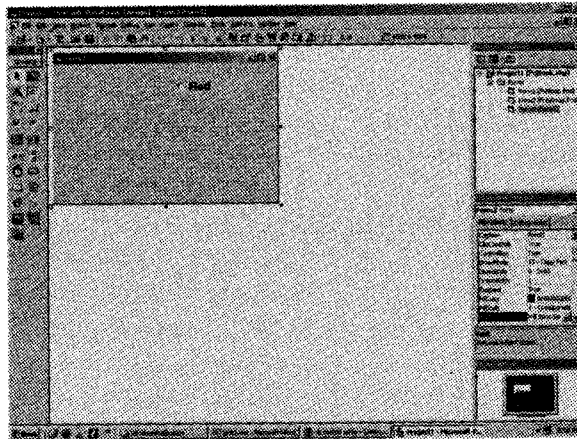


Figure 2.4

2. Click the option button once to select it. Then *Copy* it (Press Ctrl-C, or Edit >Copy, or right-click the mouse and choose Copy).

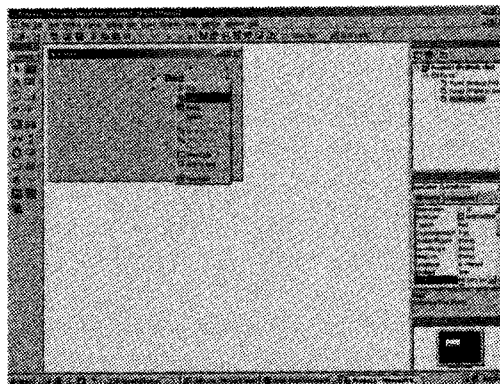


Figure 2.5

3. Click on an open area of the form and *Paste* (press Ctrl-V, or Edit >Paste, or right-click the mouse and choose Paste). The following message will appear: *You already have a control named 'optColor'. Do you want to create a control array?* Select 'Yes'. The pasted control will appear in the upper left-hand corner of the form. Move the pasted control toward the bottom, next the original. By answering yes to the prompt, VB automatically sets the *Index* property of the original option button to 0 and sets the Index of the pasted control to 1.
4. *Paste* two more times now VB knows that you want to create array so it will not prompt any message this time, moving the pasted controls next to the others. Set the Captions of optColor (1), (2), and (3) to 'Green', 'Blue', and 'Yellow' respectively. At this point your form should look like this

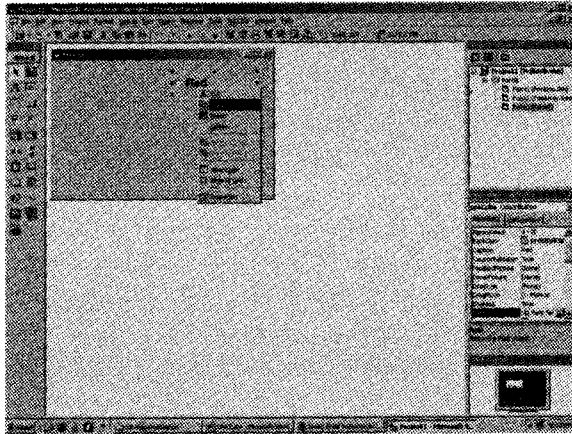


Figure 2.6

5. To make alignment of the control in proper format select all the option buttons and select Format → Align menu and select appropriate option. (To show all the option buttons in a group you may put all of them in a frame.)
6. Place one label control on left hand side of the form and set the properties:
 - a. Name: lblColor
 - b. Autosize: True
 - c. Font: Bold, 14
7. Place the following code in the optColor_Click event:

```
Private Sub optColor_Click (Index As Integer)
    If optColor (Index).Index = 0 Then
        lblColor.Caption = "Red"
    ElseIf optColor (Index).Index = 1 Then
        lblColor.Caption = "Green"
    ElseIf optColor (Index).Index = 2 Then
        lblColor.Caption = "Blue"
```

```
ElseIf optColor (Index).Index = 3 Then  
    lblColor.Caption = "Yellow"  
End If  
End Sub
```

8. Run the Form and click the various option buttons in any order. The run form will look like this

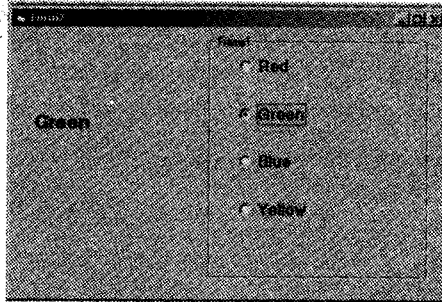


Figure 2.7

11. Functions (Built in and user defined)

Functions are the block of code which perform some predefined task. In this section we will discuss all about functions. VB supports many built-in functions as well as user defined (i.e. functions created by user) functions. Functions save the coding efforts. A function is similar to a normal 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.

The general format of a function is

```
FunctionName (arguments)
```

Where,

FunctionName is some name given to the function.

(Arguments) are values that are passed on to the function.

Visual Basic Built-in Functions

Visual Basic is rich in built-in functions. They 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 in detail

- Date and Time Functions
- String Functions
- Numeric functions

Date and Time functions

1. **Now:** The *Now* function returns a Date value that contains the current date and time.

Syntax

Now

For example, print now

2. **DateDiff:** The *DateDiff* function returns the intervals between two dates in terms of years, months or days. The syntax for this is given below.

Syntax

```
DateDiff (interval, date1, date2[, firstdayofweek  
[, firstweekofyear]])
```

3. **Date:** It returns the date of the form mm/dd/yyyy for the current date. You can reset the current date of the function.

Syntax

Date

For example, Print Date

4. **Time:** It returns the current time of the system.

Syntax

Time

For example, Print Time

String Functions

1. **Len:** Returns the length of the specified string.

Syntax

Len (string)

For example, L = Len("how are you?") 'L = 12

2

Apr. 2011 – 8M

Explain any four string functions.

Apr. 2010 – 4M

Explain any two built-in string functions with Syntax and examples.

- 2. Mid\$ (or Mid):** Returns a substring containing a specified number of characters from a string.
- Syntax*
Mid\$(string, start [, length])
- Where,
- String:* Required. String expression from which characters are returned.
- Start:* Required; Long. Character position in string at which the part to be taken begins. If start is greater than the number of characters in string, Mid returns a zero-length string ("").
- Length:* Optional; Number of characters to return. If omitted or if there are fewer than length characters in the text, all characters from the start position to the end of the string are returned.
- For example,* Ans = Mid\$("How are You?", 2, 5) ' Ans= "owar"
- 3. Left\$ (or Left):** Returns a substring containing a specified number of characters from the beginning (left side) of a string.
- Syntax*
Left\$(string, length)
- Where,
- String:* Required. String expression from which the leftmost characters are returned.
- Length:* Required; Numeric expression indicating how many characters to return. If 0, a zero-length string ("") is returned. If greater than or equal to the number of characters in string, the entire string is returned.
- For example,* strSubstr = Left\$("Visual Basic", 3) ' strSubstr = "Vis"
- 4. Right\$ (or Right):** Returns a substring containing a specified number of characters from the end (right side) of a string.
- Syntax*
Right\$(string, length)
- Where,
- String:* Required. String expression from which the rightmost characters are returned.
- Length:* Required; Numeric expression indicating how many characters to return. If 0, a zero-length string ("") is returned. If greater than or equal to the number of characters in string, the entire string is returned.
- For example,* strSubstr = Right\$("Visual Basic", 3) ' strSubstr = "sic"
- 5. UCase\$ (or UCase):** Converts all lowercase letters in a string to uppercase. Any existing uppercase letters and non-alpha characters remain unchanged.

Syntax

UCase\$(string)

For example, Ans= UCase\$("how are you?") 'Ans = "HOW ARE YOU?"

6. **LCase\$ (or LCase):** Converts all uppercase letters in a string to lowercase. Any existing lowercase letters and non-alpha characters remain unchanged.

Syntax

LCase\$(string)

For example, Ans = LCase\$("HOW ARE YOU?") 'Ans = "how are you?"

7. **Instr:** Returns a Long specifying the position of one string within another. The search starts either at the first character position or at the position specified by the *start* argument, and proceeds forward toward the end of the string (stopping when either *string2* is found or when the end of the *string1* is reached).

Syntax

InStr ([start,] string1, string2 [, compare])

Where,

Start: Optional. Numeric expression that sets the starting position for each search. If omitted, search begins at the *first* character position. The start argument is required if compare is specified.

string1: Required. String expression being searched.

string2: Required. String expression sought.

Compare: Optional; numeric. A value of 0 (the default) specifies a binary (case-sensitive) search. A value of 1 specifies a textual (case-insensitive) search.

8. **Space\$ (or Space):** Returns a string containing the specified number of blank spaces.

Syntax

Space\$(number)

For examples, Ans = Space\$(10) 'Ans = " "

9. **Replace\$ (or Replace):** Returns a string in which a specified substring has been replaced with another substring a specified number of times.

Syntax

Replace\$(expression, find, replacewith [,start[,count [, compare]])

Where,

Expression: Required. String expression containing substring to replace.

Find: Required. Substring being searched for.

Replacewith: Required. Replacement substring.

- Start:** Optional. Position within *expression* where substring search is to begin. If omitted, 1 is assumed.
- Count:** Optional. Number of substring substitutions to perform. If omitted, the default value is "1", which means make all possible substitutions.
- Compare:** Optional. Numeric value indicating the kind of comparison to use when evaluating substrings. (0 = case sensitive, 1 = case-insensitive)

For examples, Ans = Replace\$("11/07/2009", "/", "-")

'Ans = "11-07-2009"

10. **Trim\$ (or Trim):** Removes both leading and trailing blank spaces from a string.

Syntax

Trim\$(string)

For examples, strTest = Trim\$(" How Are You? ") 'strTest = "How Are You?"

11. **Asc:** Returns an Integer representing the ASCII character code corresponding to the first letter in a string.

Syntax

Asc(string)

For examples, Ans = Asc("*") 'intCode = 42

Ans = Asc("ABC") 'intCode = 65

12. **Split:** Returns a zero-based, one-dimensional array containing a specified number of substrings.

Syntax

Split (expression [, delimiter [, count [, compare]])

Where,

Expression: Required. String expression containing substrings and delimiters. If *expression* is a zero-length string (""), *Split* returns an empty array, that is, an array with no elements and no data.

Delimiter: Optional. String character used to identify substring limits. If omitted, the space character (" ") is assumed to be the delimiter. If *delimiter* is a zero-length string, a single-element array containing the entire *expression* string is returned.

Count: Optional. Number of substrings to be returned; 1 is default indicates that all substrings are returned.

Compare: Optional. Numeric value indicating the kind of comparison to use when evaluating substrings (0 = case sensitive, 1 = case-insensitive).

13. **Join:** Returns a string created by joining a number of substrings contained in an array.

Syntax

```
Join(list[, delimiter])
```

Where,

List: Required. One-dimensional array containing substrings to be joined

Delimiter: Optional. String character used to separate the substrings in the returned string. If omitted, the space character (" ") is used. If *delimiter* is a zero-length string (""), all items in the list are concatenated with no delimiters.

Numeric Function

1. **Round:** use to round off the number.

Syntax

```
Round(expression [, NumberOfDeceimalPlaces])
```

Where,

Expression: Required. It is the number which you want to round.

[, NumberOfDeceimalPlaces]: Optional. Allows to round the decimal points.

For example, Round(4.7) = 5

Round(4.732, 2) = 4.73

2. **ABS():** It gives the absolute value of whatever is inside the parenthesis. It removes minus sign if it is there.

Syntax

```
ABS(expression)
```

For example, abs(1) = 1

Abs(-234) = 234

3. **SQRT():** It returns the square root of the value given in the parenthesis. The number in the parenthesis is non negative number.

Syntax

```
SQRT(expression)
```

For example, SQRT(625) = 25

4. **EXP:** This function gives e (2.7182 approx) to the power x, where e is the base for natural logarithm and x is the value in the parenthesis.

5. **Int:** It is the function that converts a number into an integer by truncating its decimal part and the resulting integer is the largest integer that is smaller than the number.

For example, Int(3.5) = 3, Int(-5.6) = -6, Int(0.032) = 0

6. **Fix:** Truncates the decimal part of the number and return an integer. However, when the number is negative, it will return the smallest integer that is larger than the number.
For example, Fix (6.34)= 6 while Int(-6.34)=-7.
7. **Log:** is the function that returns the natural Logarithm of a number.
Log10= 2.302585
8. **Rnd:** It is very useful when we deal with the concept of chance and probability. The Rnd function returns a random value between 0 and 1.

Some Special Functions

1. **MsgBox ():** The objective of MsgBox is to produce a pop-up message box and prompt the user to click on a command button before he/she can continue. This format is as follows:

```
YourMsg=MsgBox (Prompt, Style Value, Title)
```

The first argument, *prompt*, is the actual message displayed in the out put message box. The *Style Value* will determine what type of command buttons appear on the message box, The *Title* argument will display the title of the message box

```
Private Sub CmdMsg_Click()  
    MsgBox "This is Message Box", vbOKCancel, "Note"  
End Sub
```

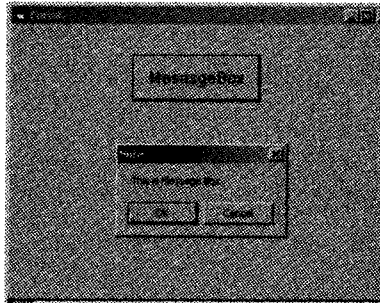


Figure 2.8

2. **InputBox ():** It Displays a prompt in a dialog box, waits for the user to input text or click a button, and returns a string containing the contents of the text box.

Syntax

```
Variable = InputBox (prompt [, title] [, default] [, xpos]  
[, ypos] [, helpfile, context])
```

Where,

Prompt: Required. String expression displayed as the message in the dialog box. The maximum length of *prompt* is approximately 1024 characters, depending on the width of the characters used.

Title: Optional. String expression displayed in the title bar of the dialog box. If you omit *title*, the application name is placed in the title bar.

Default: Optional. String expression displayed in the text box as the default response if no other input is provided. If you omit *default*, the text box is displayed empty.

xpos and *ypos*: Both optional. Numeric expressions that specify custom positioning of the box on screen (by default, the box is displayed in the center of the screen, which is usually desired).

helpfile and *context*: Both optional. Can be used if a help file has been set up for the application. If either one of these arguments are used, they both must be used.

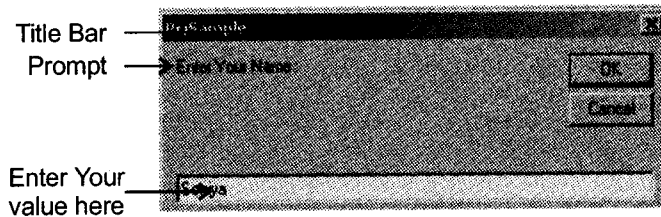


Figure 2.9

3. **Val()**: This function converts a numeric string to a number. This function is generally used to convert text box value to number. Because by default the text value is of type string.

Syntax

VAL (Expression)

For Example, Val (100)

X = Val (Text1.Text)

12. User Defined Functions

Like built in functions user can also define their own functions. In this section we will discuss about user defined functions.

We can define functions as: "Functions are block of code which have some predefined task and they return a value". That means that the function itself has a type, and the function will return a value to the calling subroutine based on the code that it contains.

Functions are similar to sub procedures. Except one difference that functions return some value. *Function* keyword is used to declare function.

Syntax

```
Function FunctionName (arguments) As DataType
```

```
.....
```

```
'Valid VB statements
```

```
FunctionName = ...
```

```
End Function
```

Where,

- *Function* is the VB keyword.
- *Arguments* are the parameter you can pass to the function. Multiple parameters can be passed using comma operators to the function.
- *DataType* refers to the value returned by the function. If the value is variant then *DataType* can be omitted.
- The function can be called like built in functions using function name.
- The function can be accessed using *Call* statement. The call statement is written as follows

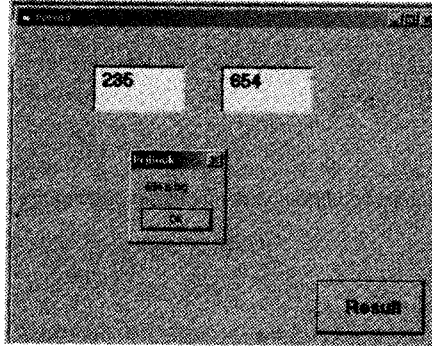
```
Call FunctionName(arguments)
```

For example, In this program we are passing two numbers to the function by value. The function compares both the numbers and print the big number. A function is declared the exact same way as a subroutine, except using the "Function" keyword instead of "Sub". When you pass arguments to the function remember that the number of parameters and its data type must be same.

This is function definition

```
Public Function MaxVal(ByVal A As Integer, ByVal B As Integer) As Integer
    Dim no As Integer
    If A > B Then
        MsgBox A & "is big", vbOKOnly
    ElseIf B > A Then
        MsgBox B & "is big", vbOKOnly
    Else
        MsgBox "Both are same", vbOKOnly
    End If
End Function
'Calling function
Dim x As Integer, y As Integer, z As Integer
```

```
Private Sub Command1_Click()
    x = Val(Text1.Text)
    y = Val(Text2.Text)
    Call MaxVal(x, y)
End Sub
```



Solved Programs

1. Write a VB program to find the factorial.

Solution

```
Private Sub Command1_Click()
    Dim n As Integer
    Dim fact As Integer
    fact = 1
    n = Val(Text1.Text)
    For i = 1 To n
        fact = fact * i
    Next
    MsgBox "factorial of " & n & " is = " & fact
End Sub
```

2

Apr.13, 12 – 4M

2. Write VB program to find even numbers from array.

Solution

```
Private Sub cmdDisplay_Click()
    Dim a() As Integer
    n = InputBox("How many elements do u want to enter")
    ReDim a(n)
    Print "Even Numbers are:"
    For i = 0 To n - 1
        If a(i) Mod 2 = 0 Then
            Print a(i)
        End If
    Next
End Sub
```

4

Apr.13, 12, 11, 10 – 4M

1

Apr.2013- 4M

3. Write program in VB to check whether given nos. is perfect or not by using msg box.

Solution

```
Private Sub cmdfind_Click()
Dim n As Integer
Dim s As Integer
s = 0
n = Val(Text1.Text)
For i = 1 To n - 1
If n Mod i = 0 Then
s = s + i
End If
Next
If n = s Then
MsgBox "Perfect Number", vbInformation, "Perfect"
Else
MsgBox "Not Perfect Number", vbInformation,
"Perfect"
End If
End Sub
```

1

Apr.2013 - 4M

4. Write a program to transfer the selected elements from list 1 to list 2.

Solution

```
Private Sub cmdMovetoRight_Click()
Dim i As Integer
If List1.ListIndex = -1 Then Exit Sub
For i = List1.ListCount - 1 To 0 Step -1
If List1.Selected(i) = True Then
List2.AddItem List1.List(i)
List1.RemoveItem i
End If
Next i
End Sub
```

1

Oct.2012 - 4M

5. Write a VB program to accept a number from user and calculate sum of even digits of given number.

Solution

```
dim input as integer
dim evenTotal
evenTotal = 0
dim digit as integer
input = Val(InputBox("Enter an even number:"))
while input > 0
```

```

digit = input mod 10
if (digit mod 2 = 0) then
evenTotal = evenTotal + digit
endif
input = input / 10
end while
Print evenTotal

```

6. Write VB program to find the prime number.

Solution

```

Dim I, N As Integer
N = Val(InputBox("Enter a number:"))
For I = 2 To N - 1
If N Mod I = 0 Then
Print "the number is not a prime number"
Exit Sub
End If
Next I
Print "the number is a prime number"

```

1

Oct.2012 - 4M

7. Write VB program to find the roots of quadratic equation.

Solution

```

Dim A As Integer, B As Integer, C As Integer
Dim Root1 As Single, Root2 As Single
A = Val(InputBox("Enter a number:"))
B = Val(InputBox("Enter a number:"))
C = Val(InputBox("Enter a number:"))
Root1 = (-B + Sqr(B ^ 2 - 4 * A * C)) / (2 * A)
Root2 = (-B - Sqr(B ^ 2 - 4 * A * C)) / (2 * A)
Print Root1
Print Root2

```

1

Oct.2012 - 4M

8. Write a VB program to check whether a number is positive or not.

Solution

```

Dim n as integer
n = Val(InputBox("Enter a number:"))
If n < 0 Then
MsgBox "Negative"
ElseIf n > 0 Then
MsgBox "Positive"
Else
MsgBox "Is is 0"
End If

```

1

Oct.2012 - 4M

1

Oct.2012 - 4M

9. Write a VB program to accept the details of doctor from user and store the details into the database. (Don't use Standard Controls)

Doctor having fields Did, Dname, Address, Phono.

Solution

```
Module Module1
Sub Main()
Dim dbfTemp As Database, recTemp As Recordset
dim dtadata As Connection
Set dtaData.Connect = "ODBC;DSN=Personnel
Database"
Set recTemp = db.OpenRecordSet("Details")
recTemp.AddNew
Console.WriteLine("Did")
recTemp.Did = Console.ReadLine
Console.WriteLine("Dname")
recTemp.Dname = Console.ReadLine
Console.WriteLine("Address")
recTemp.Address = Console.ReadLine
Console.WriteLine("Phno")
recTemp.Phno = Console.ReadLine
recTemp.Update
recTemp.Close
dbfTemp.Close
End Sub
End Module
```

1

Apr.2012 - 4M

10. Write a VB Program to concatenate two strings.

Solution

```
Dim str1, str2 As String
Private Sub Form_Load()
str1=InputBox("Enter First String")
str2=InputBox("Enter Second String")
str3=str1+str2
MsgBox("The concatenated String is: "&str3)
End Sub
End Sub
```

1

Oct.2011 - 4M

11. Write a program in Visual Basic to calculate sum of digits of a given number.

Solution

```
Private Sub Form_Load()
Dim n, r, s As Integer
```

```

n = Val(InputBox("Enter the Number"))
While n > 0
r = n Mod 10
s = s + r
n = n \ 10
Wend
MsgBox("Sum of digits of the Number is " & s)
End Sub

```

12. Write a VB Program to find the prime number.

Solution

```

Dim no as Integer, I as Integer
Dim flag as Boolean
Private sub cmoomand1_click()
    Text1.text = valueof(no)
    flag = true
    For I = 2 to no-1
        If (no mod I = 0) then
            flag = false
        End if
    Next I
    If flag = true then
        Print no & "is prime no"
    End if
End sub

```



Oct.2011 - 4M

13. Write program in VB to Print Fibonnacci Series.

Solution

```

Option Explicit
Dim x As Integer, y as Integer, z as Integer, I as Integer
x = 0
y = 1
Print x
Print y
Private Sub command1_Click()
For i = 1 To 20
    z = x + y
    Print z
    x = y
    y = z
Next i
End Sub

```



Apr.2011 - 4M

14. Write program in VB to find greatest number among three numbers.

Solution

```

Dim no1 as Integer, no2 as Integer, no3 as Integer
Private sum Commonad1_Click()
Text1.text = no1
Text2.text = no2

```



Apr.2011 - 4M

```

Text3.text = no3
If (no1 > no2 and no1 > no3) then
    Print no1 & "Is greatest no"
Else if (no2 > no1 and no2 > no3) then
    Print no2 & "Is greatest no"
Else
    Print no3 & "Is greatest no"
End if
End sub

```

1

Apr.2011 - 4M

- 15. Write program in VB to check of a given number is prime or not by using 'msgbox'.**

Solution

```

Dim no as Integer, I as Integer
Dim flag as Boolean
Private sub cmoomand1_click()
    Text1.text = valueof(no)
    Flag = true
    For I = 2 to no-1
        If(no mod I = 0) then
            Flag = false
        End if
    Next I
    If flag = true then
        Print no & "is prime no"
    End if
End sub

```

1

Oct.2010 - 4M

- 16. Write program in VB to find maximum number from an array.**

Solution

```

Option Explicit
Dim num(1 To 10) As Integer
Dim i As Integer, j As Integer, max As Integer
Max = 1
Private Sub cmdArray_Click()
    For i = 1 To 10
        num(i) = InputBox("Enter an integer number")
    If (num(i) > Max) Then
        Max = num(i)
    End If
    Next i
    Print "Max Number in the array is" + Max
End Sub

```

17. Write program in VB to check whether given no. is Armstrong or not.**Solution**

```

Dim num as Integer, rem as Integer, sum as Integer, temp as Integer
num=InputBox("Enter any no")
temp=num
sum=0
While num>0
    rem=num mod 10
    sum=sum+(rem*rem*rem)
    num=num\10
Wend
if temp=sum then
    MsgBox "Given number" & temp & "is Armstrong"
else
    MsgBox "Given number:" & temp & "is not Armstrong"
end if

```

1

Oct. 2010 – 4M

18. Write a VB Program to find Fibonacci Series.**Solution**

```

Option Explicit
Dim a As Integer, b As Integer, c As Integer, cnt As Integer
Private Sub cmdFiboSeries_Click()
a = 0
b = 1
cnt = 3
Print a
Print b
Do
    a = b
    b = c
    c = a + b
    cnt = cnt + 1
    Print c
Loop While cnt <> 20
End Sub

```

2

Apr. 10, 11 – 4M

19. Write a VB program to calculate x^y without using built-in function.**Solution**

```

Private Sub Command1_Click()
    Dim x As Integer, y As Integer, i As Integer, a As Integer
    x = Val(Text1.Text)
    y = Val(Text2.Text)
    For i = 1 To y
        x = x * y
        i = i + 1
    Next i
    Labell1.Caption = x
End Sub

```

1

Apr.2010 – 4M

20. Write a VB program to display age in year, month and days.

Solution

```
Dim dob As Date
Dim age, y, m, d
Private Sub Command1_Click()
dob = Text1.Text
age = DateDiff("yyyy", dob, Now()) + _
    Int(Format(Now(), "mmdd") < Format(dob, "mmdd"))
y = Year(Now()) - Year(dob)
m = Month(Now()) - Month(dob)
d = Day(Now()) - Day(dob)
Print "Year : " & y
Print "Month : " & m
Print "Day : " & d
Print age
End Sub
```

2

Apr. 10, 12 – 4M



PU Questions

4 Marks

- | | |
|------------------------|--|
| <u>[Apr.2013 – 4M]</u> | 1. Explain data types in VB. |
| <u>[Apr.2013 – 4M]</u> | 2. Write a VB program to find the factorial. |
| <u>[Apr.2013 – 4M]</u> | 3. Write VB program to find even numbers from array. |
| <u>[Apr.2013 – 4M]</u> | 4. Write program in VB to check whether given nos. is perfect or not by using msg box. |
| <u>[Apr.2013 – 4M]</u> | 5. Write a program to transfer the selected elements from list 1 to list 2. |
| <u>[Oct.2012 – 4M]</u> | 6. Explain constants with an example. |
| <u>[Oct.2012 – 4M]</u> | 7. Explain SELECT Case in VB with example. |
| <u>[Oct.2012 – 4M]</u> | 8. Write a VB program to accept a number from user and calculate sum of even digits of given number. |
| <u>[Oct.2012 – 4M]</u> | 9. Write VB program to find the prime number. |

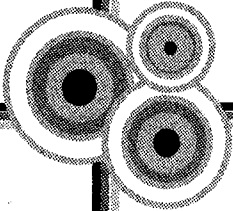
10. Write VB program to find the roots of quadratic equation. **[Oct.2012 – 4M]**
11. Write a VB program to check whether a number is positive or not. **[Oct.2012 – 4M]**
12. Write a VB program to accept the details of doctor from user and store the details into the database. (Don't use Standard Controls) Doctor having fields Did, Dname, Address, Phoneno. **[Oct.2012 – 4M]**
13. Explain any two looping structures used in VB with syntax and example. **[Oct.2012 – 4M]**
14. Describe data types used in VB. **[Apr.2012 – 4M]**
15. Write a program in Visual Basic to calculate sum of digits of a given number. **[Apr.2012 – 4M]**
16. What are Control Arrays? Explain with suitable example. **[Apr.2012 – 4M]**
17. Write a VB Program to display even numbers from an Array. **[Apr.2012 – 4M]**
18. Write a VB Program to concatenate two strings. **[Apr.2012 – 4M]**
19. Write a VB Program to find the factorial of numbers. **[Apr.2012 – 4M]**
20. Write a VB Program to display age in year, month and days. **[Apr.2012 – 4M]**
21. What are Arrays in Visual Basic? **[Oct.2011 – 4M]**
22. Explain IF-then-else statement in Visual Basic, with syntax and example. **[Oct.2011 – 4M]**
23. Write a program in Visual Basic to calculate sum of digits of a given number. **[Oct.2011 – 4M]**
24. Write a VB Program to find the prime number. **[Oct.2011 – 4M]**
25. Write a VB Program to display odd numbers from an array. **[Oct.2011 – 4M]**
26. Explain any four built-in data types in VB. **[Apr.2011 – 4M]**
27. Explain DO WHILE . . . LOOP Statement in brief. **[Apr.2011 – 4M]**
28. Write program in VB to Print Fibonacci Series. **[Apr.2011 – 4M]**
29. Write program in VB to find greatest number among three numbers. **[Apr.2011 – 4M]**
30. Write program in VB to check of a given number is prime or not by using 'msgbox'. **[Apr.2011 – 4M]**
31. Write program in VB to find all the even numbers from an array. **[Apr.2011 – 4M]**

- [Oct.2010 – 4M] 32. Describe data types in VB.
- [Oct.2010 – 4M] 33. Explain IF-THEN-ELSE statement in VB with syntax and example?
- [Oct.2010 – 4M] 34. Write a program in VB to check whether given no. is perfect or not by using 'msgbox'.
- [Oct.2010 – 4M] 35. Write program in VB to find maximum number from an array.
- [Oct.2010 – 4M] 36. Write program in VB to find factorial of a given number.
- [Oct.2010 – 4M] 37. Write program in VB to check whether given no. is Armstrong or not.
- [Apr.2010 – 4M] 38. What do you mean by Variable? Explain Scope of Variables.
- [Apr.2010 – 4M] 39. Explain any two built-in string functions with Syntax and examples.
- [Apr.2010 – 4M] 40. Write a VB Program to find Fibonacci Series.
- [Apr.2010 – 4M] 41. Write a VB program to display even numbers from an array.
- [Apr.2010 – 4M] 42. Write a VB program to calculate x^y without using built-in function.
- [Apr.2010 – 4M] 43. Write a VB program to display age in year, month and days.
- [Apr.2010 – 4M] 44. Explain different control structures used in VB with examples.
- [Apr.2010 – 4M] 45. Write short note on Control Array.

8 Marks

- [Apr.2012 – 8M] 1. Explain any two Looping Structure used in VB with syntax and example.
- [Oct.2011 – 8M] 2. What are procedures and functions in Visual Basic? Explain with syntax and example.
- [Oct.2011 – 8M] 3. What are Control Arrays? Explain with the help of a suitable example.
- [Apr.2011 – 8M] 4. Explain any four string functions.
- [Oct.2010 – 8M] 5. Explain any two looping structures in VB with syntax and examples.

WORKING WITH CONTROLS



1. Introduction

In the first unit we have studied some common properties for all the controls, some form events, and methods. Now you are familiar with the visual basic environment and are able to write simple applications. To work with VB you must be familiar about the controls of the VB and be comfortable in using VB controls. Therefore before moving to the advance features of VB we will study the VB controls in this unit in detail. VB is rich in intrinsic controls. All the controls have multiple properties and methods. Some of them are common but still there are some methods and properties which make that control distinct from other control. This unit will guide you how to use all the main VB controls. It is very important to understand the properties and methods of controls in VB. It can help you to write a good program. So, it is important to give more time to play with controls and properties.

2. Adding Controls on Form

Designing user interface is really an important skill. Adding controls on the form doesn't mean just crowding together some of the controls on the form and writing code for that. Your program's user interface must make your application easy to use. You should keep some points in mind before starting to design user interface. Before designing user interface or forms think about the user, ask some questions to yourself like who is going to use the application, what kind of application you are designing. User should not get confused while using your application. There should be proper color combination. The number of items on one form should not be more. Arrangement of the controls must be user friendly. Keep the forms simple.

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 their appearance and behavior, such as their color, position, size, response to the user input etc. They can respond to events initiated by the user.

For example, you can write code for command button control's click event that would perform any action and display a result.

In addition to properties and events, methods can also be used to manipulate controls from code. Let us discuss these controls by means of a few simple examples in the following sections.

The controls visible on the tool bar are standard controls such as command button, label, textbox; checkbox etc. are contained inside .EXE file which cannot be removed. Other controls are ActiveX controls. They exist as separate files with either .VBX or .OCX extension. They include specialized controls such as; tool bar, progress bar etc.

To add any of the control on the form follow the steps given below:

1. Select the icon of the control on the toolbox. *For example*, command button, label, textbox.
2. Move the mouse to the place on the form where you want to draw the control. (This time the cursor changes to big plus shape).
3. Click and drag the mouse to where you want to draw your control and then release the mouse button.
4. If you double click the control icon on the toolbox the VB will draw that control automatically on the form.

After arranging all the required controls on the form set some of the properties of that controls. It is not mandatory but setting these properties helps you in coding and it is a good programming practice.

Here are some important points about setting up the properties:

1. **Caption:** You should set the caption property of a control clearly so that a user knows what to do with that command. *For example*, in the customer information form, all the command

buttons should have meaningful captions so that they show the purpose of that button and the user should not have any problem in manipulating the buttons.

2. **Name:** A meaningful name for the 'name property' may be easier to write and read the event procedure and easier to debug or modify the programs later.
3. **Enabled:** One more important property is whether the control is enabled or not or you are planning to make it enable at run time.
4. **Visible:** You must also consider making the control visible or invisible at runtime, or when it should become visible or invisible.
5. **ToolTip Text:** This property is used to set help about the control at run time.

3. Working with Properties and Methods of Each Control

Many properties and methods are associated with each control that decide its behavior and appearance. Properties are displayed in property window. Setting properties through property window means design time setting and if you are changing properties through coding it is called as run time setting. Each control is different from other controls and it has some properties which makes it different from other controls. Likewise each control in VB can usually run many kinds of events or procedures; these events are listed in the dropdown list in the code window that is displayed when you double-click on an object and click on the procedures' box. Among the events are loading a form, clicking of a command button, pressing a key on the keyboard or dragging an object and more. For each event, you need to write an event procedure so that it can perform an action or a series of actions.

To start writing an event procedure, you need to double-click an object. *For example*, if you want to write an event procedure when a user clicks a command button, you double-click on the command button and an event procedure will appear as shown in *Figure 3.1*. It takes the following format:

```
Private Sub Command1_Click  
    (Key in your program code here)  
End Sub
```


Now we will discuss all the controls in detail

1. Textbox

The textbox is the standard control used to take input from the user as well as to display the output. It can handle only text (string) and numeric. Numeric text in a textbox can be converted to a numeric data by using the Val (text) function. This is the most frequently used control in VB applications. *This control has many properties and events:*

Oct.2012 – 4M
Write Short note on:
Input Box

1

Properties

- a. **Text:** This property is used to enter text into the textbox. This is the most frequently used property.
- b. **Multiline:** Set the multiline property to true if you need to display text in multiple lines in a textbox.
- c. **Scroll Bars:** Scroll bars will always appear on the textbox when its multiline property is set to true.
- d. **Alignment:** If you set the multiline property to true, you can set the alignment using the alignment property. By default alignment is left-justified.
- e. **Enabled:** Tells the user whether the control is available or not.
- f. **Index:** If you are using control array this property specifies the array index number.
- g. **Locked:** If this control is set to true user can use it else if this control is set to false the control cannot be used.
- h. **MaxLength:** Specifies the maximum number of characters to be input. Default value is set to 0 that means user can input any number of characters.
- i. **MousePointer:** Using this we can set the shape of the mouse pointer when over a textbox.
- j. **PasswordChar:** This is to specify symbolic character to be displayed in the textbox if you are using the textbox for password.
- k. **ToolTipText:** This is the text shown in the small box. Used as a help text at run time when mouse pointer will move over the textbox.
- l. **Visible:** By setting this user can make the textbox control visible or invisible at runtime.

Methods

SetFocus: Transfers focus to the textbox.

Event Procedures

- i. **Change:** Action happens when the textbox changes.
- ii. **Click:** Action happens when the textbox is clicked.
- iii. **GotFocus:** Action happens when the textbox receives the active focus.
- iv. **LostFocus:** Action happens when the textbox loses its focus.
- v. **KeyDown:** Called when a key is pressed while the textbox has the focus.
- vi. **KeyUp:** Called when a key is released while the textbox has the focus.

The following example illustrates a simple program that processes the input from the user.

In this program, two textboxes are displayed on form together with a few labels. The two textboxes are used to accept inputs from the user and one of the labels will be used to display the sum of two numbers that are entered into the two textboxes. A command button is also there to calculate the sum of the two numbers using the plus operator. The program creates a variable sum to accept the summation of values from textbox 1 and textbox 2. The procedure to calculate and to display the output on the label is shown below. The output is shown in *Figure 3.2*.

```
Private Sub Command1_Click()  
'Textboxes read the values and label shows the result.  
    Label3.Caption = Val (Text1.Text) + Val (Text2.Text)  
End Sub  
Private Sub Form_Load()  
'Clears both the textboxes at the time of form load.  
    Text1.Text = ""  
    Text2.Text = ""  
End Sub
```

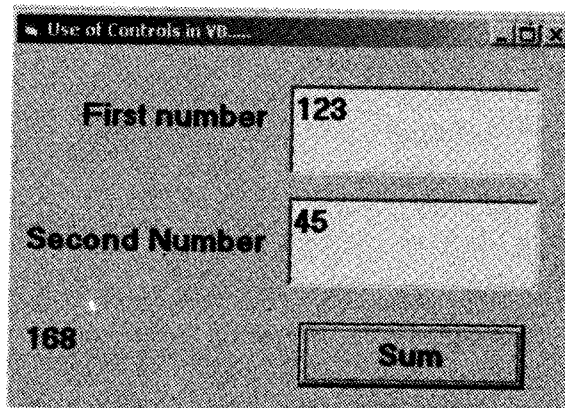


Figure 3.2

2. Label

Label is one of the most frequently used controls after textbox. Mostly it is used to display outputs. It cannot be used to read an input. This control does not have text property. Most people use label controls to provide a descriptive caption and possibly an associated hot key for other controls, like textbox, listbox, etc. that don't support the caption property. Usually we do not write code for label control. This control has events supported by other controls. Label control does not have SetFocus, GotFocus, LostFocus or any keyboard related events.

Some of the properties and events are as follows:

Properties

- a. **Caption:** This property is used to display text and numeric data. You can change caption property at design time through properties window or also at runtime through code.
- b. **BorderStyle:** Used if label control is to appear inside a 3D border.
- c. **Alignment:** If you want to align the caption to the right or center on the control.
- d. **WordWrap:** If the caption string is long, you can set WordWrap property to true.
- e. **AutoSize:** Set the AutoSize property to true and let the control automatically resize itself to accommodate longer caption strings.
- f. **BackStyle:** If this property is set to 0-Transparent. It makes the label invisible, but if you have set Tooltip text it is displayed when mouse moves from the label.
- g. **BackColor:** Sets the background color of the label. But if you have set BackStyle property to transparent then backcolor property is not visible.
- h. **ForeColor:** Sets the color of text.

Events

- i. **Click:** Action happens when the label is clicked.
- ii. **Change:** Action happens when the label changes. If you're using a label control to display data read from a database, you might sometimes find it useful to write code in its change event.
- iii. **DbClick:** Action happens when the label is double clicked.
- iv. **MouseDown:** Action happens when left mouse button is pressed.
- v. **MouseMove:** Action happens when mouse is moved from the label.
- vi. **MouseUp:** Action happens when mouse button is released.

vii. **ToolTipText**: provide help about the control.

Following *example* shows the use of label. There are 2 labels displayed on the form. And when you click on label1 the click event will be triggered and the message will be displayed on the label2.

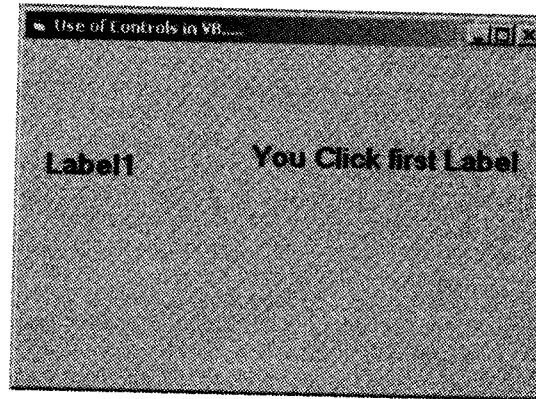


Figure 3.3

```
Private Sub Label1_Click()  
    Label2.AutoSize = True  
    Label2.Font.Bold = True  
    Label2.ForeColor = vbBlue  
    Label2.Caption = "You Click first Label"  
End Sub
```

3. Command Button

The command button is one of the most important controls as it is used to execute commands. It displays an illusion that the button is pressed when the user clicks on it. The most common event associated with the command button is the click event.

The command button control supports the usual set of keyboard and mouse events (KeyDown, KeyPress, KeyUp, MouseDown, MouseMove, MouseUp, but not the DblClick event) and also the GotFocus and LostFocus events, but you'll rarely have to write code in the corresponding event procedures.

Properties of a Command Button control

- a. **Caption**: Used to display text on a command button control.
- b. **BackColor**: To set the background color of the command button, select a color in the BackColor property.

- c. **Enabled:** To enable or disable the buttons set the enabled property to true or false. If this property is false the button does not respond to any event. The button is visible on the form but dimmed.
- d. **Visible:** To make visible or invisible the buttons at run time, set the visible property to true or false. If visible property is set to false the control remains on the form but you can't see it at run time.
- e. **Tooltips:** This can be added to a button by setting a text to the Tooltip property of the CommandButton.
- f. **Style:** If user wants to display image or any graphics on the command button, style property can be set to graphical. If it is standard you can display only text.
- g. **Default:** You can set the default property to true if the button that receives a click when the user presses the enter key.
- h. **Cancel:** You can set the cancel property to true if you want to associate the button with the Escape key.
- i. **MousePointer:** Determines the shape of the mouse cursor when the user moves the mouse over the command button.
- j. **TabIndex:** Specifies the order of the command button in the focus order.
- k. **TabStop:** Determines whether the command button can receive the focus.

Events

The Command Button control supports the usual set of keyboard and mouse events it does not support DblClick (Double click) event.

- i. **Click:** This is the most commonly used event of the command button. The event is triggered when you press and release the command button through mouse
- ii. **MouseDown:** Action happens when left mouse button is pressed.
- iii. **MouseMove:** Action happens when mouse is moved from the command button.
- iv. **GotFocus:** Action happens when the command button receives the active focus.
- v. **LostFocus:** Action happens when the command button loses its focus.
- vi. **KeyDown:** Called when a key is pressed while the command button has the focus.
- vii. **KeyUp:** Called when a key is released while the command button has the focus.

Following *example* shows the use of command buttons. In this example five command buttons are displayed on the form. Each button fires click event when it is clicked, and performs different actions. The four buttons at the bottom of the form will change the shape of the shape control and the button at the corner of the form will end the application.

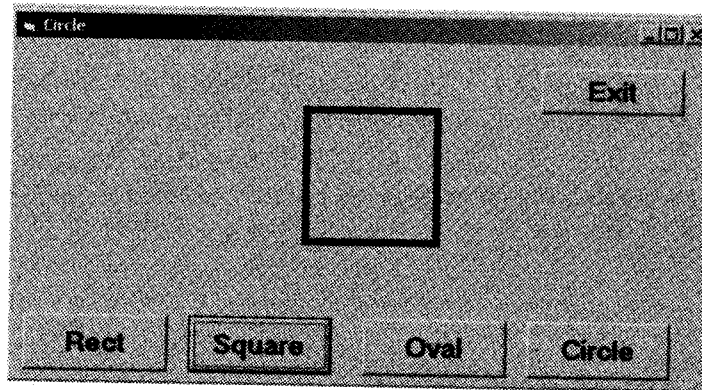


Figure 3.4

The code goes as follows:

```
Private Sub Command1_Click()  
    Shape1.Shape = 0  
    Shape1.BorderWidth = 5  
End Sub  
Private Sub Command2_Click()  
    Shape1.Shape = 1  
    Shape1.BorderWidth = 5  
End Sub  
Private Sub Command3_Click()  
    Shape1.Shape = 2  
    Shape1.BorderWidth = 5  
End Sub  
Private Sub Command4_Click()  
    Shape1.Shape = 3  
    Shape1.BorderWidth = 5  
End Sub  
Private Sub Command5_Click()  
    End  
End Sub
```

4. PictureBox

PictureBox is the control designed specifically to display images. PictureBox control can be used as a container control; in addition to displaying pictures it can be used to group and display other controls. The PictureBox is one of the controls that are 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. The image in the picturebox is not resizable. PictureBox control is one of the most powerful and complex items in the Visual Basic toolbox window.

Properties

Picturebox controls support all the properties related to graphic output, including AutoRedraw, ClipControls, HasDC, FontTransparent, CurrentX, CurrentY, and all the Drawxxxx, Fillxxxx, and Scalexxxx properties.

2

Apr.13, 12 - 4M
Write short note on
Picture Box.

Some of the properties are:

- a. **Picture:** Once you place a picturebox on a form, you can load an image in it. You can load picture by setting the picture property in the properties window. You can load images in many different graphic formats, including bitmaps (BMP), device independent bitmaps (DIB), metafiles (WMF), enhanced metafiles (EMF), GIF and JPEG compressed files, and icons (ICO and CUR).

You can make your application more interesting by loading picture at run time. You can programmatically load any image in the control using the LoadPicture function

```
Picture1.Picture=LoadPicture ("G:\Photo\Paintings\image1.jpg")
```

And you can clear the current image using either one of the following statements:

```
Picture1.Picture = LoadPicture("")
```

Or

```
Set Picture1.Picture = Nothing
```

Both are same.

VB6 picturebox control also supports icon files containing multiple icons.

Syntax

```
LoadPicture(filename, [size], [colordepth], [x], [y])
```

Where,

Values in square brackets are optional.

If filename is an icon file, you can select a particular icon using the size or colordepth arguments.

You can copy an image from one picturebox control to another by assigning the target control's picture property as shown in the following code:

```
Picture2.Picture = Picture1.Picture
```

- b. **BorderStyle:** You can display border to this control. Default BorderStyle is 0-None. If necessary you can change this to another property i.e. AutoSize. Set it to True and let the control automatically resize itself to fit the assigned image.
- c. **AutoSize:** Using the AutoSize value causes the control to resize to always fit the image.

- d. **BackColor:** Sets the background color of the control.
- e. **Align:** You can set the Align property of a picturebox control to something other than the 0-None value. By doing that, you attach the control to one of the four form borders and have Visual Basic automatically move and resize the picturebox control when the form is resized. picturebox controls expose a resize event, so you can trap it if you need to move and resize its child controls too.

Methods

Picturebox controls also support all graphic methods, such as Cls, PSet, Point, Line, and Circle and conversion methods, such as ScaleX, ScaleY, TextWidth, and TextHeight.

- i. **LoadPicture:** You can also load the picture at runtime using the **LoadPicture** method.
- ii. **PaintPicture:** Picturebox controls enable the programmer to perform a wide variety of graphic effects, including zooming, scrolling, panning, tiling, flipping, and many fading effects. The PaintPicture method performs a pixel-by-pixel copy from a source control to a destination control.

Syntax

```
DestPictureBox.PaintPicture SrcPictureBox.Picture, destX, destY,  
[destWidth], _[destHeight], [srcX], [srcY2], [srcWidth],  
[srcHeight], [Opcode])
```

Where,

The only required arguments are the source picturebox control's picture property (i.e. SrcPictureBox.Picture), the coordinates inside the destination control where the image must be copied.

The destX / destY arguments are expressed in the ScaleMode of the destination control; by varying them, you can make the image appear exactly where you want.

5. Imagebox

The imagebox is another control that holds images and pictures. This is called as lightweight control. Image control is much similar to picturebox control. However, there is one major difference, the image in an imagebox is stretchable, which means it can be resized. This feature is not available in the picturebox. Similar to the picturebox, it can also use the LoadPicture method to load the picture. *For example*, the statement loads the picture image008.jpg into the imagebox.

```
Image1.Picture=LoadPicture("G:\Photo\paintings\ iamg008.jpg")
```

As compared to picturebox control image controls are less complicated. Image control does not support some of the property like graphical methods, AutoRedraw, ClipControls properties that picturebox supports. Like picture control, image control does not work like a container. But image control is always preferable over picturebox control because they load faster and

consume less memory and system resources. Remember that image controls are windowless objects that are actually managed by Visual Basic without creating a windows object. Image controls can load bitmaps and JPEG and GIF images.

Properties

- a. **Picture:** You can load a picture into an image control by using picture property. It opens a dialogbox that lets you choose what image file to load.
- b. **Stretch:** This is the property supported by image control which is not supported by picture control. Stretch property determines whether the image control adjusts to fit the picture or the picture adjusts to fit the control. It is set to *False* by default. When the stretch property is false, the image control will automatically resize itself to expand or contract to the size of the picture that is assigned to it. If stretch property is set to true, the picture resizes to fit the control. Depending on the type of image, the image may or may not appear distorted when stretched this is something you need to experiment with.

Method

You can also set the picture property of an image control in code by using the *LoadPicture* function, which loads a picture from a file into the control at run-time. The *LoadPicture* function takes a file name as an argument. Sample syntax is:

```
Image1.Picture = LoadPicture("G:\photo\paintings\image008.jpg")
```

Using picture property is optional. As it is the default property of the image. So the following line of code is also valid:

```
Image1 = LoadPicture("G:\photo\paintings\image008.jpg")
```

6. **Checkbox**

To perform Boolean operations checkbox control is used. It lets the user check, uncheck or disable an option. The checkbox control is similar to the option button, except that a checkbox allows you to select more than one checkbox at a time whereas you cannot make multiple selection in option button.

Properties

- a. **Value:** This property supports three values for the checkbox. These are unchecked (0), checked (1), grayed (2). When the checkbox is checked, its value is set to 1; when it is unchecked, the value is set to 0 and when it is greyed its value is set to 2.

The code used to set value property is:

```
Check1.Value=1 'Set Checked
```

```
Check1.Value=0 'Set Unchecked
```

```
Check1.Value=2 'Set Greyed
```

- b. **Style:** Determines the look of the control. If it is set to standard (0) the control displays the rectangular box and caption of the control. When it is selected it ticks the right mark in that box. Whereas Graphical (1) style looks like command button; if it is selected it looks like a pressed button and becomes white in color. The standard style of the checkbox is more preferable.
- c. **Alignment:** This property aligns the text of the control.
1-Right sets the text before the checkbox, where as the default i.e. 0-Left Justify sets the text after checkbox which is more preferable.
- d. **Picture:** Using this property you can apply the image on the checkbox. But to display image the style property must be set to graphical.

Event

- i. **Click:** This is the important event for checkbox control. It is fired when either the user or the code changes the value of the control.

Example,

The program will change the background color of the form to red and checkbox caption to blue when the checkbox is checked. It will change to blue and make checkbox caption to red when the checkbox is unchecked. VbRed and vbBlue are color constants and BackColor is the background color property of the form and caption is the caption property of the checkbox control.

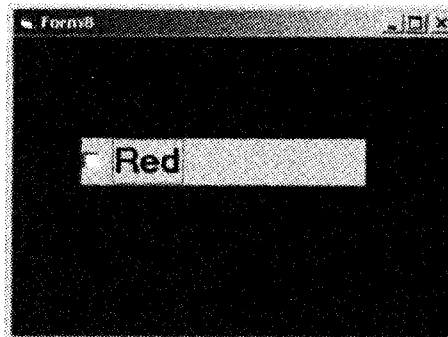


Figure 3.5

```
Private Sub Check1_Click()  
    If Check1.Value = 0 Then  
        Form8.BackColor = vbBlue  
        Check1.Caption = "Red"  
    Else  
        Form8.BackColor = vbRed  
        Check1.Caption = "Blue"  
    End If  
End Sub
```

7. **Option button:** Option button controls are also known as radio buttons because of their shape. Option button are very much similar to checkbox control, the only difference is that you always use Option button controls in a group of two or more because their purpose is to offer multiple choices. Anytime you click on a button in the group, it switches to a selected state and all the other controls in the group become unselected. Generally option button are grouped in a frame. If you have multiple groups of option button on a form it is better to put in frame to separate the group.

Apr.2012 – 4M

Write short note on
Option Button or Radio
Button.

1

Properties

- a. **Value:** This property of the option button tells whether the button was selected or not. If the value property is set to true, the button is selected. If the value is false, the button is not selected.
- b. **Style:** Determines the look of the control. If it is set to standard (0) the control displays the empty circle and caption of the control. When it is selected it shows filled circle. The graphical (1) style looks like command button; if it is selected it looks like a pressed button and becomes white in color. The standard style of the option button is more preferable.
- c. **Alignment:** This property aligns the text of the control.

1-Right sets the text before the checkbox, whereas the default i.e. 0-Left Justify sets the text after checkbox which is more preferable.

Example, In the following *example*, the shape control is placed in the form with six option boxes. When the user clicks on different option boxes, different shapes will appear. The values of the shape control are 0, 1, and 2, 3 which will make it appear as a circle, rectangle, an oval shape, and a square respectively.

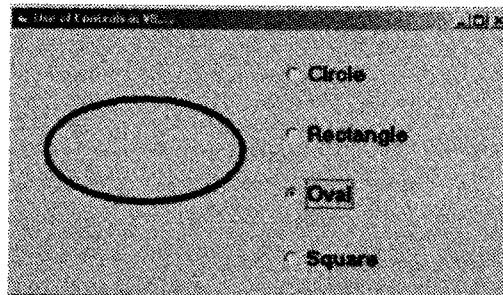


Figure 3.6


```
Private Sub Option1_Click()  
    Shape1.Shape = 3  
    Shape1.BorderWidth = 5  
End Sub  
Private Sub Option2_Click()  
    Shape1.Shape = 0  
    Shape1.BorderWidth = 5  
End Sub  
Private Sub Option3_Click()  
    Shape1.Shape = 2  
    Shape1.BorderWidth = 5  
End Sub  
Private Sub Option4_Click()  
    Shape1.Shape = 1  
    Shape1.BorderWidth = 5  
End Sub
```

Drivelistbox, dirlistbox, and filelistbox

Three controls on the toolbox let you access the computer's file system. These are `drivelistbox`, `dirlistbox` and `filelistbox` controls. These are the basic blocks for building dialogboxes that display the host computer's file system. Using these controls, user can traverse the host computer's file system; locate any folder or files on any hard disk, even on network drives. All these three controls are independent of one another, and each can exist on its own, but they are generally used together. When you select a drive in a `drivelistbox`, the `dirlistbox` control is updated to show the directory tree on that drive. When the user selects a path in the `dirlistbox` control, the `filelistbox` control shows the list of all the files in that directory. All these actions are interdependent.

All these controls are described as follows:

- 1. Drivelistbox:** This control is a combobox-like control, when it is displayed in the form it automatically lists all the drive's letters and volume labels available on your local disk. When you place this control into the form and run the program, you will be able to select different drives from your computer. The basic property of this control is the `drive` property, which sets the drive to be initially selected in the control or returns the user's selection.
- 2. Dirlistbox:** The Directory listbox is for displaying the list of directories or folders 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 basic property of this control is the `path` property, which is the name of the folder whose sub folders are displayed in the control.
- 3. Filelistbox:** This control is a special-purpose control that displays all the files in a current directory. Most of the properties are similar to listbox control. You can filter the files based on

their names, extensions, and attributes. As the size of the control increases VB automatically adds scroll bars.

The most important property of this control is the *Pattern* and *Path* property. *Pattern* determines which files are displayed in the file listbox. Its default value is *.* (all files), but you can enter whatever specification you need, and you can also enter multiple specifications using the semicolon as a separator. The default pattern is set to *.* to display all files. You can also set this property at run time, as in the following line of code:

```
File1.Pattern = "*.txt;*.doc;*.rtf"
```

The *Path* property sets the current path for the file listbox, but not for the underlying operating system.

Though the features of the *drivelistbox*, *dirlistbox* and *filelistbox* controls are powerful still they have some limitations. They are not accepted for commercial applications. These controls are not useful when listing files on network servers and sometimes even on local disk drives, when long file and directory names are used. To solve this problem dialogbox controls are preferred.

For example,

1. Listbox

The function of the listbox is 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. listbox and combobox controls present a set of choices that are displayed vertically in a column. If the number of items exceed the value that is to be displayed, scroll bars will automatically appear on the control. These scroll bars can be scrolled up and down or left to right through the list.

1
Oct.2011 – 8M
Difference between list
box and combo box.

Properties

- a. **AddItem:** This is used to add item to the list at run time. The *AddItem* property can be used as follows.

```
List1.AddItem "Apple"
```

To add items to a list at design time, click on list property in the propertybox and then add the items. Press CTRL+ENTER after adding each item. The *AddItem* method is used to add items to a list at run time. The *AddItem* method uses the following syntax.

```
Object.AddItem item, Index
```

The *item* argument is a string that represents the text to add to the list.

Index is optional. The *index* argument is an integer that indicates where in the list to add the new item. If index number is not given, listbox automatically takes default index.

For example,

```
List1.AddItem "Apple", 0
```

```
List1.AddItem "Banana", 1
```

- b. **ListIndex:** The items in the listbox can be identified by the ListIndex property, the value of the ListIndex for the first item is 0, the second item has a ListIndex 1, and the second item has a ListIndex 2 and so on.
- c. **RemoveItem:** The RemoveItem method is used to remove an item from a list.

The syntax for this is given below

```
Object.RemoveItem index
```

For example,

```
List1.RemoveItem List1. ListIndex
```

- d. **Sorted:** The sorted property is set to true to enable a list to appear in alphanumeric order and false to display the list items in the order which they are added to the list.

2. Combobox

The function of the combobox is also to present a list of items where the user can click and select the items from the list. However, the user needs to click on the small arrowhead on the right of the combobox to see the items which are presented in a drop-down list. In order to add items to the list, you can also use the *AddItem method*.

A combobox combines the features of a textbox and listbox. This enables the user to select either by typing text into the combobox or by selecting an item from the list. Combobox supports three styles that are Dropdown Combo (style 0), Simple Combo (style 1), and Dropdown List (style 2).

The Simple combobox displays an edit area with an attached listbox which is always visible immediately below the edit area. A simple combobox displays the contents of its list all the time. The user can select an item from the list or type an item in the editbox portion of the combobox. A scroll bar is displayed beside the list if there are too many items to be displayed in the listbox area.

The Dropdown combobox first appears as only an edit area with a down arrow button at the right. The list portion stays hidden until the user clicks the down-arrow button to drop down the list portion. The user can either select a value from the list or type a value in the edit area.

Common list and combobox properties

- a. **Enabled:** By setting this property to true or false user can decide whether user can interact with this control or not.
- b. **Index:** Specifies the control array index.
- c. **List:** String array contains the strings displayed in the drop-down list. Starting array index is 0.
- d. **ListCount:** It is an integer value. Shows the total number of drop-down list items.

- e. **ListIndex:** This is also an integer number which contains the index of the selected combobox item. If an item is not selected, ListIndex is -1.
- f. **Locked:** Boolean value. Specifies whether user can type or not in the combobox.
- g. **MousePointer:** Integer. Specifies the shape of the mouse pointer when over the area of the combobox.
- h. **NewIndex:** Integer. Index of the last item added to the combobox. If the combobox does not contain any items, NewIndex is -1
- i. **Sorted:** Specifies whether the control's items are sorted or not.
- j. **Style:** Integer. Specifies the style of the control's appearance
- k. **TabStop:** Boolean. Specifies whether control receives the focus or not.
- l. **Text:** Specifies the selected item in the control's list
- m. **ToolTipIndex:** Specifies what text is displayed as the control's tool tip
- n. **Visible:** Boolean. Specifies whether control is visible or not at run time

Methods

AddItem: Add an item to the control's list.

Clear: Removes all items from the control's list.

RemoveItem: Removes the specified item from the control.

SetFocus: Transfers focus to the control.

Event Procedures

- i. **Change:** Called when text in control is changed.
- ii. **DropDown:** Called when the control drop-down list is displayed.
- iii. **GotFocus:** Called when control receives the focus.
- iv. **LostFocus:** Called when control loses its focus.

Other Controls

1. Frame Controls

Frame control is generally used to group the controls like option buttons, checkboxes etc. It is used to separate the groups. Thus it helps to elaborate the form and your application gets the professional touch. Controls that are contained in the frame control are said to be child controls. To make the control a child control of the frame, first create a frame control, then by

selecting the child control's icon in the toolbox and drawing a new instance inside the frame's border will make that control child control the frame. The benefit of grouping the control is if you move a frame control, all the child controls go with it. If you make a container control disabled or invisible, all its child controls also become disabled or invisible. You can exploit these features to quickly change the state of a group of related controls.

2. Timer Control

This control is used if you want something to happen automatically. A timer control is invisible at run time; its icon appears only at design time. You can trap this pulse by writing code in the timer's timer event procedure and take advantage of it to execute a task in the background or to monitor a user's actions. This is a very interesting control used very rarely in the application, commonly used for updating status information on a regular basis. Using number of timers in a single form may create confusion.

Timer event procedure should not include lot of code because this code will be executed at every pulse and therefore can easily decrease application's performance. Do not use DoEvents statement inside a timer event procedure because it might cause the procedure to be reentered, especially if the interval property is set to a small value.

Properties

This is the control in VB having very few properties still very powerful

- a. **Name:** This is the common property for all controls. Default name is Timer1, Timer2 and so on.
- b. **Interval:** Interval stands for the number of milliseconds between subsequent pulses, these pulses are measured by milliseconds. When you place the timer control on a form, default interval is 0, which means timer is disabled. The range of milliseconds you can specify to this control is between 1 to 65535. Remember to set this property to a suitable value in the properties window or in the Form_Load event procedure to enable the timer. You can set interval through property window or through coding as shown below:

```
Private Sub Form_Load ()  
    Timer1.Interval = 400  
End Sub
```

- c. **Enabled:** This is a Boolean property which determines the timer is activated or deactivated.

3. Line Control

The line control is a decorative control whose only purpose is to let you draw one or more straight lines.

Properties

This control has few properties:

- a. **BorderColor:** Sets the color of the line.
- b. **BorderStyle:** You can apply different styles to the line, like dotted, dashed lines, solid etc.
- c. **BorderWidth:** Determines the thickness of the line. It is measured in pixel and can range from 0 to 8192.
- d. **X1, X2, Y1, Y2:** These points determine where the line should appear on the form.

4. Shape Control

The shape control is an extension of the line control. All line control, properties are supported by shape.

A few properties of shape control are different, they are as follows:

- a. **Shape:** This determines the type of the shape. There are six basic shapes that are supported by shape control: rectangle (0), square (1), oval (2), circle (3), rounded rectangle (4), and rounded square (5).
- b. **BackColor:** This property determines the background of the shape.
- c. **BorderWidth:** Determines the thickness of the line. It is measured in pixel and can range from 0 to 8192.
- d. **BorderStyle:** You can apply different styles to the line, like dotted, dashed lines, solid. There are six possible border styles supported by this control.
- e. **FillColor, FillStyle:** FillColor determines the color used to fill the shape in the manner set by the FillStyle property.

FillStyle property supports eight possible styles i.e. no border (0), solid (1), dashed line (2), dotted (3), dash dot (4), dash dot dot (5)

5. Scrollbar

The scrollbar is a commonly used control, which enables the user to select a value by positioning it at the desired location. Visual Basic supports two scroll bars that are HScrollBar (Horizontal Scroll Bar) and VScrollBar (Vertical Scroll Bar). The HScrollBar and the VScrollBar controls are perfectly identical, apart from their different orientation.

Properties

Scrollbar support total 26 properties, out of which 5 are special properties:

- a. **Value:** Indicates the position of the scroll bar. It represents a set of values. The value property of the scrollbar represents its current value that may be any integer between minimum and maximum values assigned. This is an integer number.
- b. **Min:** This is an integer number that defines the minimum value for the scroll bar. Generally we set 0 as a min value.
- c. **Max:** This is an integer number that defines the maximum value for the scroll bar.
- d. **SmallChange:** Small change is the variation in value you get when clicking on the scroll bar's arrows. If the user clicks the up scroll arrow, the value property of the scroll bar increases by the amount of SmallChange until the value property reaches the value of the max property. If the user clicks the down scroll arrow it decreases the value property similarly. The default value for SmallChange is 1. It can be set in the range between 1 to 32768.
- e. **LargeChange:** LargeChange is the variation you get when you click on either side of the scroll bar indicator. The default initial value for those two properties is 1, but you'll probably have to change LargeChange to a higher value. It too ranges between 1 to 32768.

Events

There are two key events for scrollbar controls

- a. **Change event:** This event is triggered when you click on the scroll bar arrows or when you drag the indicator; the scroll event fires while you drag the indicator.
- b. **Scroll:** With this event you get continuous updates as the action is happening. This is useful specially when you are using the long document. It updates the screen immediately to show the result of their scrolling actions.

4. Creating MDI Applications

The Multiple Document Interface (MDI) was designed to simplify the exchange of information among documents, all under the same roof. With the main application, you can maintain multiple open windows, but not multiple copies of the application. Data exchange is easier when you can view and compare many documents simultaneously.

An MDI application must have at least two forms, the parent form and one or more child forms. Each of these forms have certain properties. There can be many child forms contained within the parent form, but there can be only one parent form.

The parent form may not contain any controls. While the parent form is open in design mode, the icons on the toolbox are not displayed, but you can't place any controls on the form. The parent form can, and usually has its own menu.

To create an MDI application, follow these steps:

1. Start a new project.
2. Select Project → Add MDI form to add the parent form.
3. Set the Form's caption to MDI Window
4. Now select Project → Add form to add a SDI form.
5. Make this form as child of MDI form by setting the MDI child property of the SDI form to true. Set the caption property to MDI child window.

Visual Basic automatically associates this new form with the parent form. This child form can't exist outside the parent form.

MDI forms can have their own menus and it can contain only child forms. The MDI form usually has a menu with two commands to load a new child form and to quit the application. The child form can have any number of commands in its menu, according to the application. When the child form is loaded, the child form's menu replaces the original menu on the MDI form.

4.1 Working with Multiple Forms

Generally a window based application contains more than one form. In such applications these multiple forms are related to one another in one or the other way.

For example, consider a window based Library Management System. This application has a login form, a book information entry form, member information entry form, book issue and return form and so on. When librarian runs this application, the first form that he sees is the login form. If login is successful, then only librarian can access other forms in the application. Similarly, if there are some book entries and member entries then only librarian can work with issue and return form.

We can create these multiple forms in MDI or Non-MDI applications.

Following are the steps for working with multiple forms:

Step 1: Add a New Form to the Project

By default, Visual Basic creates a startup form for us when we create a new Project. To add a second (or additional) form to a project, all we need to do is select Project - Add Form from the Visual Basic Menu Bar.

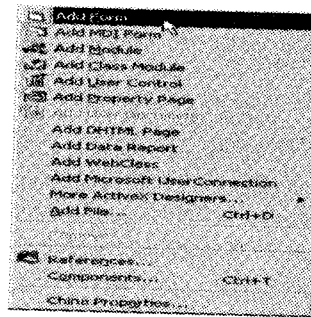


Figure 3.7

The following screen shot will appear.

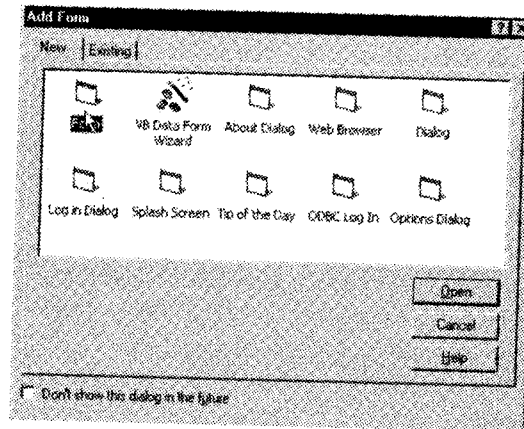


Figure 3.8

At this point, we just need to make sure that the 'New' tab is selected, and then either double-click on 'form' or select 'form' and click on the 'open' button. Visual Basic will then display a new form for us in the IDE, with a caption reading 'form1'. We now have two forms loaded into the IDE.

Step 2: Design the New Form

Our project now has two forms, form1 (main form) and form2 (new form). At this point, we need to design the second form, and then we'll add code to display it from the main form. Let's add a single label control and a command button on second form. We'll change the name of the command button to cmdOK, and change its caption Property to OK. Finally, change the name of the Label to lblAbout.

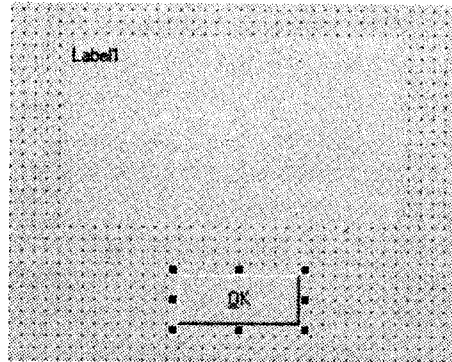


Figure 3.9

Step 3: Write Code for the New Form

Our next step is to write some code so that when the second form is displayed, the caption of the label control displays the information that we wish to display. We will also need to write code that will permit the user to close the form when he or she clicks on the OK button.

Let's start with the caption of the label. We'll place that code in the load event procedure of the form. Here's the code.

```
Private Sub Form_Load()  
lblAbout.Caption = "This is demo application to work with multiple  
forms."  
End Sub
```

Finally, here's the code to close the second form, which we'll place in the Click Event Procedure of cmdOK.

```
Private Sub cmdOK_Click()  
Unload Me  
Set form2 = Nothing  
End Sub
```

Step 4: Write code to display the new form

We will display the second form from the main form (first form) on the click of a command button. So, place a command button (cmdShow) on the main form and on the click procedure write the following code:

```
Private Sub cmdShow_Click()  
Load form2  
form2.Show  
End Sub
```

4.2 Loading, Showing and Hiding Forms

Loading and Unloading Forms

In order to load and unload the forms, load and unload statements are used. The Load statement has the following syntax:

```
Load FormName
```

And the unload statement has the following syntax:

```
Unload FormName
```

The FormName variable is the name of the form to be loaded or unloaded. Unlike the show method which cares of both loading and displaying the form, the load statement doesn't show the form. You have to call the form's show method to display it on the desktop.

Showing Forms

Show method is used to show a form. If the form is loaded but invisible, the show method is used to bring the form on top of every other window. If the form is not loaded, the show method loads it and then displays it.

Syntax of the show method of the form

```
FormName.Show
```

Hiding Forms

The hide method is used to hide a form. The following is the syntax of the hide method.

```
FormName.Hide
```

To hide a form from within its own code, the following code can be used.

```
Me.Hide
```

You must understand that the forms that are hidden are not unloaded; they remain in the memory and can be displayed instantly with the show method. When a form is hidden, you can still access its properties and code. For instance, you can change the settings of its control Properties or call any public functions in the form.

Finding out the difference between Unload and Hide method

To know what the difference is between Unload and Hide methods we will do an example. Open a new project and save the project. Draw two buttons on the form and name those as shown below:

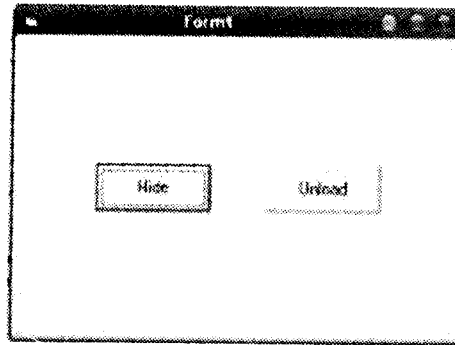


Figure 3.10

In the click event of the Hide button following code is entered.

```
Me.Hide
```

In the click event of the Unload button following code is entered.

```
Unload Me
```

Save the project and run the application. Once you click on Hide button you can note that the form is invisible but the application is still running. But when you click on Unload button you can see that the application is terminated.

4.3 Setting the start-up Form

A typical application has more than a single form. When an application runs the main form is loaded. By setting the Project properties you can control which form is to be displayed in the start-up of the application.

Follow the following steps to change startup object:

1. Select Project -> Project Properties (In place of project in project properties option, the name of the current project name is displayed).
2. Visual Basic displays project properties dialog box.
3. Select General tab if it is not selected.
4. From Startup Object drop down, select the desired form name to make it the startup object.
5. Click on Ok to close properties dialog box.

Now, if you run the project, the form that you set as a start up form will be displayed as the first form.

4.4 Creating Forms in Code

We can create the forms in VB dynamically as and when required. The steps are:

Step 1: Create a list of global variables for the list to follow, *for example*:

```
Option Explicit
Private allowNumericOnly As Boolean
Private frm As Form
Private lblDisplay As Button
Private WithEvents cmdOK As CommandButton
Private WithEvents cmdCancel As CommandButton
Private WithEvents txtInput As TextBox
```

Step 2: Create a procedure for the form that will determine how the form appears to the user and what, if any text and captions will appear. Use the following example of code to set this up for your form:

```
Private Sub GenerateRuntimeForm()
Dim ctrl As Control
Set frm = New Form 1
Set cmdOK = Nothing
Set cmdCancel = Nothing
Set txtInput = Nothing
Set lblDisplay = Nothing
For Each ctrl In frm
ctrl.Visible = False
Next
```

Step 3: Set the different commands for the buttons, using the following code as a basis for your project:

```
Set cmdOK = frm.Controls.Add("VB.CommandButton", "cmdOK")
Set cmdCancel = frm.Controls.Add("VB.CommandButton", "cmdCancel")
Set txtInput = frm.Controls.Add("VB.TextBox", "txtInput")
```

Step 4: Complete the form code by adding the following display conditions and ending the subroutine with "End Sub" as follows:

```
cmdOK.Visible = True
cmdCancel.Visible = True
lblDisplay.Visible = True
txtInput.Visible = True
form.sow vbModal
End Sub
```

4.5 Arranging MDI Child Window

Often, applications will have menu commands for actions such as tile, cascade, and arrange, with regard to the open MDI child forms. You can use the `LayoutMDI` procedure with the `MDILayout` enumeration to rearrange the child forms in an MDI parent form.

One of the four different `MDILayout` enumeration values can be used by the `LayoutMDI` procedure. The enumeration values will display child forms as cascading, as horizontally or vertically tiled, or as child form icons arranged along the lower portion of the MDI form.

Often, these methods are used as the event handlers called by a menu item's click event. In this way, a menu item with the text "Cascade Windows" can have the desired effect on the MDI child windows.

The `LayoutMDI` enumerations are:

Member name	Description
<code>Arrangelcons</code>	All MDI child icons are arranged within the client region of the MDI parent form.
<code>Cascade</code>	All MDI child windows are cascaded within the client region of the MDI parent form.
<code>TileHorizontal</code>	All MDI child windows are tiled horizontally within the client region of the MDI parent form.
<code>TileVertical</code>	All MDI child windows are tiled vertically within the client region of the MDI parent form.

Step to Create and Implement MDI Child Form

1. Assume there is an MDI parent form having `MenuStrip` with option new, window and close. In new menu, main form contains one child form having a `RichTextBox`.

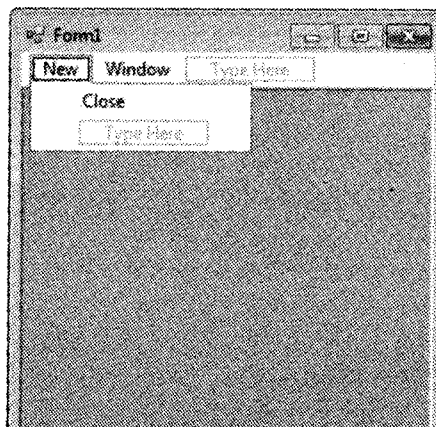


Figure 3.11

2. Add one more control in main form MenuStrip as cascade Windows.

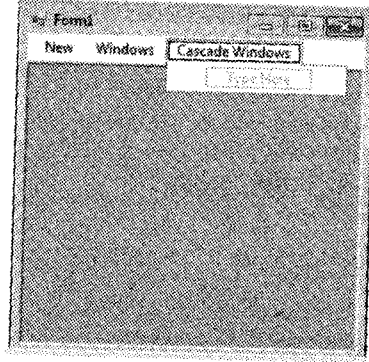


Figure 3.12

3. Double click on cascade windows control and write this code.
`Me.LayoutMdi (MdiLayout.Cascade)`
4. Debug the application and click on New button two times then two MDI child form with RichTextBox will open. Now by using cascade windows control in the main menu you can arrange all the opened MDI Child form in cascade mode.

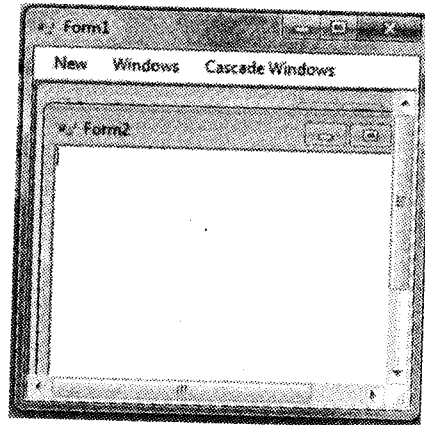


Figure 3.13

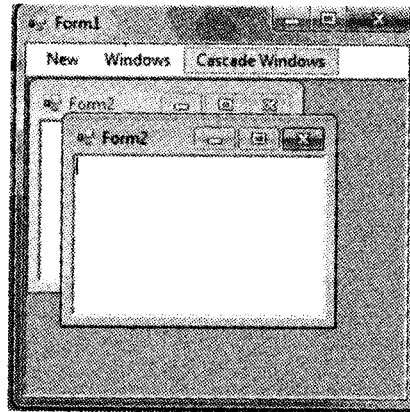


Figure 3.14

4.6 Opening new MDI Child Window

MDI form cannot contain objects other than child forms, but MDI forms can have their own menus. However, because most of the operations of the application have meaning only if there is at least one child form open, there's a peculiarity about the MDI forms.

The MDI form usually has a menu with two commands to load a new child form and to quit the application. The child form can have any number of commands in its menu, according to the application. When the child form is loaded, the child form's menu replaces the original menu on the MDI form.

Following example illustrates the above explanation:

1. Open a new Project and name the form as Menu.frm and save the Project as Menu.vbp
2. Design a menu that has the following structure.
MDIMenu Menu caption
 - MDIOpen opens a new child form
 - MDIExit terminates the application
3. Then design the following menu for the child form ChildMenu Menu caption
 - Child open opens a new child form
 - Child Save saves the document in the active child form
 - Child Close Closes the active child form

4. At design time double click on MDI Open and add the following code in the click event of the open menu.

```
Form1.Show
```

And so double click on MDI Exit and add the following code in the click event

```
End
```

Double click on Child Close and enter the following code in the click event

```
Unload Me
```

Before running the application, in the project properties set MDI form as the start-up form. Save and run the application. Following output will be displayed:

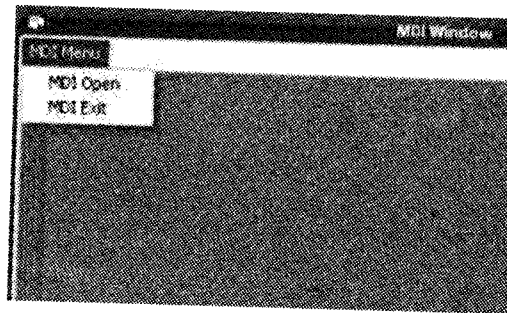


Figure 3.15

As soon as you click MDI Open you can notice that the main menu of the MDI Form is replaced with the Menu of the Child Form.

4.7 Creating Properties in a Form

You enclose a property definition between a property statement and an End Property statement. Within this definition you define a Get procedure, a Set procedure, or both. All the property's code lies within these procedures.

The Get procedure retrieves the property's value, and the Set procedure stores a value. If you want the property to have read/write access, you must define both procedures. For a read-only property, you define only Get, and for a write-only property, you define only Set.

To create a property

1. Outside any property or procedure, use a Property Statement, followed by an End Property statement.

2. If the property takes parameters, follow the Property keyword with the name of the procedure, then the parameter list in parentheses.
3. Follow the parentheses with an As clause to specify the data type of the property's value. You must specify the data type even for a write-only property.
4. Add Get and Set procedures, as appropriate.

For example,

```
Dim firstName, lastName As String
Property fullName() As String
    Get
        If lastName = "" Then
            Return firstName
        Else
            Return firstName & " " & lastName
        End If
    End Get
    Set (ByVal Value As String)
        Dim space As Integer = Value.IndexOf(" ")
        If space < 0 Then
            firstName = Value
            lastName = ""
        Else
            firstName = Value.Substring(0, space)
            lastName = Value.Substring(space + 1)
        End If
    End Set
End Property
```

4.8 Creating a Method in a form

We create methods in classes. A method created in a Class is nothing more than a Function or a Sub. The process for creating method is similar to the process of creating function or sub. A method is created using following steps:

Step 1: Create a class by clicking on Project → Add Class.

Step 2: As a method of class, add sub or function. We create a sub routine using the keyword 'sub' and function using the keyword 'function'. *For example,* let us add a subroutine in class.

```
Public class Dummy
    Sub foo()
        Dim strFullName As String
```

```

    strFullName = "Mr. Abc"
End Sub
End Class

```

Solved Programs

1

Apr.2010 - 4M

1. Write a program to accept details of teachers from user and store those details into the database (Don't use standard control). Teachers having fields Tno, Tname, Salary, Dateofjoining.

Solution

```

Private Sub cmdSave_Click()
    ADODB.AddNew
    ADODB.Recordset.Fields("TNo") = txtno.Text
    ADODB.Recordset.Fields("Tname") = txtname.Text
    ADODB.Recordset.Fields("Salary") = txtsal.Text
    ADODB.Recordset.Fields("doj") = txtdoj.Text
    ADODB.Recordset.Update
End Sub

Private Sub Form_Load()
Dim cnn As New ADODB.Connection
Dim rs As New ADODB.Recordset
Dim strSQL As String
Set cnn = New ADODB.Connection
cnn.Open "Microsoft.Jet.OLEDB.3.51;Persist Security
Info=False;Data Source=C:\Documents and Settings\Administrator
\My Documents\TeacherDB.mdb"
rs.CursorType = adOpenDynamic
rs.CursorLocation = adUseClient
rs.LockType = adLockOptimistic
rs.Open strSQL, cnn, , , adCmdText
End Sub

```

1

Oct.11 - 4M

2. Write a program to transfer the selected elements from the list 1 to list 2.

Solution

```

Private Sub cmd1_Click()
List2.AddItem(List1.List(List1.ListIndex))
List1.RemoveItem(List1.ListIndex)
End Sub

Private Sub cmd2_Click()
List1.AddItem(List2.List(List2.ListIndex))

```

2. If the property takes parameters, follow the Property keyword with the name of the procedure, then the parameter list in parentheses.
3. Follow the parentheses with an As clause to specify the data type of the property's value. You must specify the data type even for a write-only property.
4. Add Get and Set procedures, as appropriate.

For example,

```
Dim firstName, lastName As String
Property fullName() As String
    Get
        If lastName = "" Then
            Return firstName
        Else
            Return firstName & " " & lastName
        End If
    End Get
    Set(ByVal Value As String)
        Dim space As Integer = Value.IndexOf(" ")
        If space < 0 Then
            firstName = Value
            lastName = ""
        Else
            firstName = Value.Substring(0, space)
            lastName = Value.Substring(space + 1)
        End If
    End Set
End Property
```

4.8 Creating a Method in a form

We create methods in classes. A method created in a Class is nothing more than a Function or a Sub. The process for creating method is similar to the process of creating function or sub. A method is created using following steps:

Step 1: Create a class by clicking on Project → Add Class.

Step 2: As a method of class, add sub or function. We create a sub routine using the keyword 'sub' and function using the keyword 'function'. *For example,* let us add a subroutine in class.

```
Public class Dummy
    Sub foo()
        Dim strFullName As String
```

```

    strFullName = "Mr. Abc"
End Sub
End Class

```

Solved Programs

1

Apr.2010 - 4M

1. Write a program to accept details of teachers from user and store those details into the database (Don't use standard control). Teachers having fields Tno, Tname, Salary, Dateofjoining.

Solution

```

Private Sub cmdSave_Click()
    ADODB.AddNew
    ADODB.Recordset.Fields("TNo") = txtno.Text
    ADODB.Recordset.Fields("Tname") = txtname.Text
    ADODB.Recordset.Fields("Salary") = txtsal.Text
    ADODB.Recordset.Fields("doj") = txtdoj.Text
    ADODB.Recordset.Update
End Sub

Private Sub Form_Load()
    Dim cnn As New ADODB.Connection
    Dim rs As New ADODB.Recordset
    Dim strSQL As String
    Set cnn = New ADODB.Connection
    cnn.Open "Microsoft.Jet.OLEDB.3.51;Persist Security
    Info=False;Data Source=C:\Documents and Settings\Administrator
    \My Documents\TeacherDB.mdb"
    rs.CursorType = adOpenDynamic
    rs.CursorLocation = adUseClient
    rs.LockType = adLockOptimistic
    rs.Open strSQL, cnn, , , adCmdText
End Sub

```

1

Oct.11 - 4M

2. Write a program to transfer the selected elements from the list 1 to list 2.

Solution

```

Private Sub cmd1_Click()
    List2.AddItem(List1.List(List1.ListIndex))
    List1.RemoveItem(List1.ListIndex)
End Sub

Private Sub cmd2_Click()
    List1.AddItem(List2.List(List2.ListIndex))

```

```

List2.RemoveItem(List2.ListIndex)
End Sub
Private Sub cmd3_Click()
Dim i As Integer
If List1.ListIndex = -1 Then Exit Sub
For i = List1.ListCount-1 To 0 Step -1
If List1.Selected(i)= True Then

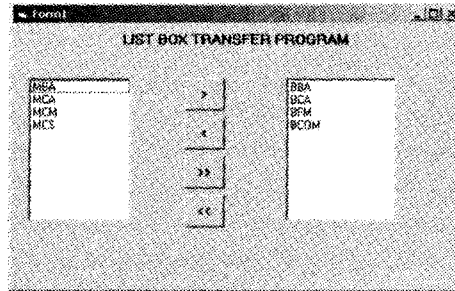
List2.AddItem List1.List(i)
List1.RemoveItem i
End If

Next i
End Sub

Private Sub cmd4_Click()
Dim i As Integer
If List2.ListIndex=-1 Then Exit Sub

For i = List2.ListCount-1 To 0 Step -1
If List2.Selected(i)=True Then
List1.AddItem List2.List(i)
List2.RemoveItem i
End If
Next i
End Sub

```



Control Name	Property Name	Value
List1	Name	Lst1
	Multiselect	Simple
List2	Name	Lst2
	Multiselect	Simple
Command Button1	Name	Cmd1
Command Button2	Name	Cmd2
Command Button3	Name	Cmd3
Command Button4	Name	Cmd4



PU Questions

4 Marks

[Apr.2013 – 4M]

1. Write Short note on: Picture Box.

[Oct.2012 – 4M]

2. Write Short note on: Input Box

[Apr.2012 – 4M]

3. Write short notes: Option Button or Radio Button

[Apr.2012 – 4M]

4. Write short notes: Picture Box

[Oct.2011 – 4M]

5. State the difference between Combo Box and List box.

[Oct.2011 – 4M]

6. Write a program to transfer the selected elements from the list 1 to list 2.

[Apr.2010 – 4M]

7. Write a program to accept details of teachers from user and store those details into the database (Don't use standard control). Teachers having fields Tno, Tname, Salary, Date of joining.

16 Marks

[Apr.2013 – 16M]

1. Explain the following property settings:
 - a. Property used to Disable Label Control.
 - b. Property to set maximum number of characters to be input using textbox.
 - c. Property used to display a read only combo.
 - d. Property used to set Timer control.
 - e. Property used to set special Password character.
 - f. Property used to set Value of Check Boxes.
 - g. To resize image control.
 - h. Property used to count number of item in the listbox control.
 - i. Property used to place a picture on a command button.
 - j. Property to set job order for the control of the form.

2. Explain the following property settings: (Any Eight) **[Oct.2012 – 16M]**
- a. Property used to make the background of the label transparent.
 - b. Property used to display information on the command button.
 - c. Property used to display information in text box control.
 - d. Property used to set the value of check boxes.
 - e. Property used to resize picture dynamically to fit the dimensions of the picture box control.
 - f. Property to sort the items in a combo box.
 - g. Property used to remove items from and list box.
 - h. Property used to draw circle from the shape control.
3. Explain the following property settings: **[Apr.2012 – 16M]**
- a. Property to Set Path Property of DIR List Box.
 - b. To change the title of a Form.
 - c. Property used to Set Value of Check Box.
 - d. Property used to Set Special Password Character of Text Box.
4. Explain the following property settings: (Any Eight) **[Apr.2011 – 16M]**
- a. Property to display picture to run time.
 - b. Property to display Value of check box.
 - c. Property to Items alphabetically in the list.
 - d. Property to display text in multiple line.
 - e. Property to add a horizontal scroll bar to a text box.
 - f. Property to set font style using common dialog box.
 - g. Property to disable textbox control.
 - h. Property to set record set in the data control.
 - i. Property to set caption of a label.
 - j. Property to hide image at run time.

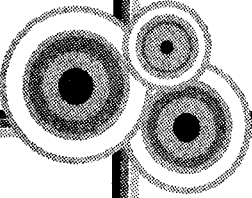
[Apr.2011 – 16M]

5. Explain the following property settings: (Any Eight)
- a. Property to place a picture on the command button.
 - b. Property to set maximum number of characters to be input using textbox.
 - c. Property to set control items alphabetically in a combo box'.
 - d. Property used to set values of check boxes?
 - e. Property used to disable label control?
 - f. Property used to remove an item from a list?
 - g. Property to maximize form at run time.
 - h. Property to set font size using common dialog box?
 - i. Property to set path property of DIR List box?
 - j. Property to set tab order for the control on the form?

[Apr.2011 – 16M]

6. Explain the following property settings:
- a. Property used to enable TextBox Control.
 - b. Property used to display all *.doc extension file in Filelistbox.
 - c. Property used to resize picture to fit in the Image Control.
 - d. Property to set tab order for the control of the form.
 - e. Property used to display a read only combo box.
 - f. Property used to set timer control.
 - g. Property used to display text on label control.
 - h. Property used to set special password character of textbox control.
 - i. Property used to count number of item in the listbox control.
 - j. Property used to place a picture on a command button.

WORKING WITH ACTIVEX CONTROLS AND MENUS



1. Introduction

An ActiveX is a miniature program that you can plug into your VB program to give them added features without writing much code. ActiveX controls can help you write a program quickly and easily. VB has many ActiveX controls available on toolbox. In spite of that you can have tons of ActiveX controls which you can add to your project but before that you have to add it on toolbox.

To add an ActiveX control on the project follow these steps:

1. Select project Menu → Components or Press Ctrl+T. Component dialog box will appear from where you can add different controls. (Refer *Figure 4.1*)

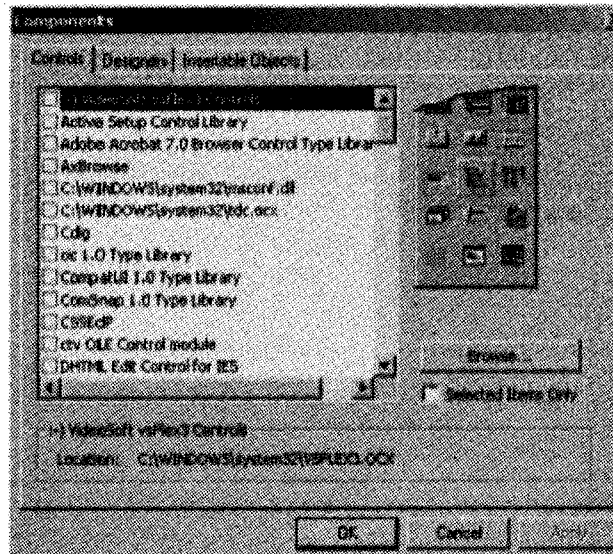


Figure 4.1: Component Dialog box

2. Check the checkbox of the control that you want to add and then click on OK button. Now you can see the control on the toolbox. Drag the control and draw it on the form. Along with all these control you can create your own ActiveX control in VB.

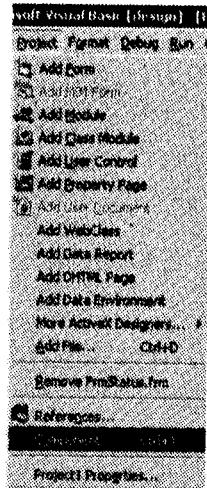
2. Creating Status Bar for your Program

Status bar is an information area generally found at the bottom of the screen. Status bars are used to display status messages at the bottom of the form. It also contains the information about date, time, page number, current status of application like page no, caps on/off, num on/off. It can also indicate the progress of any running application. In case of web browser, status bars are used to indicate progress of the loading of web pages into the browser window. There are two kinds of status bars: simple status bars and status bars that display a panel. Simple status bars display a single message on the status bar and a status bar with panel can display multiple messages. Upto 16 panel objects can be contained in the collection. You can display an image and text on the panel.

You can build status bar using the status bar ActiveX control. This control has many properties that help you build multiple panels on the status bar. You can display different information on all the panels and control all the panels through program.

Status bar control is not default control on the tool bar. You have to add it through components. Status bar control is the part of Microsoft common controls. Follow the steps to add status bar on the form

1. Select project Menu → Components or Press Ctrl+T. Component dialog box will appear from where you can add different controls. (Refer Figure 4.2)



1
 Oct.2012 – 4M
 Write Short note on:
 Status Bar

Figure 4.2

2. Select the Microsoft Common Controls 5.0 (SP2) check box, and click on the OK button.

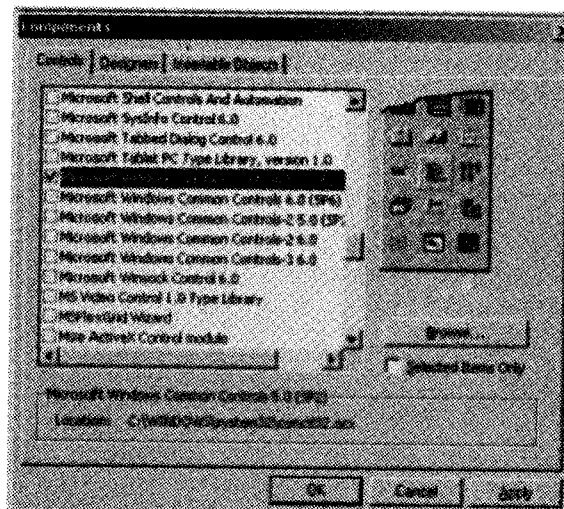


Figure 4.3

3. Now you can see the controls added on the toolbox. One of them is status bar control. Like other controls you can select and display it on the form. (Refer *Figure 4.4*).

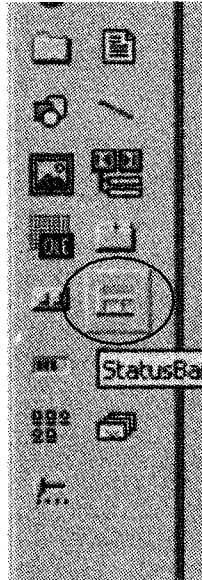


Figure 4.4

4. After displaying the status bar control on the form your form should look like this. (*Figure 4.5*)

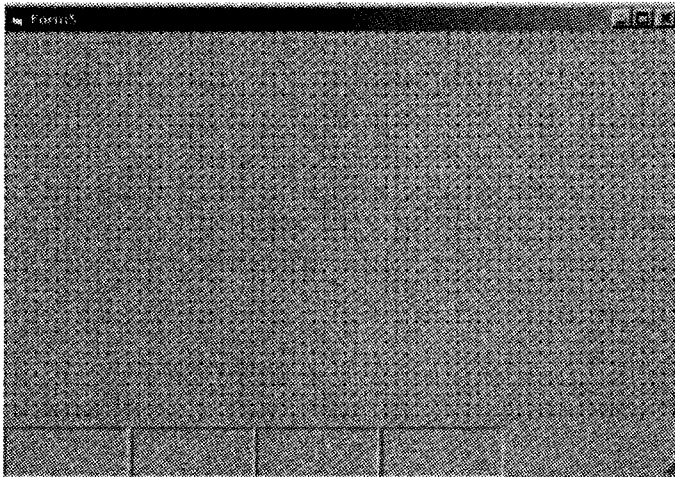


Figure 4.5: Form showing Status Bar Control at the bottom

5. You can add the panels by using the panels' page. Right click on the status bar control or select custom property from the property window. VB will open the Property page where you can see different tabs. (Figure 4.6). You add panels by clicking the Add button found in the editor. While adding panels you can set the Text to be displayed for each panel, an icon, tooltip, width for each panel you add.

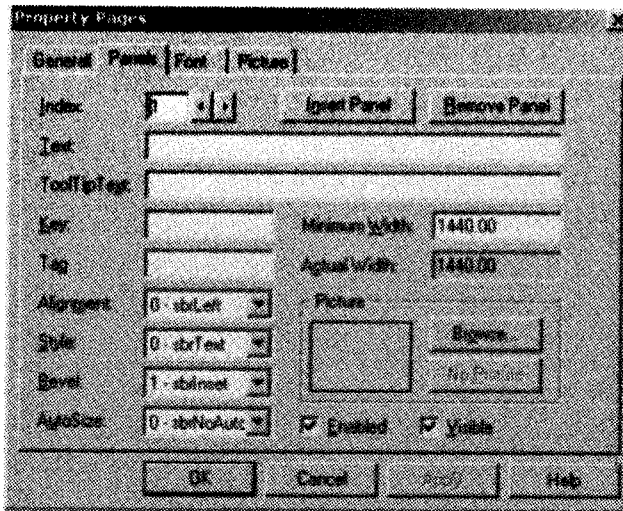


Figure 4.6: Property page

6. To add panels to status bar in code we use the *StatusBar.Panel.Add* method and *StatusBar.Panels.Remove*, *StatusBar.Panels.RemoveAt* to remove the panels. To access text in each panel you use the text property of StatusBarPanel as: *StatusBarPanels (0).Text="I am panel one"*.

To handle status bar panel clicks you use the PanelClick event as shown in the code below. To work with this code, add a status bar control to the form, open its properties window, select the Panels property and add three status bar panels. For StatusBarPanel1 set the text "Date:" for StatusBarPanel2 set the text "Time:".

```
StatusBar1.Panels (1).Text = "Date" & Date
StatusBar1.Panels (2).Text = "Time: " & Time
```

The form in design view should look like the image below:

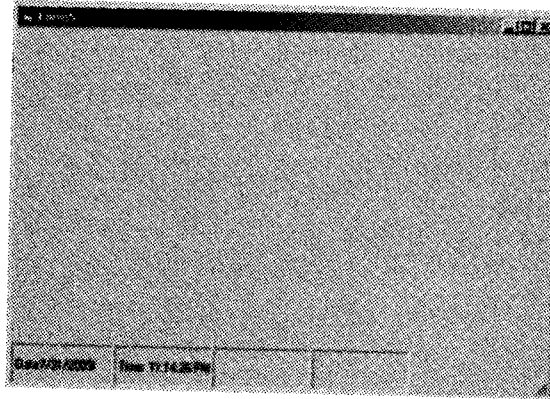


Figure 4.7: Date and Time on status bar

3. Working with Progress Bar

A progress bar is used to inform the user about time consumed in processing. It shows the user the status of processing, and illustrates that the application has not gone into a “not responding” state. A large number of applications, such as setups, database-driven applications, and file transfer tools, show progress bars at the time of processing. Progress bar gives the user some visual feedback on what is happening during a time consuming operation. It shows color bar that grows in the control to show how the operation is proceeding, usually 0 to 100 percent. At run time the progress bar value property determines how much of the control has been filled. The min and max properties set the limits of the control.

3

Apr. 13, 12, 10 – 4M
Write a short note on:
Progress Bar

Visual Basic has the ActiveX progress bar control. It can be included on forms through the Microsoft Windows Common Controls component.

It has various properties and methods, but the most commonly used properties are *Min*, *Max* and *Value*.

Min is used to denote the lowest value a progress bar can take. This is the initial starting position of the progress bar.

Max is used to depict the maximum value that can be assigned to the progress bar. A progress bar can't take a value greater than the one specified in the Max property.

Value property can be used to retrieve a value which is in between min and max value to the progress bar, so that the *bar* in the progress bar can increase appropriately.

For example, this example explains the use of progress bar. In this example the form's background color changes as the progress bar value increases. The value of progress bar starts with 0 and end with 100. It starts processing on the timer. The value increases by 5 after 1000 milliseconds. Once the value of progress bar reaches to 100 the timer stops.

Follow the steps given below:

Step 1: Now you have the progress bar control on your tool box. In the previous section we have added Microsoft Common controls for Status bar control. Progress Bar is also one of the components of Microsoft Common controls. (Refer Figure 4.8).

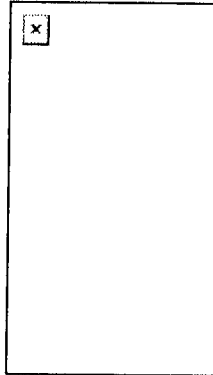


Figure 4.8: Progress bar control encircled

Step 2: Select the control shown and draw on the form. As shown in the Figure 4.9. Along with progress bar draw timer and one label. Set the following properties of all the controls as follows:

Timer1: Interval = 1000

Label1: Font: 14 Bold

ProgressBar1: Min = 0, Max = 100

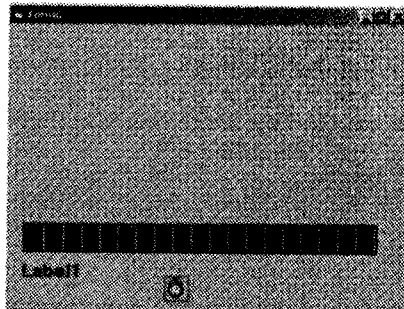


Figure 4.9: Form with Progress Bar

Step 3: Write the following code on the Timer1_Timer event.

```
Private Sub Timer1_Timer ()  
    ProgressBar1.Value = ProgressBar1.Value + 5  
    Form6.BackColor =  
        RGB (Rnd * ProgressBar1.Value, Rnd *  
            ProgressBar1.Value, Rnd * ProgressBar1.Value)  
    Label1.Caption = ProgressBar1.Value & "%"  
    If ProgressBar1.Value = 100 Then  
        Timer1.Enabled = False  
    End If  
End Sub
```

Step 4: Save the code and run the form. Observe the changes in the color of the form while progress bars value increases. The label shows the value of the progress bar. The timer event will be triggered after 1000 milliseconds. After every 1000 milliseconds the value of progress bar will be increased by 5 which you can observe in label. The form color changes according to that value.

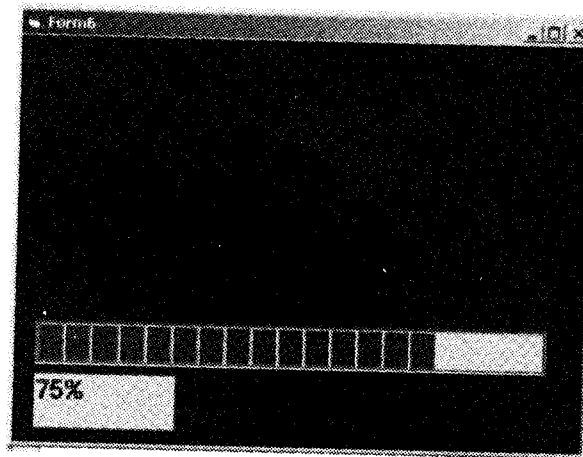


Figure 4.10: Run Form showing status of progress Bar

4. Working with Toolbar and Setting up the Image List Controls

Like status bar and progress bar we can also have our own tool bar in our application. This can look identical to VB's standard toolbar. Along with menu if you add tool bar to your application it gives a professional look to your application. Toolbars are seen in almost every application, and gives users quick access to regularly used features like open, new, close, save etc. Microsoft offers a toolbar control in its Common Controls which makes it very easy to add a toolbar to your own application. Creating a toolbar in Visual Basic is a multi step process, and all the steps are discussed in detail in this section.

To add the Toolbar control to your VB project, click Project → Components menu, and check the box next to "Microsoft Windows Common Controls xx" where x is the version of Visual Basic you are using. This we have already done in the previous sections. The Toolbar control is as shown in the *figure 4.11*.

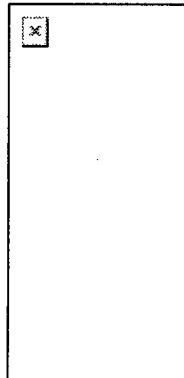


Figure 4.11: Toolbar Control on the toolbox

A toolbar control contains a collection of button objects used to create the toolbar. These button objects correspond to the application in your project. You can display text as well as images on these buttons. Text can be displayed using caption property. And if you want images to appear on these buttons; first you need to add those images in image list control.

The ImageList control is used to store a number of images used by other controls at runtime. This control maintains a series of bitmaps in its memory so that any control can access it quickly at runtime. To use ImageList control in a project, add the windows common Controls – 2 component. Image list control is as shown in the *Figure 4.12*. If you want to set the images at design time, the associated ImageList control must be on the same form as the toolbar control. After the images are stored in an ImageList control, you set the ImageList property of the toolbar control to be the name of the ImageList control.

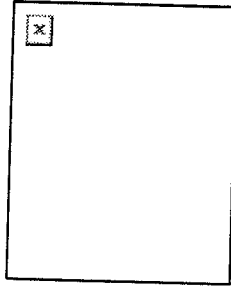


Figure 4.12: Image control on toolbox

Designing toolbar includes many steps. Following steps will guide you to understand the procedure to design tool bar:

Step 1: First of all we need Toolbar and ImageList control added on your toolbox. For this Select Project → Component option from VB main menu. (Figure 4.13)

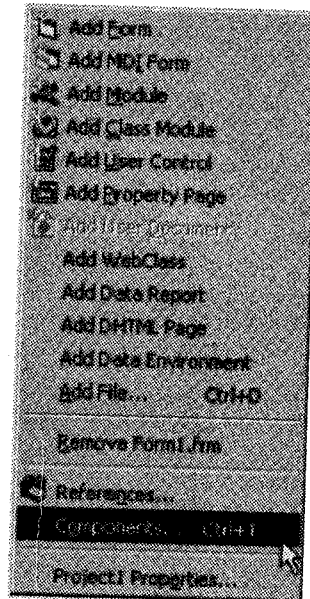


Figure 4.13: Project → Component menu

It will open component dialog box. Search for “Microsoft Windows Common Controls“ you will get both the controls enclosed in this. Select the Check box as shown in the Figure 4.14. Click on OK button. Now you can see many controls added on your tool box

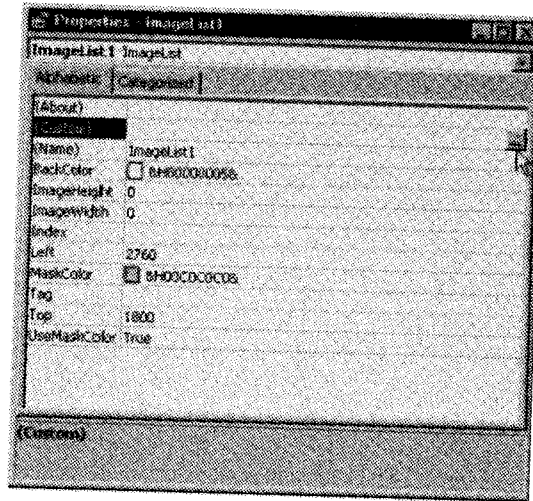


Figure 4.16

This will open the Property Page for the ImageList control as shown in the *Figure 4.17*.

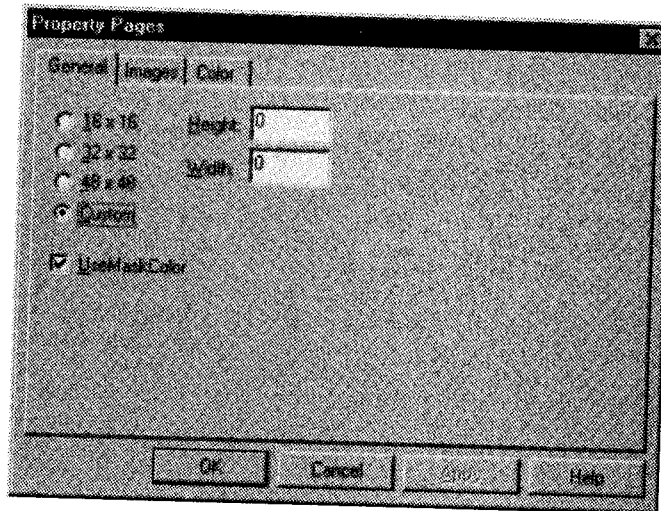


Figure 4.17

By default General tab is selected. It shows the Information about the size of the buttons that you will see on the Toolbar. Select 32 × 32 option button.

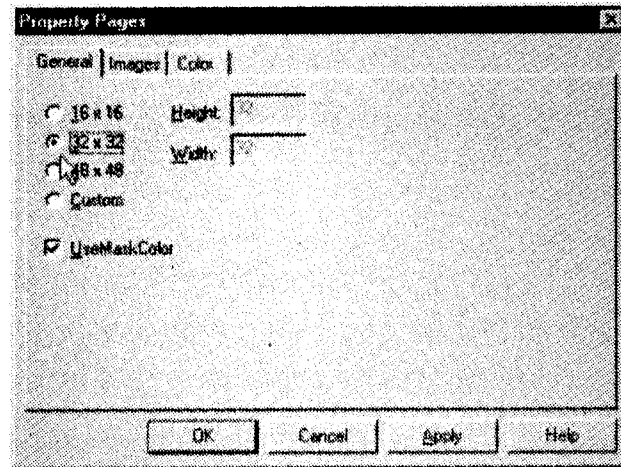


Figure 4.18

This is enough large toolbar icons. Now we will start adding images to the ImageList control. We do that by selecting the Images tab...

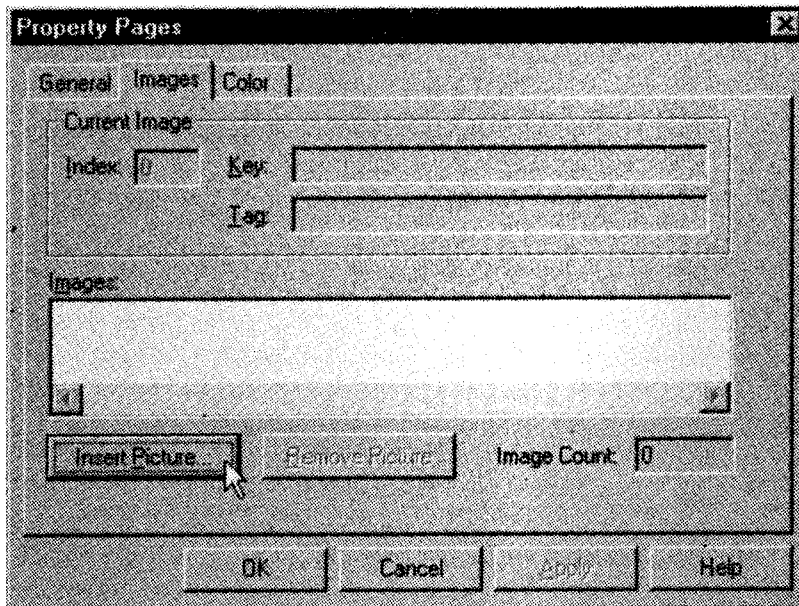


Figure 4.19

Now select the *Insert Picture* button it will ask us for an image to add to the ImageList control. You can select any of the image.

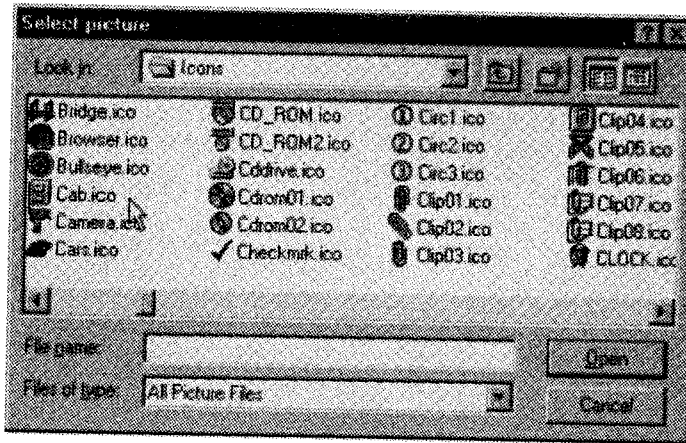


Figure 4.20

As soon as you open the image, that image is added into the list of the ImageList control. See the following screen. You can see the Cabinet image in the list. As this is first image its index value is 1 and an Image Count Property also 1.

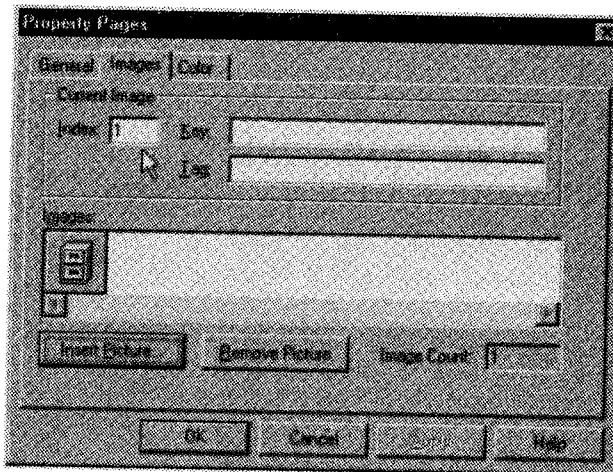


Figure 4.21

Each image in the ImageList Control can be referred to by its Index value. The toolbar control will use the Index value to refer to each Image.

Use the same procedure to add images to the ImageList Control. You can see Remove Picture on the property page. We can remove images if we wish by selecting this button. In this example 3 more images have been added to the ImageList control. Your property page should look like the following screen. These many images are enough for now.

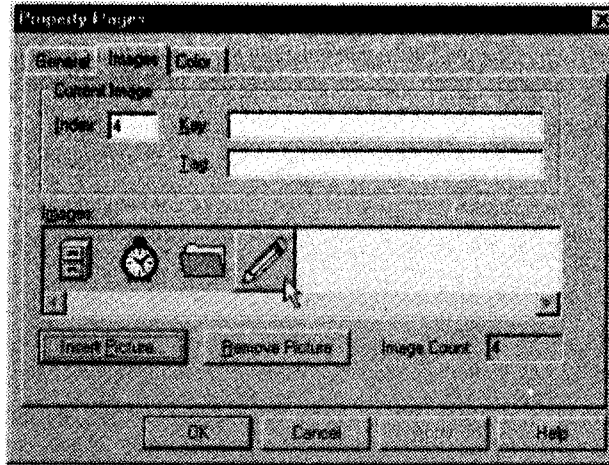


Figure 4.22

Step 4: Now we have added enough images to the ImageList control. Now set the properties in the Toolbar control. Add buttons that will appear on the toolbar, and set the images which we have right now added to ImageList Control.

Hence, open Toolbar property page. For this select Custom from property window or right click on the Toolbar Control on the form.

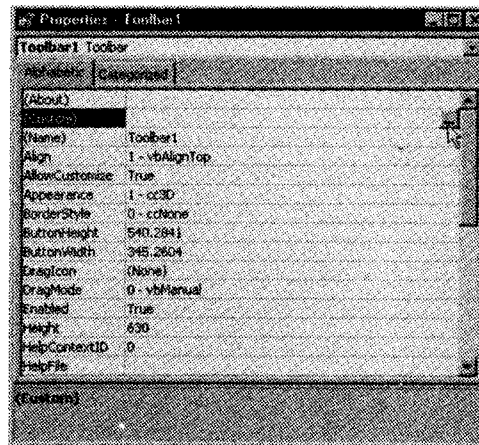


Figure 4.23

This will open the property page for the toolbar control. The screen should look as follows:

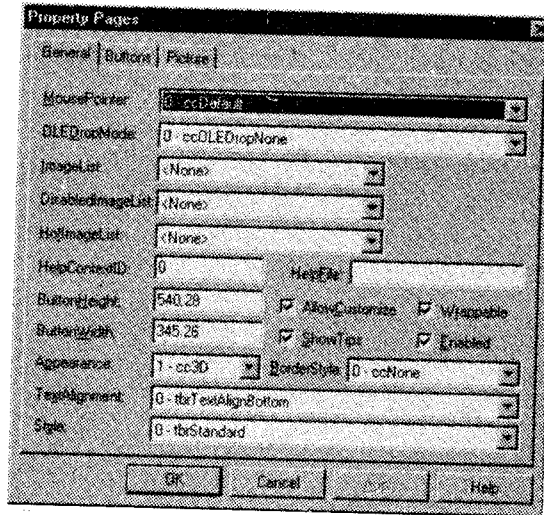


Figure 4.24

You can see toolbar property page is more complicated than ImageList property page. Here we will see how to add buttons to the toolbar, associate them with an image in the ImageList control. In General tab you can see third item i.e. ImageList dropdown list box. We need to tell the toolbar control where to find the images for the buttons that we're about to add. For this purpose click on the dropdown ListBox for the ImageList property and select the ImageList Control that contains our images. When you click on the ImageList dropdown list box it will display ImageList1 control name. You just select it. This means we've associated our Toolbar with an ImageList control.

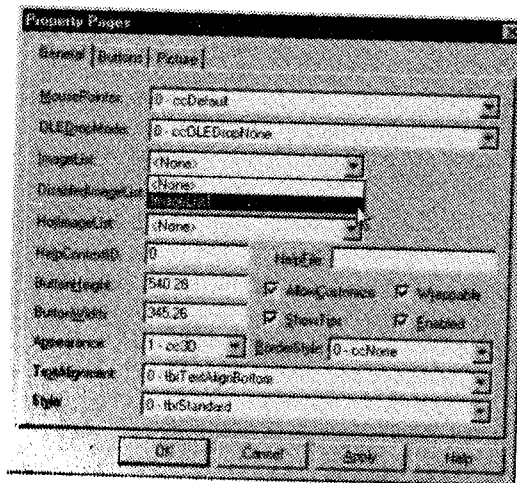


Figure 4.25

The next step is to start adding buttons to the toolbar. We do that by selecting the Buttons tab.

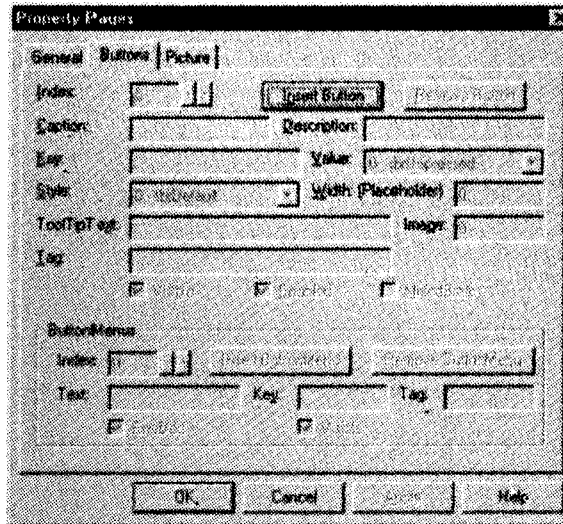


Figure 4.26

Through this tab you can add as many as buttons you want to your toolbar. Initially the Index property for the Buttons tab is dimmed that means toolbar has no buttons. To add buttons to the toolbar click on 'Insert Button'.

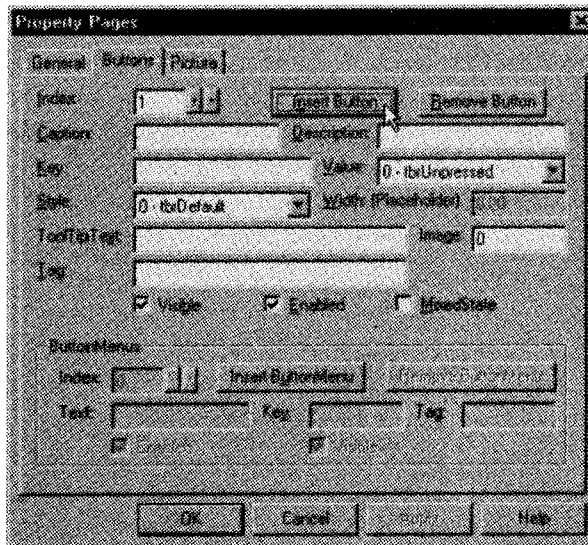


Figure 4.27

Once we click on 'Insert Button', the index property of the Buttons becomes undimmed. It becomes 1, indicating we are editing the Property values for the first button on the toolbar. We can now associate an image in the ImageList control with this button. To do that, specify the Index value of the appropriate image in the ImageList in the Image Property of the Buttons tab. As shown in the following screen.

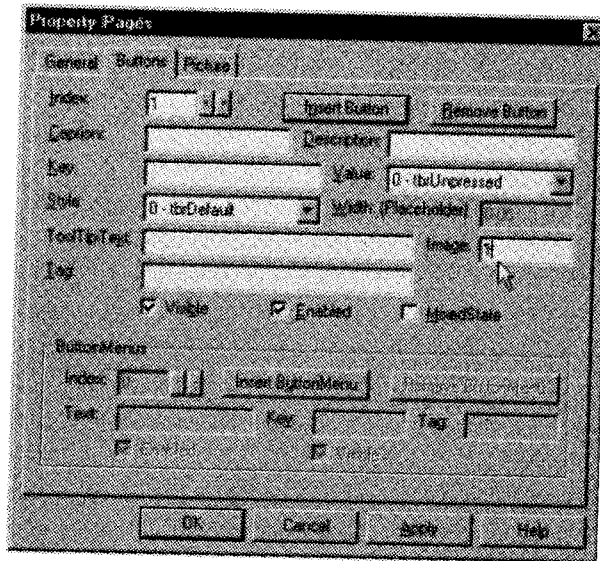


Figure 4.28

By specifying a value of 1 for the Image Property of the first button, we are telling Visual Basic to display the Cabinet icon as the image for the first button. If we now click on the Apply button, we'll see that our toolbar now looks like this...

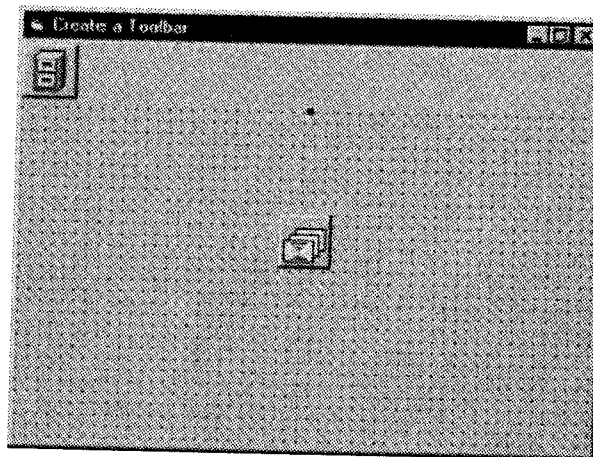


Figure 4.29

You can provide additional help to the tool bar by displaying ToolTip Text for the buttons we add to the Toolbar by writing the name of the button in the ToolTipText Property of the Buttons tab.

Same way you can keep adding button to the tool bar i.e. press “Insert Button” and adding index number to the “Image” textbox. Write some value to the Tooltip text textbox.

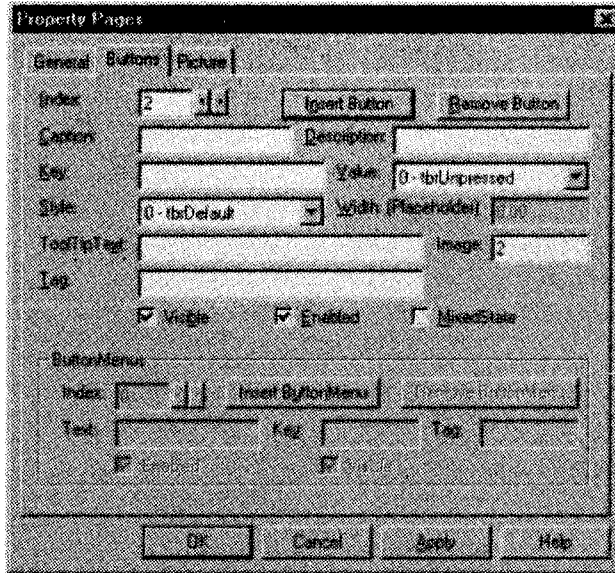


Figure 4.30

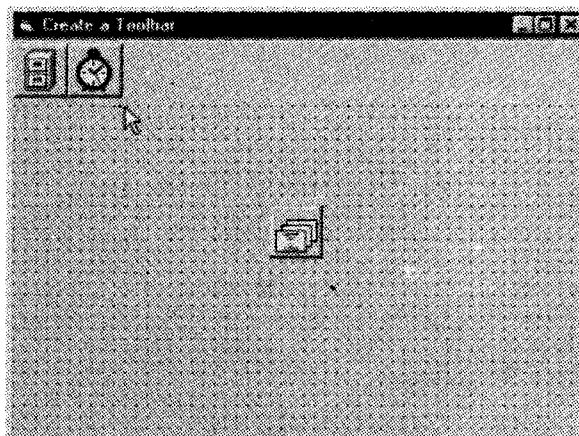


Figure 4.31

You can insert a 'gap' or separator between groups of buttons. To do that, click on 'Insert Button', but this time instead of specifying a value for the Image Property, leave it at 0, and specify a value of **3-tbrSeparator** for the Style Property.

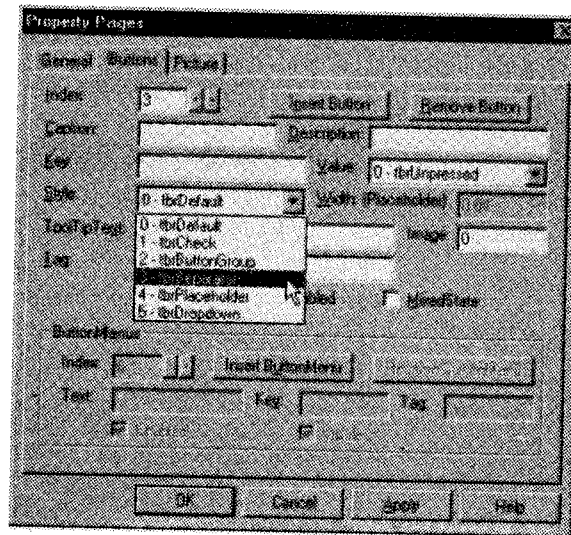


Figure 4.32

After inserting separator add one more button; specify the Image and Tooltip text and now you will be able to see the separator or gap between the buttons.

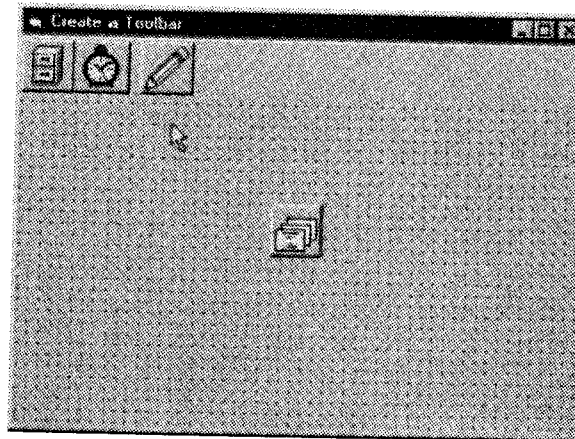


Figure 4.33

At this point, we have a Toolbar that looks beautiful as in *Figure 4.33*, but its not working. If you run this form now you can click the button on the tool bar but they wouldn't work. We need to write some code to that buttons if we want these buttons should trigger some action.

One problem here is that the individual buttons on the Toolbar do not have their own events. There are only events associated with the Toolbar control itself. So, if we want to write code to perform some action when a particular button on the toolbar is clicked, we first need to distinguish which button has been clicked. If you double click on the toolbar control it will open the ButtonClick Event Procedure as shown in the *Figure 4.34*.

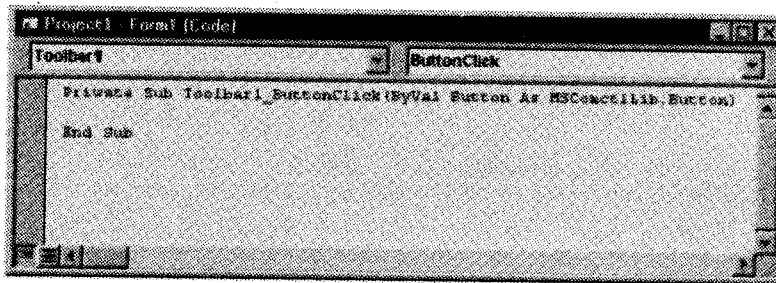


Figure 4.34

The ButtonClick Event Procedure of the toolbar is passed an Object Variable representing the button on the Toolbar that has been clicked. We can determine the Index value of that button to determine which button has been clicked. Knowing that, we can write code that is triggered when a particular button is clicked.

Suppose we want to write code for our toolbar. We have now 4 buttons out of which button number 3 is separator. Therefore it does not require any code. So we can write code for the button 1, 2 and 4. Check the following code here we have written code for the button 1 and 2. If you click Button 1 it will open a message box window. And button 2 will close the application. The same way you can open another form or can perform any valid VB operations through this buttons.

```
Private Sub Toolbar1_ButtonClick (ByVal Button As  
ComctlLib.Button)  
    If Button.Index = 1 Then  
        MsgBox "Very Good!!! You did very good job."  
    ElseIf Button.Index = 2 Then  
        Unload Me  
    End  
End If  
End Sub
```

5. Study of Different Dialog Boxes

We all are familiar with windows common dialog boxes like Open, Save, Print, font dialog boxes. Dialog boxes are used to create professional applications that have the same menu and dialog box structure as most windows application. In Visual Basic 6.0 we have the facility to build the common dialog box in our application using the CommonDialog ActiveX control. You can save lot of programming time by using Common Dialog control, as well as provide a standard Windows look to certain parts of your program.

In this session we will discuss how to add common dialog control in our application.

To add this control select Project → Components menu → Microsoft Common Dialog Control 6.0 on the Controls tab in the Components window as shown in the following figure. Then, the control icon will appear in your toolbox and you can add it as a form like any other control. At run time, it's invisible until you need it.

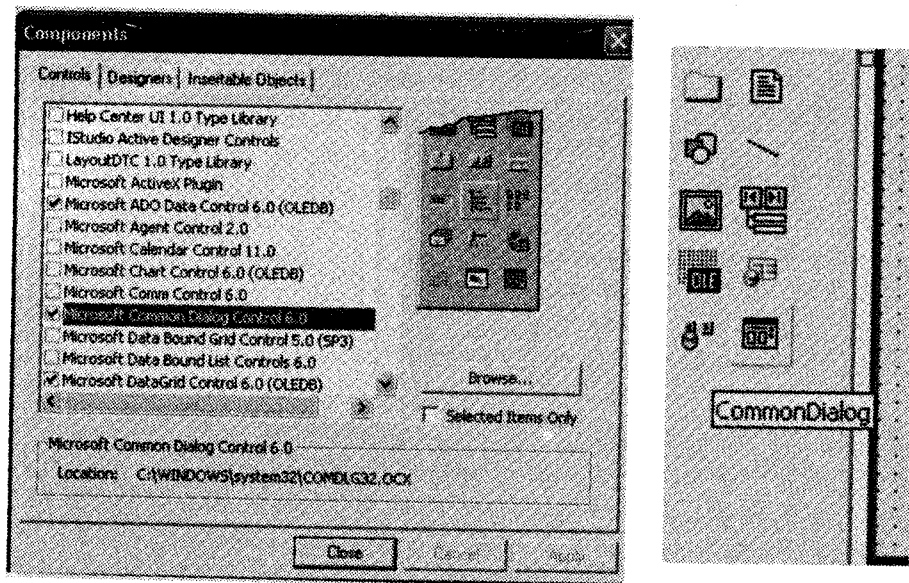


Figure 4.35

There are six types of dialog box with this control. You choose the dialog box you need by the method you call on the control.

The lists of all six types of dialog boxes are:

1. **ShowOpen:** Shows a File Open dialog box.
2. **ShowSave:** Shows a File Save dialog box.
3. **ShowColor:** Shows a Select Color dialog box.
4. **ShowFont:** Shows a Font Selection dialog box.
5. **ShowPrinter:** Shows a Print/Print Options dialog box.
6. **ShowHelp:** Shows the help dialog box.

To display any of the common dialog boxes you must call it using showXXX method. *For example*, ShowColor to open color dialog box, ShowFont for Font dialog box. Each dialog box has a number of properties that you can use. Most of the properties are optional. Some are common to all dialog boxes.

Some of the properties of dialog box are discussed as follows:

CancelError	Each dialog box has a cancel button, which should signal to your application the user's intention to cancel the current operation.
DialogTitle	This property sets the string displayed in the title bar of the dialog box.
Flags	Used to adjust the function of each common dialog box. The value of the flag will vary depending on the specific dialog box being opened. All the flags of all the dialog box control are discussed in the following tables.
Min and Max	This property is used to the Print and Font dialog box.
Action	This property determines which dialog to display. Or you can use Showxxx method to display specific dialog box.

Common dialog control flags

1. File Open/Save Dialog Box Flags

Constant	Description
cdIOFNAllowMultiselect	Specifies that the File Name list box allows multiple selections. The user can select more than one file at run time by pressing the SHIFT key and using the UP ARROW and DOWN ARROW keys to select the desired files. When this is done, the FileName property returns a string containing the names of all selected files. The names in the string are delimited by spaces.
cdIOFNCreatePrompt	Specifies that the dialog box prompts the user to create a file that doesn't currently exist. This flag automatically sets the cdIOFNPathMustExist and cdIOFNFileMustExist flags.
cdIOFNExplorer	Use the Explorer-like Open A File dialog box template. Common dialogs that use this flag do not work under Windows NT using the Windows 95 shell.
CdIOFNExtensionDifferent	Indicates that the extension of the returned filename is different from the extension specified by the DefaultExt property. This flag isn't set if the DefaultExt property is Null, if the extensions match, or if the file has no extension. This flag value can be checked upon closing the dialog box.

cdIOFNFileMustExist	Specifies that the user can enter only names of existing files in the File Name text box. If this flag is set and the user enters an invalid filename, a warning is displayed. This flag automatically sets the cdIOFNPathMustExist flag.
cdIOFNHelpButton	Causes the dialog box to display the Help button.
cdIOFNHideReadOnly	Hides the Read Only check box.
cdIOFNLongNames	Use long filenames.
cdIOFNNoChangeDir	Forces the dialog box to set the current directory to what it was when the dialog box was opened.
CdIOFNNoDereferenceLinks	Do not dereference shell links (also known as shortcuts). By default, choosing a shell link causes it to be dereference by the shell.
cdIOFNNoLongNames	Do not use long file names.
CdIOFNNoReadOnlyReturn	Specifies that the returned file won't have the Read Only attribute set and won't be in a write-protected directory.
cdIOFNNoValidate	Specifies that the common dialog box allows invalid characters in the returned filename.
cdIOFNOverwritePrompt	Causes the Save As dialog box to generate a message box if the selected file already exists. The user must confirm whether to overwrite the file.
cdIOFNPathMustExist	Specifies that the user can enter only valid paths. If this flag is set and the user enters an invalid path, a warning message is displayed.
cdIOFNReadOnly	Causes the Read Only check box to be initially checked when the dialog box is created. This flag also indicates the state of the Read Only check box when the dialog box is closed.
CdIOFNShareAware	Specifies that sharing violation errors will be ignored.

2. Color Dialog Box Flags

Constant	Description
cdICCFullOpen	Entire dialog box is displayed, including the Define Custom Colors section.
cdICCShowHelp	Causes the dialog box to display a Help button.
cdICCPreventFullOpen	Disables the Define Custom Colors command button and prevents the user from defining custom colors.
cdICCRGBInit	Sets the initial color value for the dialog box.

3. Fonts Dialog Box Flags

Constant	Description
cdICFANSIOOnly	Specifies that the dialog box allows only a selection of the fonts that use the Windows character set. If this flag is set, the user won't be able to select a font that contains only symbols.
cdICFApply	Enables the Apply button on the dialog box.
cdICFBoth	Causes the dialog box to list the available printer and screen fonts. The hDC

	property identifies the device context associated with the printer.
cdICFEffects	Specifies that the dialog box enables strikethrough, underline, and color effects.
cdICFFixedPitchOnly	Specifies that the dialog box selects only fixed-pitch fonts.
cdICFForceFontExist	Specifies that an error message box is displayed if the user attempts to select a font or style that doesn't exist.
cdICFHelpButton	Causes the dialog box to display a Help button.
cdICFLimitSize	Specifies that the dialog box selects only font sizes within the range specified by the Min and Max properties.
cdICFNoFaceSel	No font name selected.
cdICFNoSimulations	Specifies that the dialog box doesn't allow Graphic Device Interface (GDI) font simulations.
cdICFNoSizeSel	No font size selected.
cdICFNoStyleSel	No style was selected.
cdICFNoVectorFonts	Specifies that the dialog box doesn't allow vector-font selections.
cdICFPrinterFonts	Causes the dialog box to list only the fonts supported by the printer, specified by the hDC property.
cdICFScalableOnly	Specifies that the dialog box allows only the selection of fonts that can be scaled.
cdICFScreenFonts	Causes the dialog box to list only the screen fonts supported by the system.
cdICFTTOnly	Specifies that the dialog box allows only the selection of TrueType fonts.
cdICFWYSIWYG	Specifies that the dialog box allows only the selection of fonts that are available on both the printer and on screen. If this flag is set, the cdICFBoth and cdICFScalableOnly flags should also be set.

4. Printer Dialog Box Flags

Constant	Description
cdIPDAIIPages	Returns or sets the state of the All Pages option button.
cdIPDCollate	Returns or sets the state of the Collate check box.
cdIPDDisablePrintToFile	Disables the Print To File check box.
cdIPDHelpButton	Causes the dialog box to display the Help button.
cdIPDHidePrintToFile	Hides the Print To File check box.
cdIPDNoPageNums	Disables the Pages option button and the associated edit control.
cdIPDNoSelection	Disables the Selection option button.
cdIPDNoWarning	Prevents a warning message from being displayed when there is no default printer.
cdIPDPPageNums	Returns or sets the state of the Pages option button.

cdIPDPrintSetup	Causes the system to display the Print Setup dialog box rather than the Print dialog box.
cdIPDPrintToFile	Returns or sets the state of the Print To File check box.
cdIPDReturnDC	Returns a device context for the printer selection made in the dialog box. The device context is returned in the dialog box's hDC property.
cdIPDReturnDefault	Returns default printer name.
cdIPDReturnIC	Returns an information context for the printer selection made in the dialog box. An information context provides a fast way to get information about the device without creating a device context. The information context is returned in the dialog box's hDC property.
cdIPDSelection	Returns or sets the state of the Selection option button. If neither cdIPDPageNums nor cdIPDSelection is specified, the All option button is in the selected state.
cdIPDUseDevModeCopies	If a printer driver doesn't support multiple copies, setting this flag disables the copies edit control. If a driver does support multiple copies, setting this flag indicates that the dialog box stores the requested number of copies in the Copies property.

5. Help Constants

Constant	Description
cdlHelpCommandHelp	Displays Help for a particular command.
cdlHelpContents	Displays the contents topic in the current Help file.
cdlHelpContext	Displays Help for a particular topic.
cdlHelpContextPopup	Displays a topic identified by a context number.
cdlHelpForceFile	Creates a Help file that displays text in only one font.
cdlHelpHelpOnHelp	Displays Help for using the Help application itself.
cdlHelpIndex	Displays the index of the specified Help file.
cdlHelpKey	Displays Help for a particular keyword.
cdlHelpPartialKey	Calls the search engine in Windows Help.
cdlHelpQuit	Notifies the Help application that the specified Help file is no longer in use.
cdlHelpSetContents	Designates a specific topic as the contents topic.
cdlHelpSetIndex	Sets the current index for multi-index Help.

Example

The following example demonstrates all types of dialog boxes and its application. Follow the steps given below:

1. We have added Common dialog box control in the toolbox. Display it on the form. The form should look like following figure. The control is visible at design time but invisible at run time.

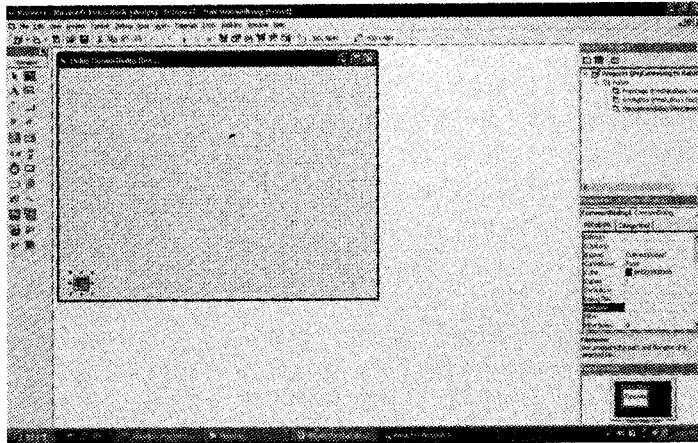


Figure 4.36

2. You need some interface to open the dialog box. We will use command button to open the dialog box. Design the form as shown below. This form contains six command buttons for six dialog boxes, one label and Common Dialog control.

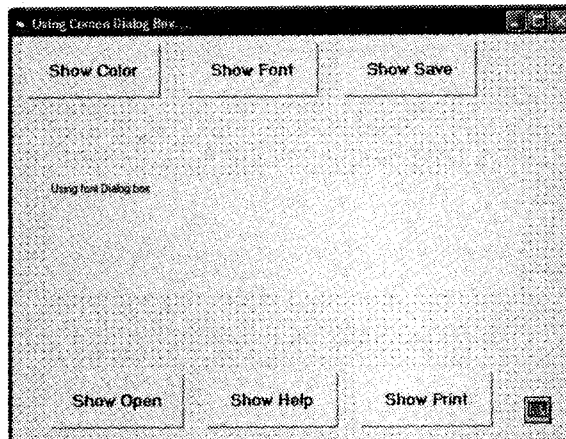


Figure 4.37

3. Set the properties and write the code on click event of the entire command button to open all the common dialog boxes.

Name	Label1
Caption	Using font Dialog box

Name	cmdColor
Caption	Show Color
Code	<pre>Private Sub cmdColor_Click() CommonDialog1.Flags = cdlCCRGBInit CommonDialog1.ShowColor frmcommondialog.BackColor = CommonDialog1.Color End Sub</pre>

Name	cmdFont
Caption	Show Font
Code	<pre>Private Sub cmdFont_Click() CommonDialog1.Flags = cdlCFBoth + cdlCFEffects CommonDialog1.ShowFont Label1.Font.Name = CommonDialog1.FontName Label1.Font.Size = CommonDialog1.FontSize Label1.Font.Bold = CommonDialog1.FontBold Label1.Font.Underline = CommonDialog1.FontUnderline Label1.ForeColor = CommonDialog1.Color End Sub</pre>

Name	cmdSave
Caption	Show Save
Code	<pre>Private Sub cmdSave_Click() CommonDialog1.ShowSave Label1.Caption = "FileName = " & CommonDialog1.FileName End Sub</pre>

Name	cmdOpen
Caption	Show Open
Code	<pre>Private Sub cmdOpen_Click() CommonDialog1.DialogTitle = "Open File" CommonDialog1.Filter = "Text (*.txt) *.txt" CommonDialog1.ShowOpen Label1.Caption = "FileName = " & CommonDialog1.FileName End Sub</pre>

Name	cmdHelp
Caption	Show Help
Code	<pre>Private Sub cmdHelp_Click() CommonDialog1.DialogTitle = "Help" CommonDialog1.HelpFile = "windows.hlp" CommonDialog1.HelpCommand = cdlHelpContents CommonDialog1.ShowHelp End Sub</pre>

Name	cmdPrint
Caption	Show Print
Code	<pre>Private Sub cmdPrint_Click() CommonDialog1.DialogTitle = "Print File" CommonDialog1.ShowPrinter MsgBox "Orientation = " & CommonDialog1.Orientation & "Copies = " & CommonDialog1.Copies End Sub</pre>

4. Save and run the application and check all the operations. Following figures shows the different dialog box displayed when you run this application.

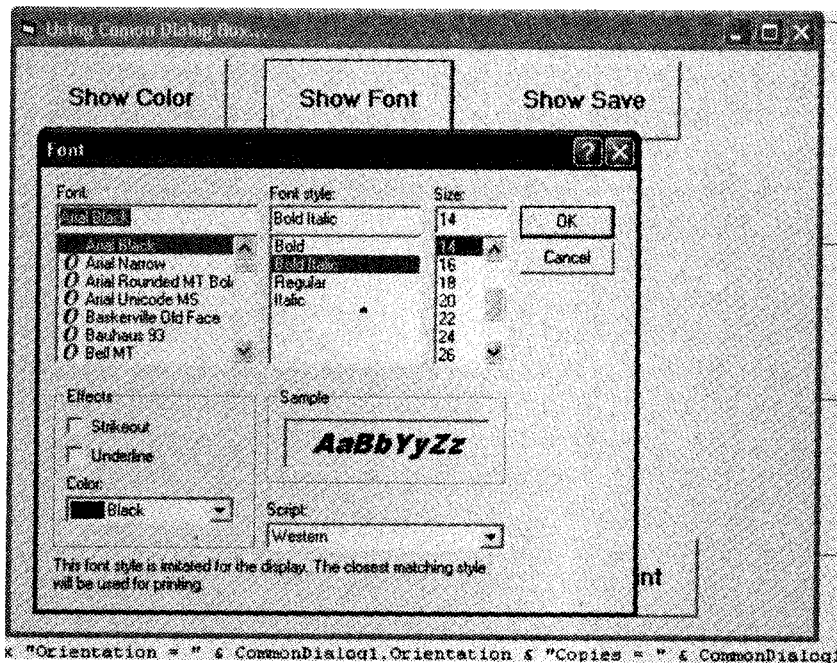


Figure 4.38: Showing Font Dialog Box

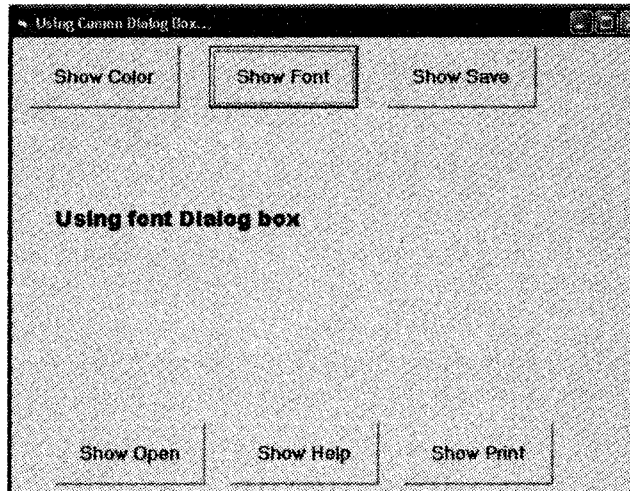


Figure 4.39

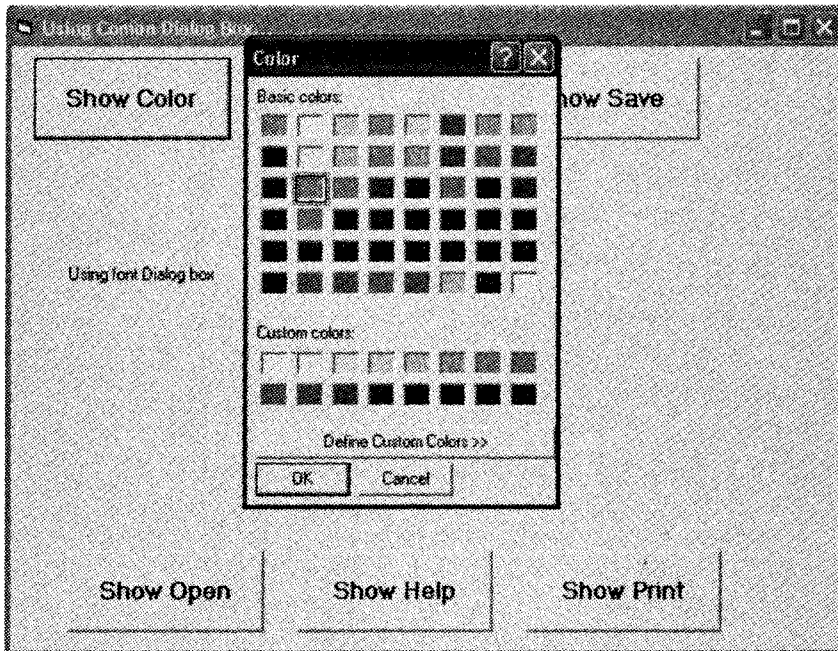


Figure 4.40: Showing Color dialog box

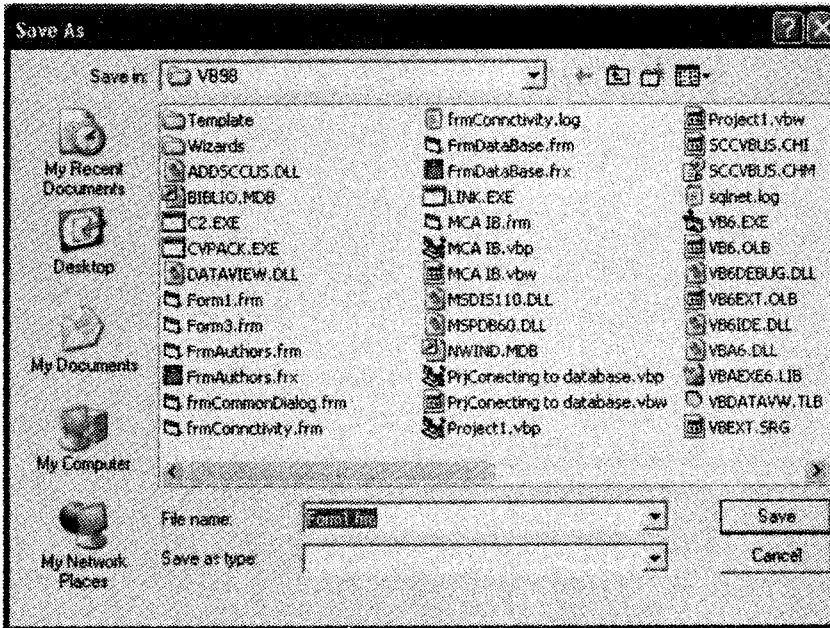


Figure 4.41: Save Dialog box

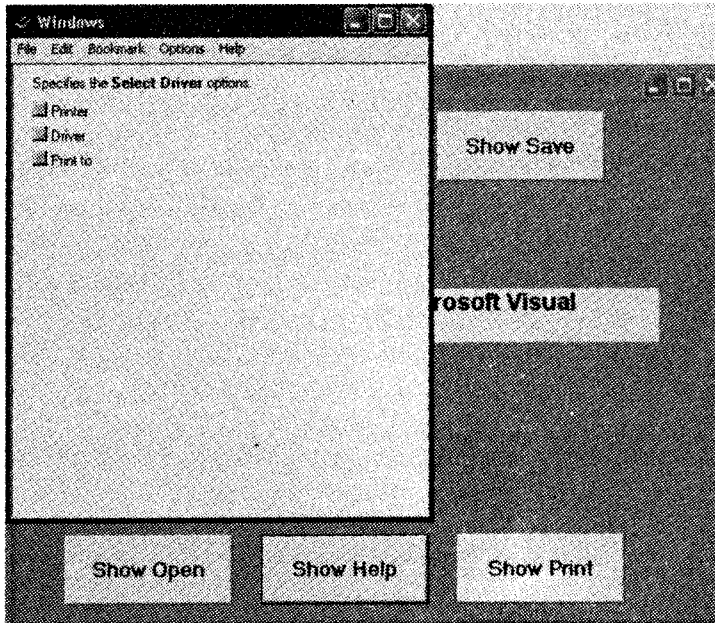


Figure 4.42: Help Dialog Box

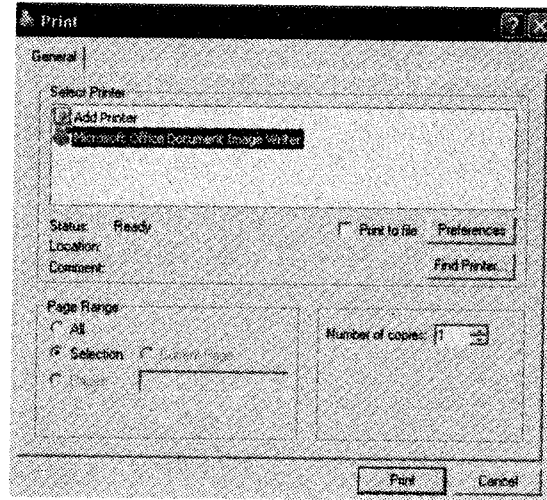


Figure 4.43: Print Dialog box

6. Creating a Menu System

We all are familiar with the windows menu system, it is one of the important and attractive item in Graphical User Interface. Visual Basic IDE also have menu similar to Microsoft Office menus. Most of the options are also similar. Microsoft has the consistency in designing the windows. All the Microsoft products windows look alike so that the user can work easily. Like VB standard menu you can design your own menu for your applications.

Menus are used to organize large number of options. They are fixed on the window. You can on / off tool bars but menus are always there on the screen. It is the standard way to interact with your application. You can use it with or without keyboard. Designing proper menu is one of the most important feature in developing an application because it is the gateway of your application. It should be user friendly so that user can easily handle your application.

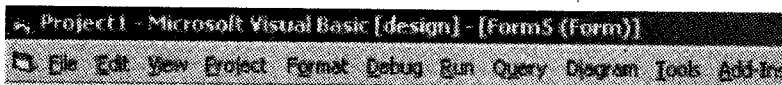


Figure 4.44: Visual Basic menu

You can see standard Visual Basic menu in *Figure 4.44*. You can have your own menu similar to this menu. In this unit we will learn designing your dynamic menu system, short cut keys to use menu, pop up menus etc.

6.1 Designing the Menu

Before starting designing menu let us understand all the components of the menu. In *Figure 4.45* you can see Visual basic Tools menu and its sub menus. Usually all the menu options appear in menu bar as shown in *Figure 4.44*. When you click any specific menu item it opens its sub menu options called as pull down menu.

Oct.2012 – 4M

How to create a Menu?
Explain with an example.

Now let's discuss about the entire components of the menu. Refer *Figure 4.45* to understand the following description. The menu options on the menu bar are *Top level menu* (Tools is the top level menu). When you click top level menu it opens its pull down menu. It is called as *command*. These commands can have *submenu* or it may execute directly. Submenus can also have other submenus. There is no limit on the number of submenus but more submenus can create confusion. Up to two or maximum three submenus are generally seen in the practice. (Publish menu has submenu) When the menu is not accessible to the user it is *disabled* or grayed out (Add procedure menu is grayed). But there should not be too many disabled menus instead you can add or remove menus at run time. To access the menu through keyboard you can create *access key* or it is also called as hot key. User can directly select the command by choosing access character and Alt key. The underlined character in the menu is called as an Access character. These keys are faster than selecting item through mouse. (T is access key in Tools menu) When you select any item it is highlighted by blue color (Publish menu is selected in the *Figure 4.45*). You can see there are three dots (...) with some menu items. It is called as *ellipses*. When you select ellipses it opens a dialog box. You have to reply that dialog box to execute that command. (If you select Menu Editor... option it will open menu editor window through which you can create your own menu.) One more component of menu is *short cut keys*. Shortcut keys provide an easier and quicker method of navigating and using the menu (Ctrl + E is the short cut for Menu editor option) you can directly access that command by the short cut key without selecting main menu. It saves your time. You cannot assign short cut key to top level menu. If you have more items in the menu you can group the items and can put *separator bar*. To create separator bar put Hyphen (-) character in the Caption property in the menu editor. But even these separator items must be assigned a unique value for the Name property.

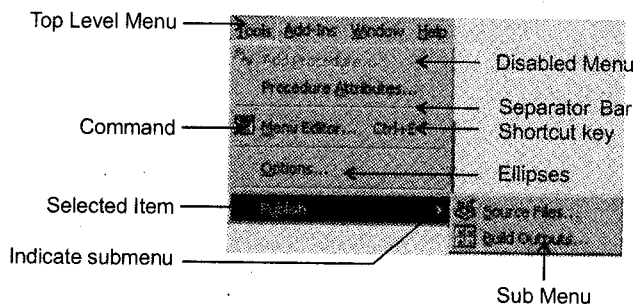


Figure 4.45: Different components of Menu

You can create your own menu similar to standard VB menu which we have discussed in the above section. Here we have discussed all the components of menu.

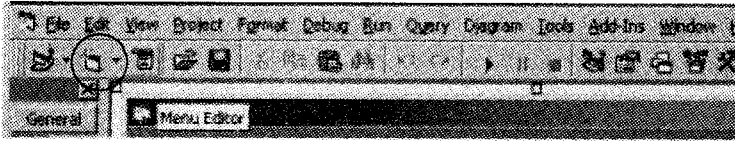


Figure 4.46: Menu Editor Button on tool bar

Visual basic provides menu editor to design menus. You can open menu editor by selecting Tools → Menu Editor option from main menu (you can see Menu Editor Menu in the *Figure 4.46*) or selecting Menu Editor Icon from standard tool bar (*Figure 4.46*). Or Press Ctrl + E short cut key to activate menu editor. You can also display the Menu Editor window by right clicking on the Form and selecting Menu Editor (*Figure 4.46*).

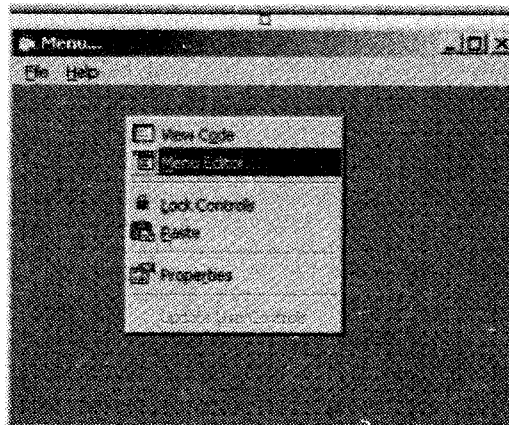


Figure 4.47

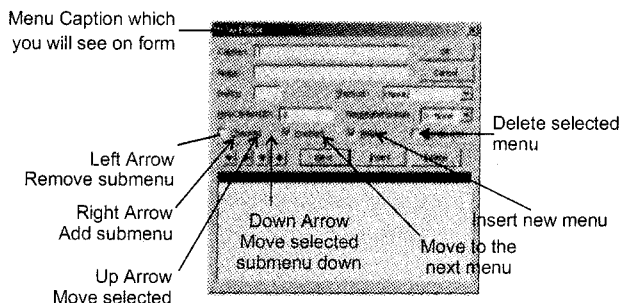


Figure 4.48: Visual Basic Menu Editor

The above window (*Figure 4.48*) will appear once you activate Menu Editor. The Menu Editor command is disabled unless the form is visible. Now before building menu with menu editor one should understand the use of all the components given on the menu editor.

1. **Caption:** This item is similar to the label or command button's caption property. It is the value which is visible on the form. You have to give some caption to the menu so that it can be visible on the form and it is possible to access it. As soon as you start typing the command's caption, it also appears in a new line in the list at the bottom of the Menu Editor window. To add more commands click Enter and type the Caption. Caption can be up to 40 characters long. The caption can be combination of number, special characters, alphabets, spaces, underscore. You can assign access key or hot key to the menu. Use & (ampersand) sign in front of the letter which you want to make as an access key. (*For example, &File: F* will be hot key, *E&xit: x* will be hot key) it makes that letter underlined in the caption. (*&File* → *File* or *E&xit* → *Exit*) with the use of hot key you can pull down that menu by pressing and holding down alt key and the letter which is underlined.
2. **Name:** Name value is not visible on the screen. But it is used in coding. If you forget to enter a menu item's Name, the Menu Editor throws an error when you wish to close menu editor. The prefix used with menu names is "mnu" as we have discussed in first unit that all control names start with some naming convention so that you can easily identify which control is that. Name cannot include spaces, keywords, and special characters.
3. **Index:** This item is used to create an array of menu commands. All the commands of the array have the same name and unique index that distinguishes them. It is similar to the array of the control.
4. **Shortcut:** At design time, you can assign the menu item a shortcut key so that your end users don't have to go through the menu system each time they want to execute a frequent command. The assigned shortcut key can't be queried at run time.
5. **Enabled:** Specifies whether a menu is disabled or not. A disabled command in a menu means that feature is not available.
6. **Visible:** By using this check box you can make your menu command visible or invisible on the screen.
7. **Checked:** This is unchecked by default and allows the programmer the option of creating a checked menu item. A menu item that act as a toggle and displays a check mark when selected.
8. **Window list:** This option is used with MDI applications to maintain a list of all open windows.
9. **Next:** Next moves you to the next menu item or inserts a new item if you are at the end of the menu. The indentation of the new item starts out the same as the indentation of the previous item. ALT+N is the access key

10. **Insert:** It inserts a menu item above the selected menu item. ALT+I is the access key to insert new menu item.
11. **Delete:** Deletes or removes the currently selected menu item. You can use ALT+T access key to delete the item. DEL key on the keyboard is also used for the same purpose.
12. **Ok and Cancel:** Click on the Ok button when you finished designing the menu. Cancel button is used if you decided to cancel designing your menu.
13. **Help Context ID:** If you want to add help system you can use help context ID.
14. **Left / Right arrow:** These buttons work with current menu items. When you want to create a submenu, you press the Right Arrow button (or the Alt+R hot key).
15. **Up / Down arrow:** You can move items up and down in the hierarchy by clicking the corresponding buttons or the hot keys Alt+U and Alt+B, respectively. Up and down arrows do not change the indentation.

One drawback of the menu editor is that you cannot copy menu from one application to another application. You can copy paste code or other controls like command button, text box etc. but it is not possible to copy menu editor menus from one place to another.

So are you ready to create your menu system? Then let's try it.

6.2 Creating the Menu with the Menu Editor

This section will give you an experience to create your menu system. Here I am creating menu with an *example* along with its description. You just need to follow all the instructions given below.

In this *example* we will create a menu which will look like *Figure 4.49*.

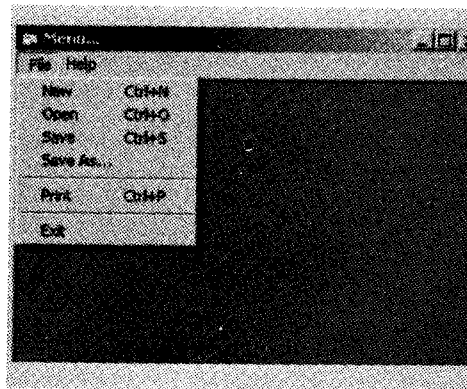


Figure 4.49: Menu

This form (Figure 4.49) contains a menu bar with two menus i.e. *File* and *Help*. These are called the Top level menu items.

If you click on The *File* menu, (shown in Figure 4.49), it will open pull-down menu having submenus: *New*, *Open*, *Save*, *Save As*, *Print*, and *Exit*. The line appears below and above *Print* menu and below *Open* menu is called as separator bar.

The *Help* menu contains just one item below it, i.e. *About* menu.

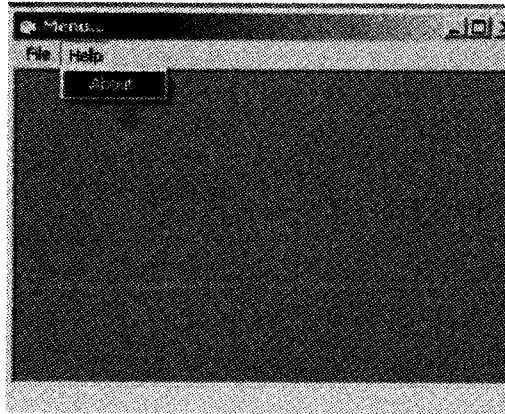


Figure 4.50

1
Apr. 2012 – 4M
What is Menu? How to create Menus using Menu Editor?

Figure 4.50 and Figure 4.51 had shows your final output. Now let's start from scratch.

As we have discussed earlier that to build menu you have Menu Editor. This appears as an icon in the toolbar of the VB IDE. It is the circled item in the screen shot below

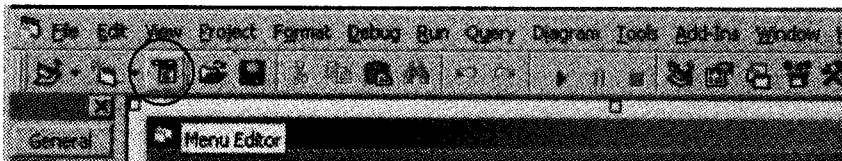


Figure 4.51: Menu Editor Icon

Alternatively, you can open the Menu Editor from the *Tools* menu item as shown below

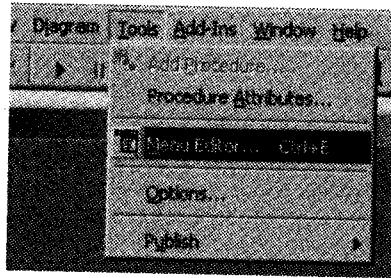


Figure 4.52: Menu Editor Option through tools menu

Step 1: Start a new VB project and invoke the Menu Editor using any of the method discussed above.

1. Click the Menu Editor Toolbar icon or
2. Select the Menu Editor option from the Tools menu. or
3. Right click on the form and open menu editor or
4. Press Ctrl+E short cut key to invoke Menu Editor.

The Menu Editor screen appears, as shown below in the *Figure 4.53*. If you have studied “section 5.1 designing the menu” well then you are already familiar with this window.

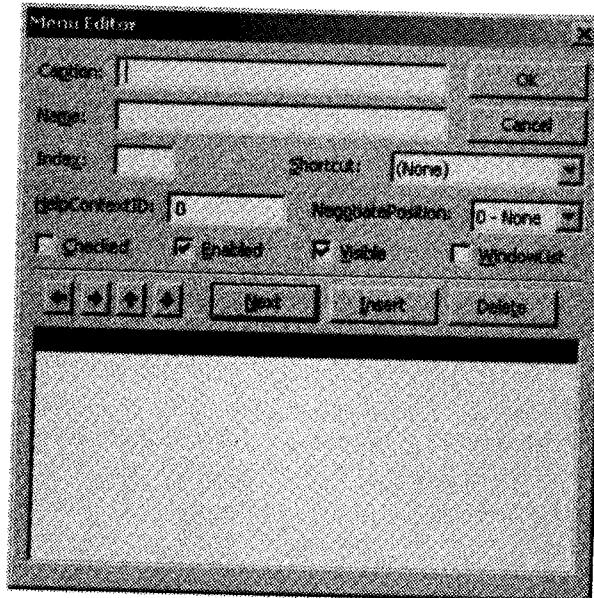


Figure 4.53: Menu Editor Window

Step 2: In "Caption" text box, type `&File` (The ampersand sign before "F", will make "F" as an access key for the File menu. It enables the user to drop down the File menu by keying "ALT+F" on the keyboard in addition to clicking the "File" menu with the mouse).

In "Name" text box, type `mmuFile`. Keep remaining settings as it is. You cannot assign short cut key to this menu as it is Top level menu. Therefore to move ahead click on the *Next* Button.

(Now onwards prefix every menu name with "mmu" convention).

Your Menu Editor screen should look like this

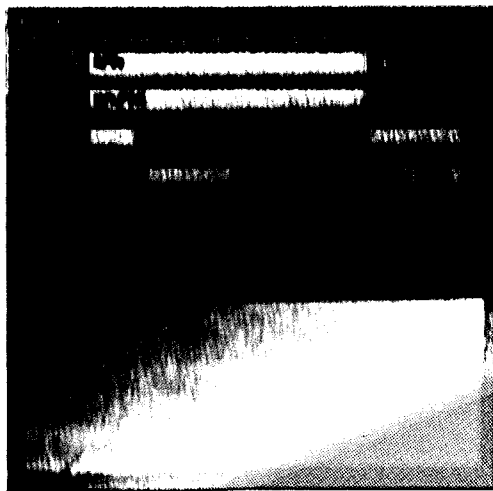


Figure 4.54

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

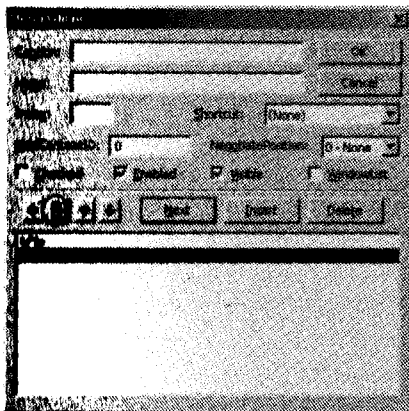


Figure 4.55

In "Caption" text box, type *&New*; and in "Name" text box, type *mnuNew*, and create a "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

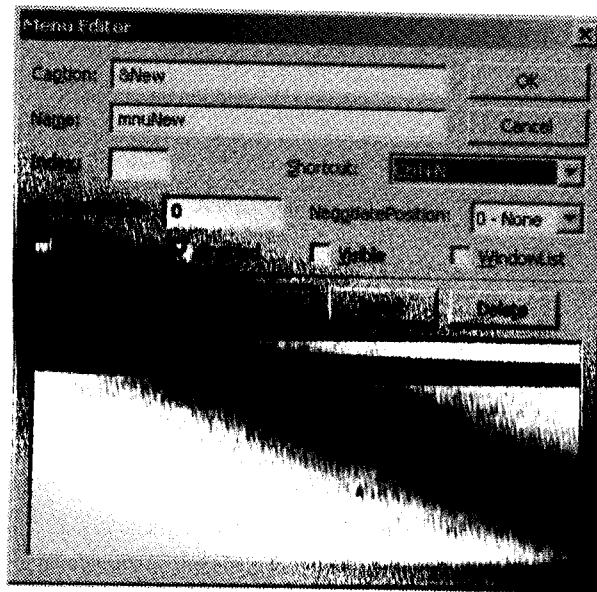


Figure 4.56

Click the Next button.

Step 4: In "Caption" text box, type *&Open*; in "Name" text box, type *mnuOpen*, and "Shortcut", select *Ctrl+O*. Your Menu Editor screen should look like this:

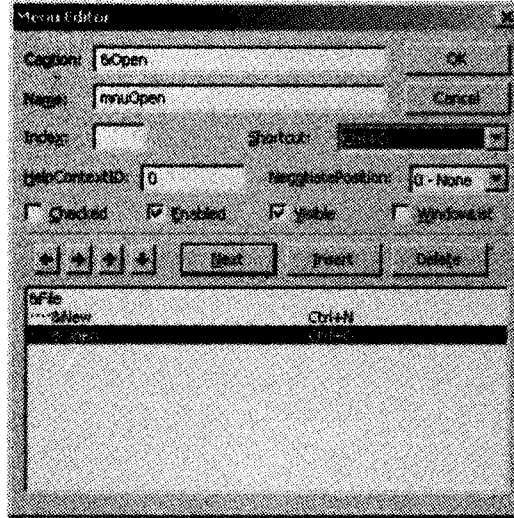


Figure 4.57

Click the Next button.

Step 5: In "Caption" text box, type - (a hyphen), and in "Name" text box, 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

Click the Next button.

Step 6: In "Caption" text box, type *&Save*; in "Name" text box, type *mnuSave*, and for "Shortcut", select *Ctrl+S*. Your Menu Editor screen should look like this

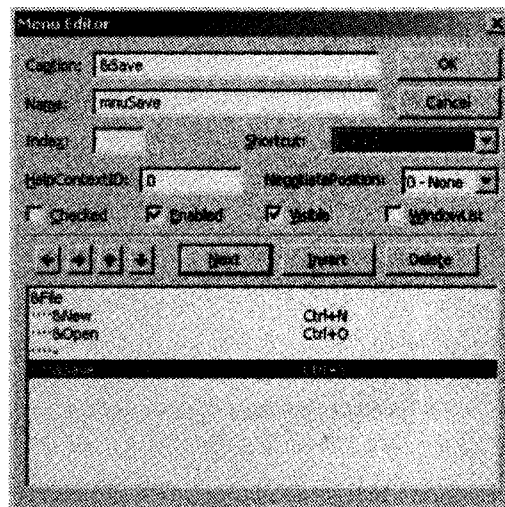


Figure 4.58

Click the Next button.

Step 7: In "Caption" text box, type *Save &As ...*, and in "Name" text box, type *mnuSaveAs*. Your Menu Editor screen should look like this

Click the Next button.

Step 8: In "Caption" text box, type *-*, and in "Name" text box, type *mnuFileBar2*. Your Menu Editor screen should look like this

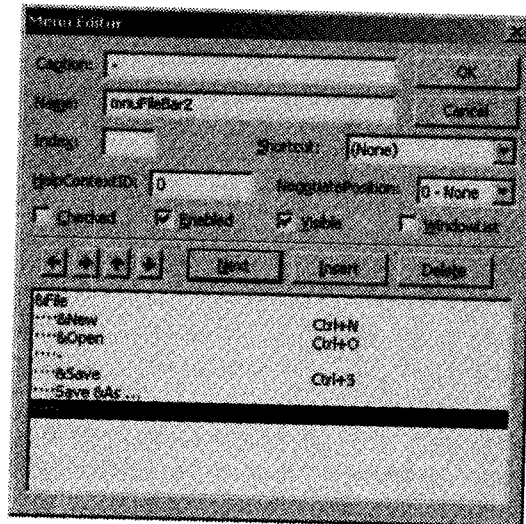


Figure 4.59

Click the Next button.

Step 9: In "Caption" text box, type *&Print*; in "Name" text box, type *mnuPrint*; and for "Shortcut", select *Ctrl+P*. Your Menu Editor screen should look like this

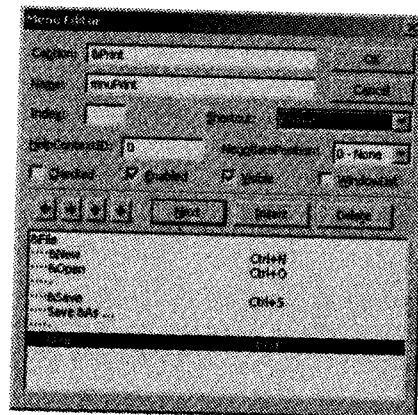


Figure 4.60

Click the Next button.

Step 10: In "Caption" text box, type - and in "Name" text box, type *mnuFileBar3*. Your Menu Editor screen should look like this:

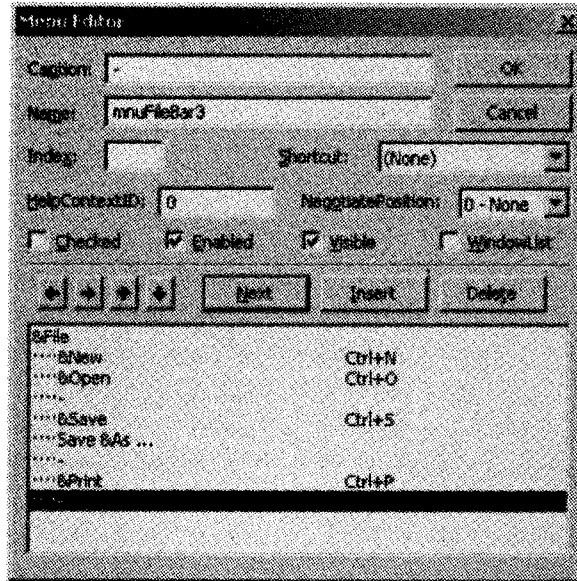


Figure 4.61

Click the Next button.

Step 11: In "Caption" text box, type *Exit*, and in "Name" text box, type *mnuExit*. Your Menu Editor screen should look like this

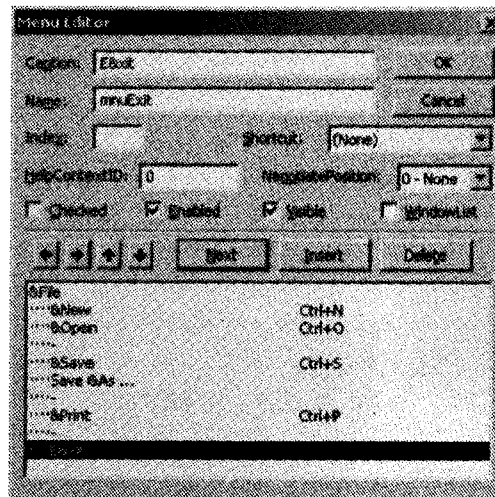


Figure 4.62

Click the Next button.

Step 12: Click the "left-arrow" button (shown circled below). The ellipsis (...) is removed, that means we are back to the top-level items.

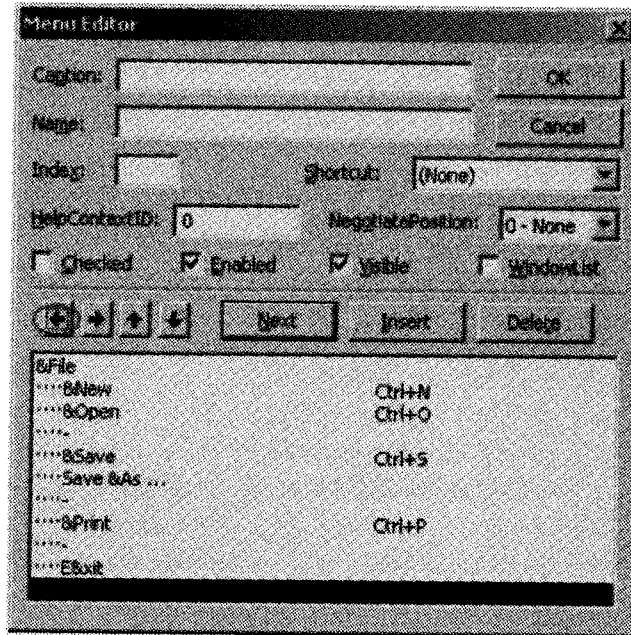


Figure 4.63

In "Caption" text box, type *&Help*; and in "Name" text box, type *mnuHelp*. Your Menu Editor screen should look like this:

Click the Next button.

Step 13: Click the "right-arrow" button to create a level-two item below "Help". In "Caption" text box, type *&About*; and in "Name" text box, type *mnuAbout*. Your Menu Editor screen should look like this:

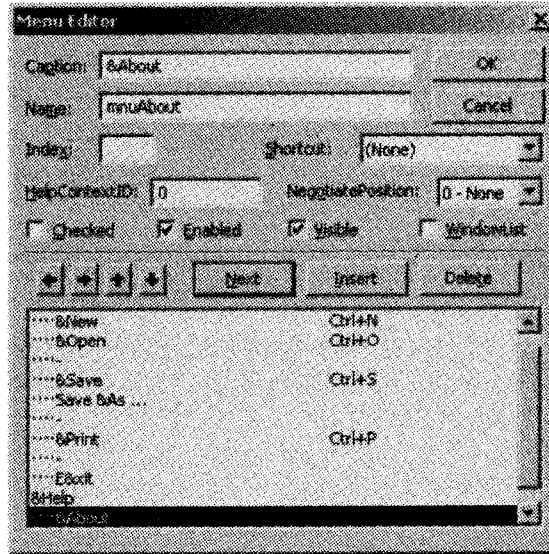
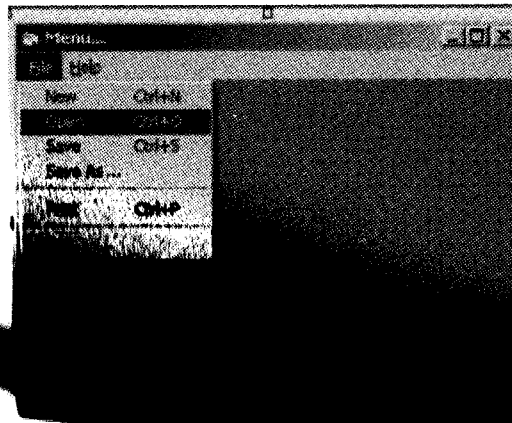


Figure 4.64

Step 14: All the menu entries are done at this point, so click the *OK* button. That will close the menu editor and return focus to the VB IDE.

Step 15: 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* or *Help*, the level-two menu will drop down. Or you can also use Alt+F for File and Alt+H for Help to invoke both the menus with keyboard.



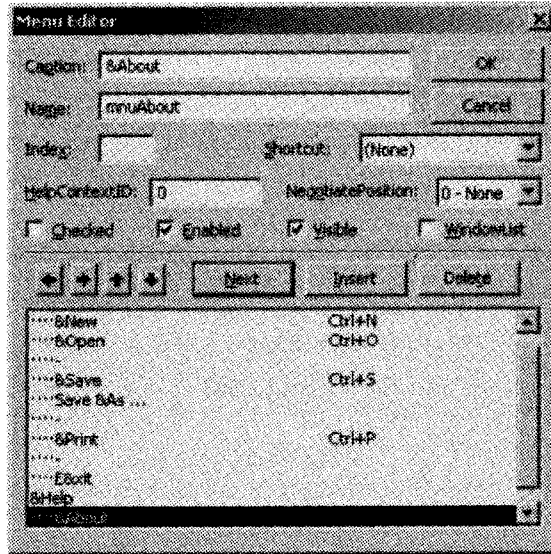
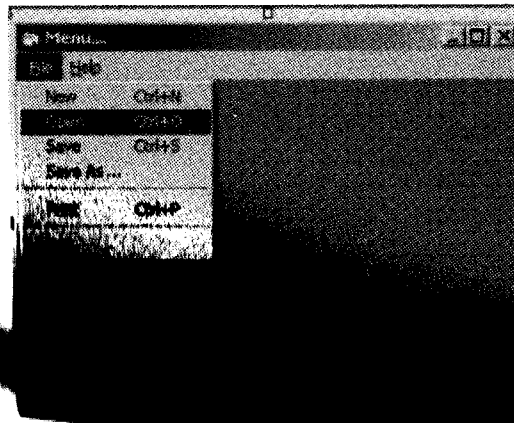


Figure 4.64

Step 14: All the menu entries are done at this point, so click the *OK* button. That will close the menu editor and return focus to the VB IDE.

Step 15: 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* or *Help*, the level-two menu will drop down. Or you can also use Alt+F for File and Alt+H for Help to invoke both the menus with keyboard.



Step 1 to 16 tell you how to design menu. Now let's have a look on the events and coding of the menu. To perform any action through menu item we can write code for that menu items. But remember that *Click* is the only event that a menu item can respond to.

Step 16: Click on the *Open* menu item. The code window for the *mnuOpen_Click* event opens, as shown below.

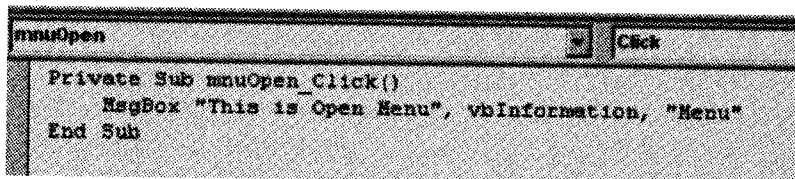


Figure 4.66

Under the sub procedure *mnuOpen_Click* (), place the code you want to execute when the user clicks the Open menu item. You can write any valid VB code over there. As this is demo example we will place simple *MsgBox* statement in the event procedure:

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

Step 17: Add similar code for other menu items like New, Save, Save As, and Print menu. To open sub procedure of that menu just click on the menu item on the form and VB will open code window with the sub procedure for that menu. Or in code window select the menu item from object list and its event from procedure list. Here it will be *Click* event for all objects.

```
Private Sub mnuNew_Click ()
    MsgBox "Code for 'New' goes here.", vbInformation,
"Menu Demo"
End Sub
Private Sub mnuSave_Click ()
    MsgBox "Code for 'Save' goes here.", vbInformation,
"Menu Demo"
End Sub
Private Sub mnuSaveAs_Click ()
    MsgBox "Code for 'Save As' goes here.", vbInformation,
"Menu Demo"
End Sub
Private Sub mnuPrint_Click ()
    MsgBox "Code for 'Print' goes here.", vbInformation,
"Menu Demo"
End Sub
```


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

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

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

```
Private Sub mnuAbout_Click ()  
    MsgBox "Menu..." & vbNewLine & "Copyrights 2009 M S",  
    vbInformation, "Menu"  
End Sub
```

We have finished coding. Now try your application.

Step 20: Run the program.

(If you have many forms in your project and this menu form is not currently selected as startup object. you need to change start up object. Select Project menu → Project Properties... This will open a dialog box as shown in the *Figure 4.67*. Select your form name from the Startup Object List box and click OK button.)

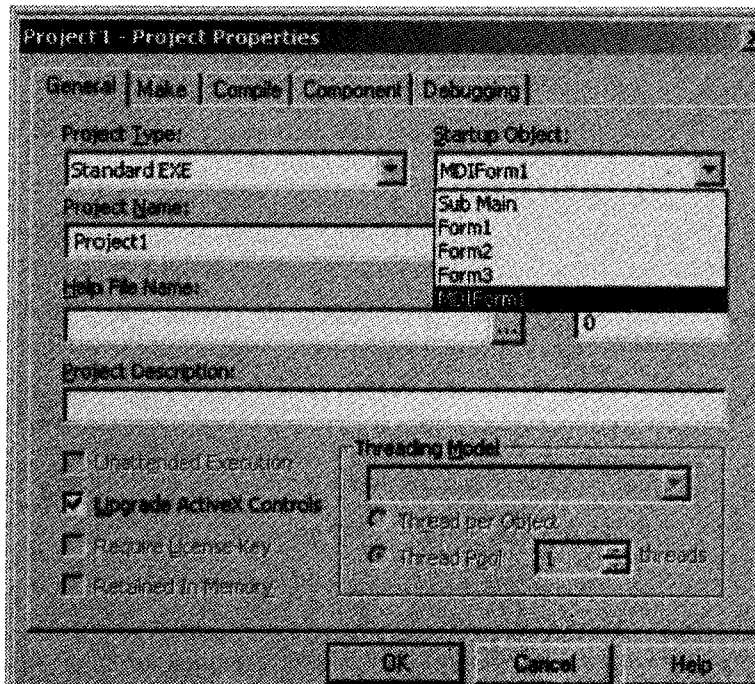


Figure 4.67

Step 21: Now run your application. Check how the code executes when you click on the various menu items. Also test the use of the access keys (*For example, Alt+F, N*) and shortcut keys (e.g., *Ctrl-O*).

Step 22: Save the program and exit VB.

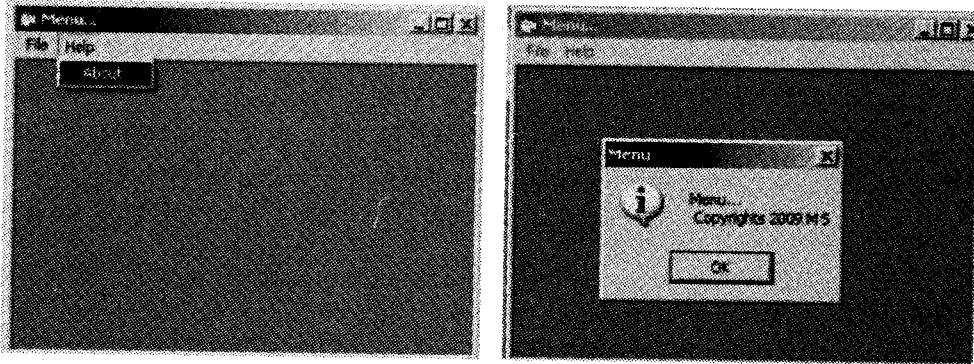


Figure 4.68

6.3 Adding Shortcut and Access Keys to Menu Items

You must be agreeing that using the keyboard is sometimes faster than using mouse. Shortcut keys can help you bypass menus and carry out commands directly. Shortcut keys are listed next to the command name on menus. *For example*, on the *File* menu, the *Save* commands have the shortcut CTRL+S. While you design your menu system through VB menu editor there is one list box labeled as shortcut, you can assign shortcut key for the menu item from this list. (See the circled item in *Figure 4.69*)

Access keys are established when you give caption to the menu. You have to prefix the access letter with & (ampersand) sign. An access key appears as underlined letters in the command itself. *For example*, on *File* Menu *F* is access key.

Remember that you can assign access key to the top level menu but you cannot have short cut key to top level menu. Both the keys must be unique for all menus in the application.

In the above section we have discussed one example of menu in details. In that *example* we have already assigned shortcut keys and access keys to the menu items.

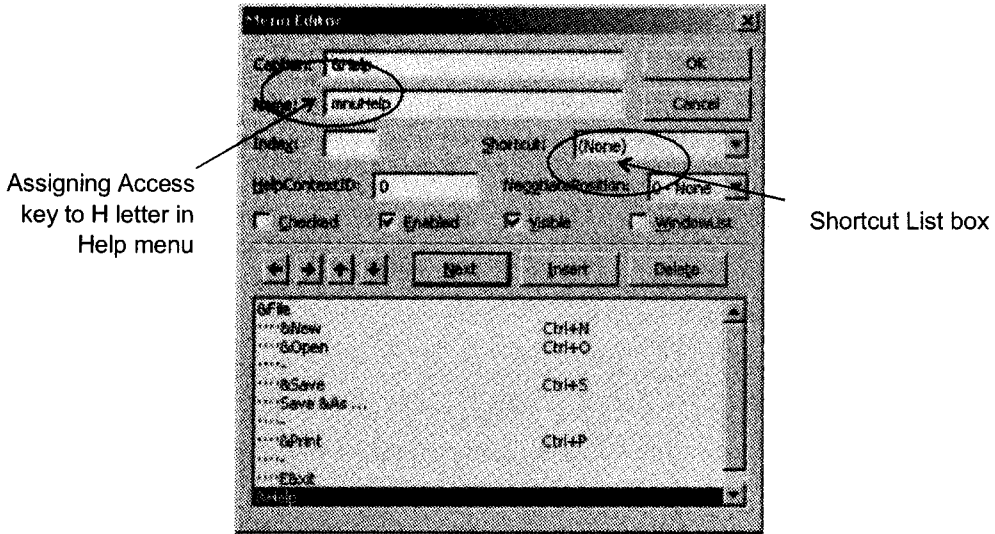


Figure 4.69

7. Creating and Accessing Popup Menu

Along with tool bars another frequently used menu type is popup menu. It can appear anywhere within the form or any control through the right mouse button. Normally it appears at the location of the mouse click. Nearly every window application provides a popup menu also called as shortcut menu or context menu. This menu can be displayed anywhere on a form or a control. It is invoked with `popupmenu` method. The popup menu method is usually called from within textbox or picturebox control because these controls can carry out edit operations.

The Syntax for the `PopupMenu` method

`PopupMenu` *menuname*, *flags*, *x*, *y*, *boldcommand*

Where,

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:

1

Apr.2013 – 4M

Write a short note on:
Popup.

Constant (location)	Value	Description
<code>vbPopupMenuLeftAlign</code>	0	(Default) The left side of the pop-up menu is located at x.
<code>vbPopupMenuCenterAlign</code>	4	The pop-up menu is centered at x.
<code>vbPopupMenuRightAlign</code>	8	The right side of the pop-up menu is located at x.

Constant (behavior)	Value	Description
vbPopupMenuLeftButton	0	(Default) An item on the pop-up menu reacts to a mouse click only when you use the left mouse button.
vbPopupMenuRightButton	2	An item on the pop-up menu reacts to a mouse click when you use either the right or the left mouse button.

You can specify both "location" and "behavior" constant, by adding the two values together.

For example,

PopupMenu PMenu, vbPopupMenuRightAlign + vbPopupMenuRightButton

X & Y: arguments are the coordinates of a point on the form or control where the menu will be displayed. If x & y coordinates are ignored the popup menu will appear at the pointers location.

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.

Popup menus are sometimes referred to as speed menus, right-click menus, or context menus. Popup menus are generally invoked by right-clicking the mouse button. Most applications will display a specific menu. It depends on the user context that which menu is shown hence the name context menu.

7.1 Creating Pop-up Menu

Popup menus are created using menu editor. Make a top-level menu and set its visible property to False. In the MouseDown event handler where you want to display the menu, make sure the right button is pressed and then use PopupMenu to display the menu.

Now we will learn to create a *popup menu* through following examples. Follow the steps given below:

Example 1

In this example we will create a simple popup menu. This menu has four options. Menu will be activated when you right click anywhere on the command button or on the form. If you select any of the option it will display a message box.

Step 1: Start a new VB project, add a form and place a command button and label on the form and set the following properties of both the controls: the form should look like *Figure 4.70*.

Command Button

Name: cmdPopupMenu.

Caption: "Show Popup Menu"

Font: Bold, size 14

Label

Name: lblMsg

Caption: "Right Click with mouse anywhere on the form to see pop up menu"

Font: Bold, size12

Autosize: True

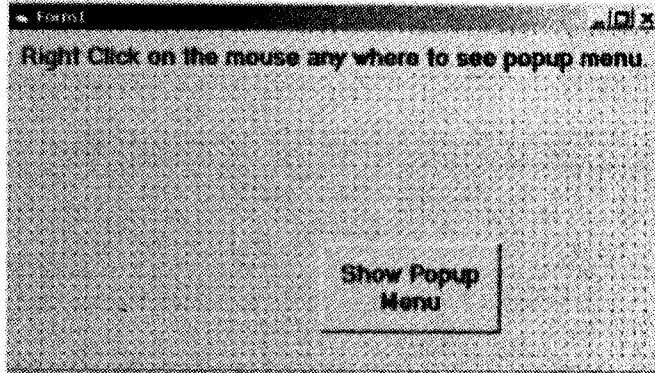


Figure 4.70

Step 2: Open the Menu Editor, and create a top-level item with a Caption value of *Popup Menu* and the Name *mnuPopup*. Do not forget to *uncheck* the *visible* checkbox. To make menu as a pop-up menu, it must be invisible. (See the circled item below)

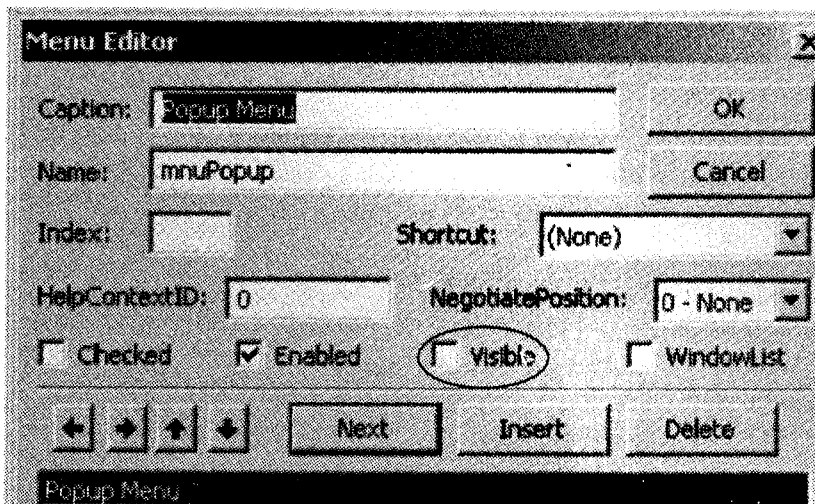


Figure 4.71

Step 3: Create level-two menu items below the **Popup Menu** top-level menu. When creating these level-two items, keep the Visible box checked. Click right arrow to create submenu and create three menu items at the same level as shown in the *Figure 4.71*. Those are as follows

Caption Property	Name Property
Option &1	mnuOpt1
Option &2	mnuOpt2
Option &3	mnuOpt3

After entering all the menu items, your menu editor screen should look like *Figure 4.72*

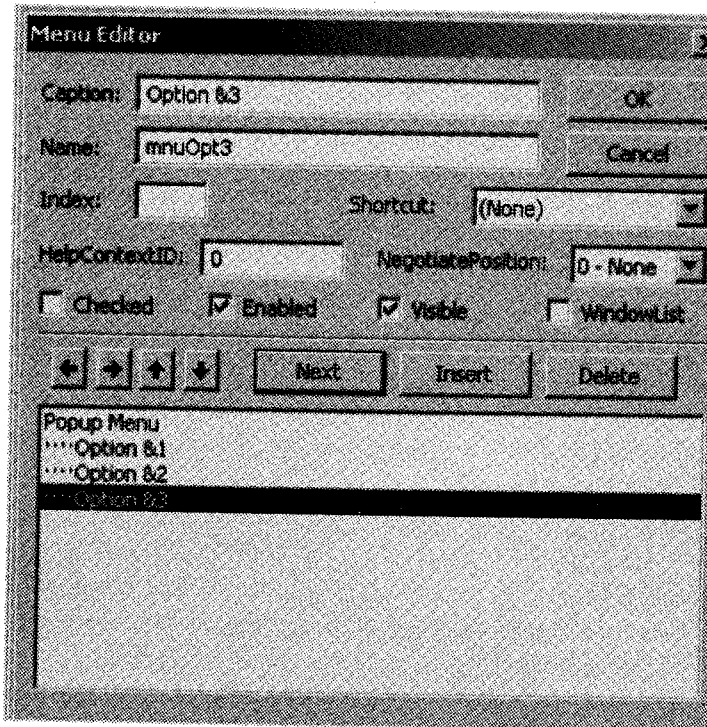


Figure 4.72

- Step 4:** Click OK to save and close your menu. When you return to the form, you will see only label and command button there is no menu on the form. Because pop-up menu will only be visible when invoked through code.
- Step 5:** Now code the MouseDown event for command button and Form. The code is as shown below. The Button parameter is tested for vbRightButton 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 appears.

Option Explicit

```
Private Sub cmdPopupMenu_MouseDown (Button As Integer,
Shift As Integer, X As Single, Y As Single)
```

```
    If Button And vbRightButton _
        Then PopupMenu mnuPopup
```

```
End Sub
```

```
Private Sub Form_MouseDown (Button As Integer, Shift As
Integer, X As Single, Y As Single)
```

```
    If Button And vbRightButton _
        Then PopupMenu mnuPopup
```

```
End Sub
```

Step 6: When you select any of the Option submenus it will trigger click event and display a message box. So code for all the option menus i.e. option1, option2 and option3 is as shown below.

```
Private Sub mnuOpt1_Click()
    MsgBox "I am Option 1"
```

```
End Sub
```

```
Private Sub mnuOpt2_Click ()
```

```
    MsgBox "I am Option 2"
```

```
End Sub
```

```
Private Sub mnuOpt3_Click ()
```

```
    MsgBox "I am Option 3"
```

```
End Sub
```

Save the code.

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

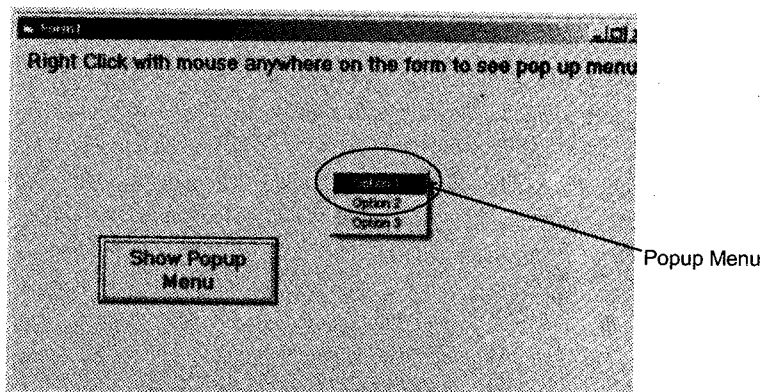


Figure 4.73

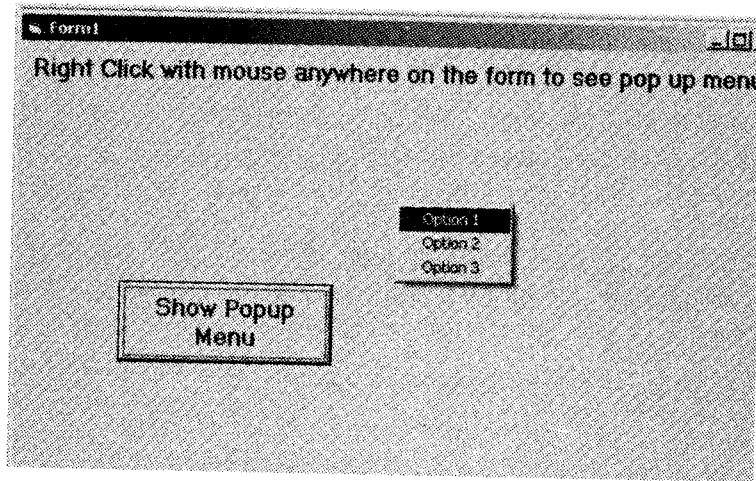


Figure 4.74

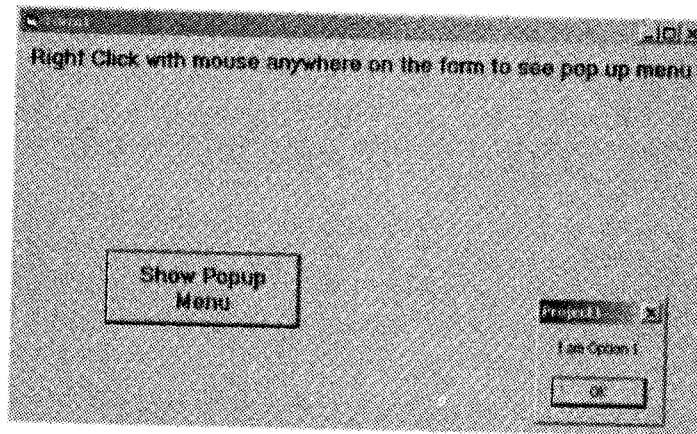


Figure 4.75

Step 8: Save the application and exit VB.

Example 2

In this *example* we will try some more VB operations through popup menu. Create a VB project and display one label on the form. Change the font color and font size of the label through popup menu.

Step 1: Start a new VB project and place a label on the form. Set the following properties of the label control:

Name: *lblMsgText*

Caption: *Hello how are you!!!!*

Autosize: *True*

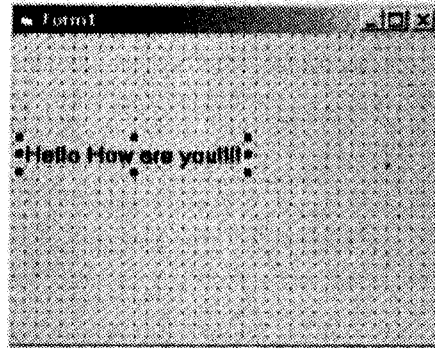


Figure 4.76

Step 2: Open the Menu Editor, and create a top-level & low-level menu as shown in the *Figure 4.76*. Set top-level item with a Caption value of PopUpMenu and the Name `mnuPopUpMenu`. Also *uncheck* the *visible* checkbox and create other menu items. (As we have practiced designing menu in the previous examples so it is considered that you know how to design the following menu system as in *Figure 4.76*)

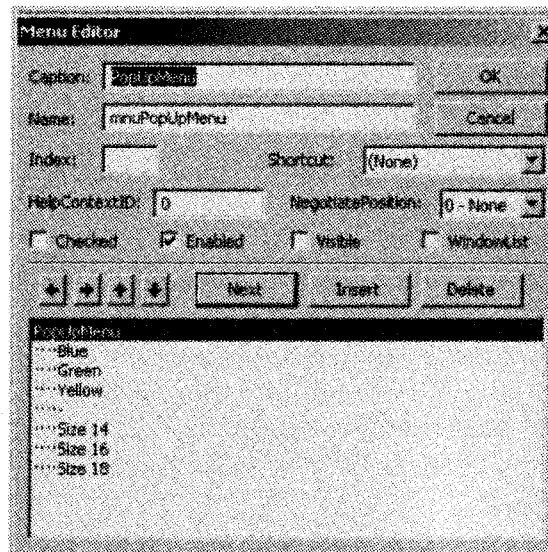


Figure 4.77

Step 3: Once you finished with designing menu system click *OK* to save and close menu editor. You will see the form still look like the *Figure 4.77*. i.e. there is no any menu on the form. As it is popup menu it is not visible at this point. It will activate when you write code for that.

Step 4: Code the *lblMsgText_MouseDown* event as shown below. This will pop up the menu only if the user right-clicks on the label. It is not available on the form. If the user clicks the right mouse button, the *PopupMenu* statement is executed. It is this statement that makes the pop-up menu appears.

```
Option Explicit
Private Sub lblMsgText_MouseDown (Button As Integer, Shift
    As Integer, X As Single, Y As Single)
    If Button = vbRightButton Then
        PopupMenu mnuPopuUpMenu, vbPopupMenuRightButton
    End If
End Sub
```

Step 5: Code the *mnuBlue_Click* event as shown below. When you click on Blue menu item the text color will become blue

```
Private Sub mnuBlue_Click()
    lblMsgText.ForeColor = vbBlue
End Sub
```

Step 6: Code the *mnuGreen_Click* event as shown below. When you click on Green menu item the text color will become Green

```
Private Sub mnuGreen_Click()
    lblMsgText.ForeColor = vbGreen
End Sub
```

Step 7: Code the *mnuYellow_Click* event as shown below. When you click on Yellow menu item the text color will become Yellow

```
Private Sub mnuYellow_Click()
    lblMsgText.ForeColor = vbYellow
End Sub
```

Step 8: Code the *mnusize14_Click* event as shown below. When you click on Size 14 menu item the text Size will become 14

```
Private Sub mnusize14_Click ()
    lblMsgText.FontSize = 14
End Sub
```

Step 9: Code the *mnusize16_Click* event as shown below. When you click on Size 16 menu item the text Size will become 16

```
Private Sub mnuSize16_Click()
    lblMsgText.FontSize = 16
End Sub
```

Step 10: Code the *mnusize18_Click* event as shown below. When you click on Size 18 menu item the text Size will become 18

```
Private Sub mnusize18_Click ()  
    lblMsgText.FontSize = 18  
End Sub
```

Step 11: Save the code. Now your application is now ready to run. Close the code window and return to the VB IDE. Run the program and check out the various options you have coded.

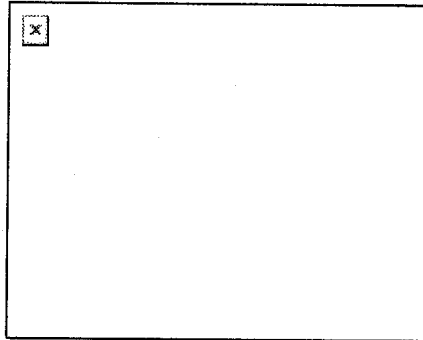


Figure 4.78

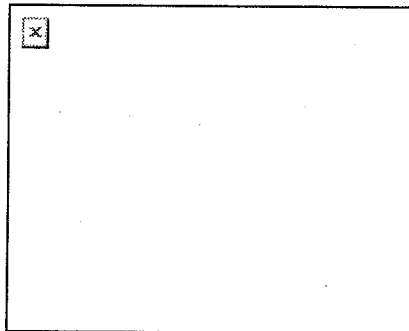


Figure 4.79

8. Adding or Modifying Menu at Run Time (Dynamic Menu)

This section describes the use and implementation of dynamic menu i.e. the menu which can be changed at run time. You can add, delete or modify the menu items at run time. There are some situations where user needs to do such operations. Fortunately VB6 provides this facility. Further topic describes the dynamic menu in detail.

Menus created at design-time can be modified in runtime and are referred as dynamic menus. To create a dynamic menu, you need to create a control array of menu items. But you have to create the first item in design-time, and then you can dynamically add items in runtime using a control array. In the menu editor window, add a menu item and set its index property to 0. You can then add commands with the same name and consecutive index values. At design time you don't have to add more than one item. One menu item with index value set to 0 is sufficient to create the menu control array. You can use this array's name and an index number to add new option at run time. You can hide the items you create in runtime using Hide method or setting the visible property to False.

Let's follow the tradition and explain this too with an example. Follow the instruction given below. You will learn how to design this menu. So ready!!!

In this example we will create a simple run time menu system,

Step 1: Create new VB project and open the Menu Editor.

Create two menu items

1. Caption: &MyMenu Name: mnuMyMenu and
2. Caption-Name: mnuMenuList, and do not forget to set Index property to 0. As shown in the following *Figure 4.80*

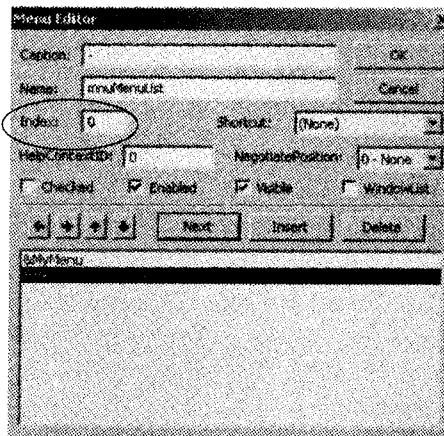


Figure 4.80

Step 2: Display two command buttons on the form and set the following properties of the command button

Name: cmdAdd and cmdDel

Caption: "Add Menu Item" and "Remove Menu Item"

Your screen should look like *Figure 4.81*

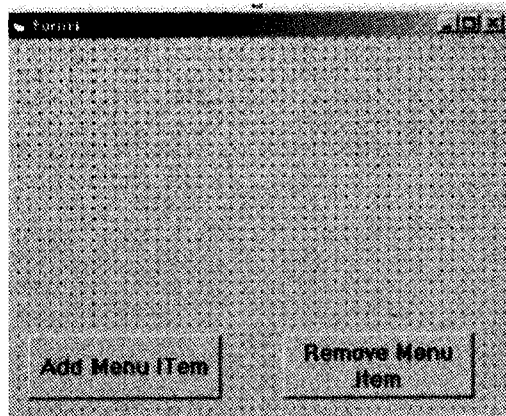


Figure 4.81

Add the following code to the cmdAdd_Click () event. You can add menu items through this event:

```
Private Sub cmdAdd_Click()
    AddMenu
End Sub
```

Then, add this code to the cmdDel_Click () event. You can delete menu items through this event:

```
Private Sub cmdDel_Click()
    DelMenu
End Sub
```

Now, write the code for the Private Sub AddMenu (). This code will ask the user the number of menu items he/she wants to add. And will insert that many items in that menu

```
Private Sub AddMenu ()
    Dim i As Integer, j As Integer
    Call DelMenu
    j = InputBox ("How many elements you want to enter?")
    For i = 1 To j
        Load mnuMenuItem (i)
        mnuMenuItem (i).Caption = "Menu item " & i
    Next i
    mnuMenuItem (0).Visible = False
End Sub
```

Finally, add Private Sub DelMenu () with this code. This code will empty the menu completely but leave the divider which is created in design-time

```
Private Sub DelMenu ()
    Dim i As Integer
    mnuMenuItemList(0).Visible = True

    For i = 1 To mnuMenuItemList.UBound
        Unload mnuMenuItemList(i)
    Next i
End Sub
```

Step 3: Save the code and run the project. If you click on the “MyMenu” menu, you’ll see the divider, as shown in *Figure 4.82*. But when you click on the Add Menu Items button one input box will popup. (*Figure 4.83*) Asking you the how many number of elements you want to insert? Enter any integer number into the textbox. And press OK button. Now click on the MyMenu menu again, you will see menu items added in the menu as low level menu items. The number of menu item will be equal to the number you have entered in the input box. This time your screen should look like *Figure 4.83*.

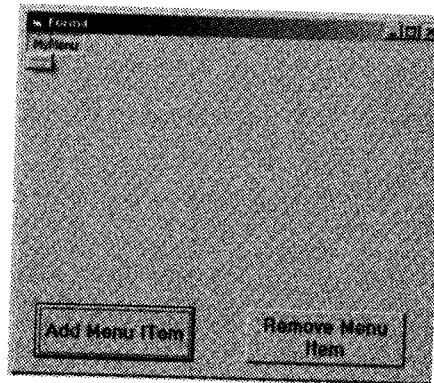


Figure 4.82: Menu with separator

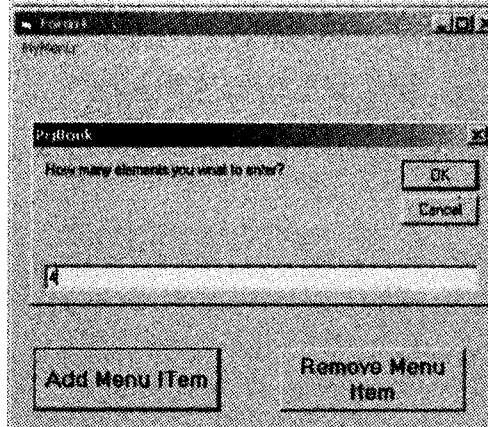


Figure 4.83

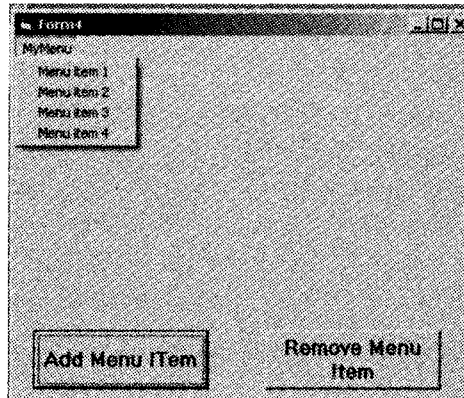


Figure 4.84

Step 4: Now if you click on the “Remove Menu Item” command button it will delete all the menu items inserted by “Add Menu Item” button only separator will remain because it was created at the time of menu design. Now the screen will again look like *Figure 4.84*.

Enabling Menu Item in response to program state

You can enable or disable menu items at design time as well as at run time by using `Enabled` property of the menu. Set this property to `True` to enable and to `false` to disable the menu. If you want to set this property at design time select or deselect `\enabled` checkbox on the menu editor. The menu item will behave according to the selection of check box. Or you can set it through coding at run time.

For example, if you want to make any menu disable at run time write the following code.

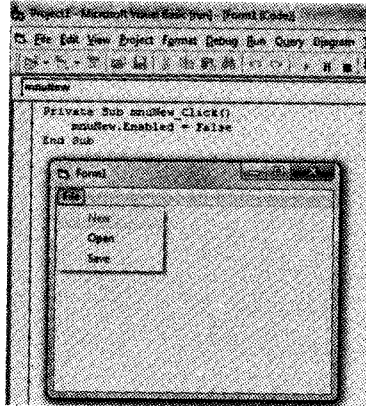


Figure 4.85

```
Private Sub mnuNew_Click()  
    mnuNew.Enabled = False  
End Sub
```

The same way you can make menu items visible or invisible at run time as well as design time. Set visible property to true to make menu item visible or to false to make it invisible. In the following example I have created one command button which makes the File menu visible and invisible. The following code is written on the click event of the cmdVisible command button.

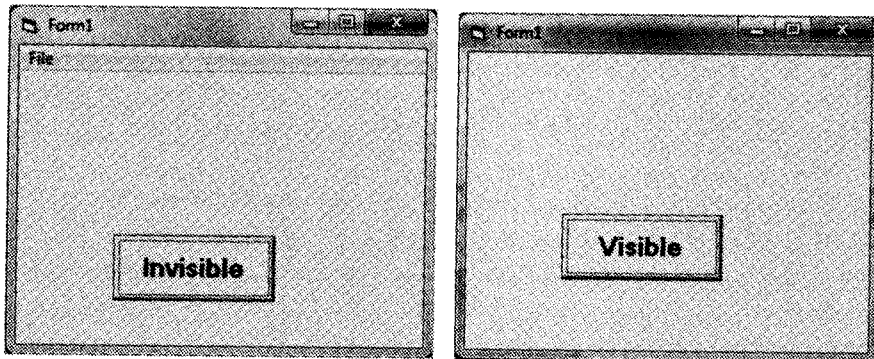


Figure 4.86

```
Private Sub cmdVisible_Click()  
If cmdVisible.Caption = "Visible" Then  
    mnuFile.Visible = True  
    cmdVisible.Caption = "Invisible"
```



```
Else
    mnuFile.Visible = False
    cmdVisible.Caption = "Visible"
End If
End Sub
```

9. Adding Menu Item For MDI Child Form

MDI (Multiple Document Interface) Application is an application in which we can view and work with several documents at once. MDI was designed to simplify the exchange of information among documents, all under the same roof. With the main application, you can maintain multiple open windows, but not multiple copies of the application.

3
Apr. 13, 10, Oct. 12 – 4M
Write short note on:
MDI.

You almost certainly use Windows applications that can open multiple documents at the same time and allow the user to switch among them with a mouse-click. Each document is displayed in its own window, and all document windows have the same behavior. The main Form, or MDI Form, isn't duplicated, but it acts as a container for all the windows, and it is called the parent window. The windows in which the individual documents are displayed are called Child windows.

An MDI application must have at least two form, the parent form and one or more child forms. Each of these forms have certain properties. There can be many child forms contained within the parent form, but there can be only one parent form i.e. only one MDI form. The parent form may not contain any controls except child forms. The parent form usually has its own menu.

You can add MDI form to your application using following steps

1. Start a new project
2. Select Project Menu → chooses "Add MDI Form" option.
3. Set the Form's caption to MDI Window
4. Select Project → Add Form to add a SDI Form.
5. Make this form as child of MDI form by setting the MDI Child property of the SDI form to True. (You can make all the form child form to this MDI parent form by setting MDI child property true.)
6. Set the caption property to MDI Child window.

VB6 automatically associates this new Form with the parent Form. This child Form can't exist outside the parent Form; in the words, it can only be opened within the parent Form.

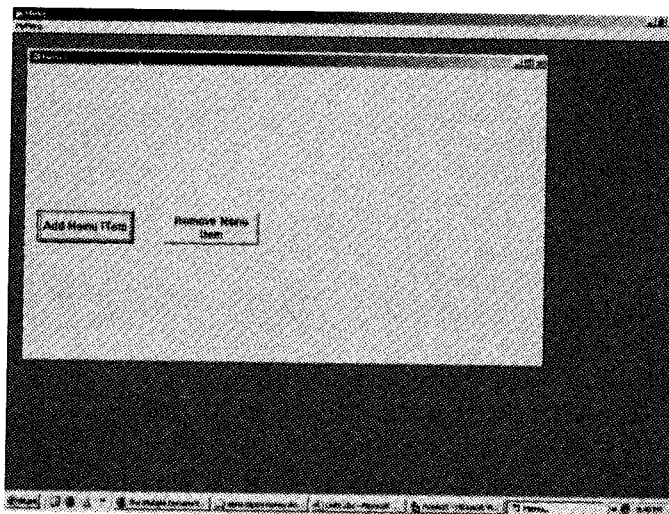


Figure 4.87: Parent and Child Menu

The MDI Form usually has load and quit menu for the application. The child Form is nothing but the general VB form having any number of controls, operations and code for any event. When the child Form is loaded, the child Form's menu replaces the original menu on the MDI Form. As you can see in the *Figure 4.87* dynamic menu "MyMenu" which is designed with child form is now visible as parent form's menu.

Solved Programs

2

Apr.13, 12 – 4M

1. Write a menu driven program for:

- i. Area of Circle
- ii. Area of Rectangle

Solution

```

Declare X As Float
Declare Y As Float
Declare Choice As Integer
X= Val(InputBox("Enter a number:"))
Y= Val(InputBox("Enter a number:"))
Call InputChoice(Choice)

Select Case Of Choice
Case: 1
Call Circle(X)
Case: 2
Call Rectangle(X,Y)
default:

```

```
Write "Input not understood. Run the program again."  
End Case  
End Program
```

```
Subprogram InputChoice(Integer Choice As Ref)  
Write "Enter 1 to compute the area of a circle "  
Write "Enter 2 to compute the area of a rectangle "  
Input Choice  
End Subprogram
```

```
Subprogram Circle (Float X)  
Declare A As Float  
Set A = 3.14*X^2  
Write "The area of the circle is:"  
Write A  
End Subprogram
```

```
Subprogram rectangle(Float X, Float Y)  
Declare A As Float  
Set A = X * Y  
Write "The area of the rectangle is:"  
Write A  
End Subprogram
```

2. Write a menu driven program in VB to perform the following:

- i. Area of Square**
- ii. Area of Rectangle**
- iii. Area of Triangle**

Solution

```
Declare X As Float  
Declare Y As Float  
Declare Choice As Integer  
X= Val(InputBox("Enter a number:"))  
Y= Val(InputBox("Enter a number:"))  
Call InputChoice(Choice)  
  
Select Case Of Choice  
Case: 1  
Call Square(X)  
Case: 2  
Call Rectangle(X,Y)  
Case: 3  
Call Triangle(X,Y)  
Default:  
Write "Input not understood. Run the program again."  
End Case  
End Program  
Subprogram InputChoice(Integer Choice As Ref)
```

```
Write "Enter 1 to compute the area of a square "
Write "Enter 2 to compute the area of a rectangle "
Write "Enter 3 to compute the area of a triangle "
Input Choice
End Subprogram
```

```
Subprogram Square(Float X)
Declare A As Float
Set A = X^2
Write "The area of the square is:"
Write A
End Subprogram
```

```
Subprogram rectangle(Float X, Float Y)
Declare A As Float
Set A = X * Y
Write "The area of the rectangle is:"
Write A
End Subprogram
```

```
Subprogram Triangle(Float X, Float Y)
Declare A As Float
Set A = * X* Y
Write "The area of the triangle is:"
Write A
End Subprogram
```

2

Apr.12, Oct.10 – 4M

3. Write a Menu Driven Program in VB for calculating:
- i. Area of Circle
 - ii. Area of Rectangle

Solution

```
Option Explicit
Dim r as integer, l as Integer, b as Integer, area as Integer
Private Sub mnuCircle_Click()
    r = InputBox("Enter Radius: ")
    area = 3.14*r*r
    MsgBox "The area of circle is: "& area
End Sub
Private Sub mnuRectangle_Click()
    l = InputBox("Enter Length of Rectangle:")
    b = InputBox("Enter Breadth of Rectangle:")
    area = l * b
    MsgBox "The area of Rectangle is: "& area
End Sub
Private Sub mnuExit_Click()
```

```
Unload Me
End Sub
```

4. Write a VB Program to display even numbers from an Array.

1
Apr.2012 - 4M

Solution

```
Option Explicit
Dim num(1 To 10) As Integer
Dim i As Integer
Private Sub cmdArray_Click()
For i=1 To 10
    num(i)=InputBox("Enter an integer number")
    If(num(i)mod 2=0) Then
        Print num(i)
    End If
Next i
End Sub
```

5. Write a menu driven program in VB for:

- | | |
|----------------------------|------------------------|
| i. Addition | ii. Subtraction |
| iii. Multiplication | iv. Division |

2
Oct.11, Apr.10 - 4M

Solution

```
Option Explicit
Dim x, y, z
Private Sub mnuAdd_Click()
    x = InputBox("Enter first number: ")
    y = InputBox("Enter first number: ")
    z = x + y
    Print "Addition of two numbers is: " & z
End Sub
Private Sub mnuDiv_Click()
    x = InputBox("Enter first number: ")
    y = InputBox("Enter first number: ")
    z = x / y
    Print "Division of two numbers is: " & z
End Sub
Private Sub mnumul_Click()
    x = InputBox("Enter first number: ")
    y = InputBox("Enter first number: ")
    z = x * y
```

```
Print "Multiplication of two numbers is: " & z
End Sub
Private Sub mnusub_Click()
    x = InputBox("Enter first number: ")
    y = InputBox("Enter first number: ")
    z = x - y
    Print "Subtraction of two numbers is: " & z
End Sub
```



PU Questions

2 Marks

[Apr.12,Oct.12 – 4M]

1. Explain the property settings: To Hide Data Control of Runtime.

4 Marks

[Apr.2013 – 4M]

1. Write Short note on: MDI

[Apr.2013 – 4M]

2. Write Short note on: Popup menu

[Apr.2013 – 4M]

3. Write Short note on: Progress Bar

[Apr.2013 – 4M]

4. Write a menu driven program for:

i. Area of Circle ii. Area of Rectangle

[Oct.2012 – 4M]

5. Explain the following property settings:

i. Property used to select *.doc files from a file list box control.

ii. Property used to specify the high end range of the scroll bar control.

[Oct.2012 – 4M]

6. Write Short note on: List View Control

[Oct.2012 – 4M]

7. Write Short note on: Status Bar

[Oct.2012 – 4M]

8. Write Short note on: MDI

[Oct.2012 – 4M]

9. How to create a Menu? Explain with an example.

10. Write a menu driven program in VB to perform the following: **[Oct.12, Apr.11 – 4M]**
i. Area of Square
ii. Area of Rectangle
iii. Area of Triangle
11. Explain Message Box with Syntax and example. **[Apr.2012 – 4M]**
12. What is Menu? How to create Menus using Menu Editor? **[Apr.2012 – 4M]**
13. Write a menu driven program for: **[Oct.10, Apr.13 – 4M]**
i. Area of Circle.
ii. Area of Rectangle.
14. Write a VB Program to display even numbers from an Array. **[Apr.2012 – 4M]**
15. Write a VB Program to display age in year, month and days. **[Apr.2012 – 4M]**
16. Write a menu driven program in VB for: **[Oct.2011 – 4M]**
i. Addition ii. Subtraction
iii. Multiplication iv. Division
17. Write a short note: Tool Bar **[Oct.2011 – 4M]**
18. Write a short note: Menus in Visual Basic **[Oct.11,10 – 4M]**
19. Explain ImageList Control in detail. **[Apr.2011 – 4M]**
20. Write short note on: Tabstrip Control **[Apr.2011 – 4M]**
21. Differentiate between Simple Form and MDI Form. **[Apr.2011 – 4M]**
22. Write short note on: Status Bar **[Oct.2010 – 4M]**
23. Explain briefly MDI Form. How it differs from simple Form? **[Oct.2010 – 4M]**

8 Marks

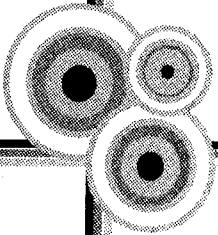
1. Write short notes **[Apr.2012 – 8M]**
a. Predefined Dialog Boxes. b. Progress Bar
2. Write short notes on: Predefined Dialog Box and Progress Bar. **[Apr.2010 – 8M]**
3. Write short notes on: MDI and Popup Menu. **[Apr.2010 – 8M]**
4. How do you create menus in Visual Basic? **[Oct.2010 – 8M]**

10 Marks**[Apr.2012 – 10M]**

1. Explain the following property settings:
 - a. Property to Set Font Size using Common Dialog Box.
 - b. To Status Bar for your program.
 - c. Property used to Set Timer Control.
 - d. Property used to place a picture on a Command Button.
 - e. To resize Image Control.



WORKING WITH DATABASE



1. Introduction

A database is a collection of information that is organized so that it can easily be accessed, managed, and updated. It is a structured collection of records or data that is stored in a computer system. In simple terms we can say that database is a collection of information. The most common example of a database is a phone book, which is a collection of names, addresses, and phone numbers. Each line in a phone book is a record that contains the information for a single person or family. The entire set of records that is, all the listings in the book is a table. Another important characteristic a phone book has in common with most databases is that information is presented in a specific order in the case of the phone book, alphabetically by last name.

Databases are similar to phone books in that they provide a way to store and retrieve information easily and quickly. You can also show relationship in database. Those types of databases are called as relational databases. The language used for accessing, updating, modifying the databases is Structured Query Language (SQL). It is a standard language through which you can write queries for database management systems such as Microsoft's Access and database products from Oracle, Sybase etc.

In this unit we will understand how to handle database through Visual Basic. Visual Basic has standard controls which are used to perform database operations. In this unit you will learn how to connect database and access, manipulate the records through data control and ADO control. It is considered that you already know how to create database. Here in this unit we will directly start with using that database.

2. Data Control

Data control is available on VB toolbar. It provides an interface for navigating data. You can move around in a database from record to record and display and manipulate data from the records in bound controls. This control displays a set of arrow buttons the user can manipulate to move through a data base and the records from that database are displayed in bound controls. You can perform most data access operations using the data control without writing any code. The data controls database and recordset properties refer to those database and recordset objects and you can manipulate the data using those properties. *For example*, if you have SQL statement to execute, you place that statement in the data controls record source property and the result appears in the recordset property. To connect a data control to a database you just set the data control's database name property to the path and name of the access / jet database files you want to open. In record source property select the table you want to work with.

The *Data* control also provides an interface for navigating data, with buttons for moving back and forth through rows in a database table. Visual Basic 2005 has an equivalent control, the Binding Navigator control, which also contains buttons for adding and deleting rows. (*Figure 5.2*)

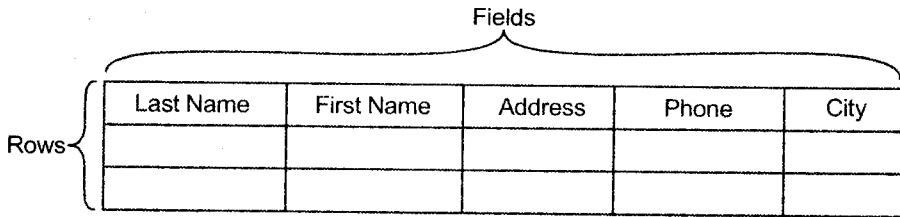


Figure 5.1

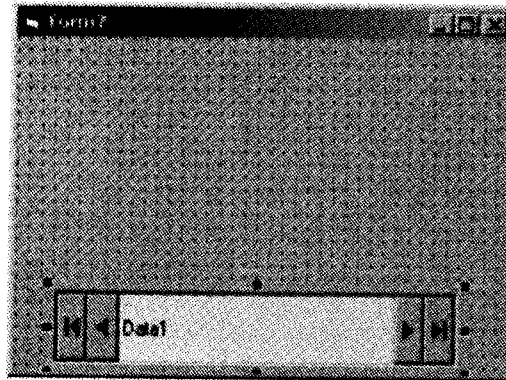


Figure 5.2: Form with Data Control

2.1 Studying the Properties and Methods of Data Control

There are many properties of the data controls as follows:

1. **Connect:** It specifies the type of database. The default database is Microsoft Access. The connect value is usually the name of the data file type.
 - a. *RecordsetType:* A *Recordset* object represents the records in a base table or the records that result from running a query. You use *Recordset* objects to manipulate data in a database at the record level.

Recordset is of three types Table (0), Dynaset (1) and Snapshot (2).

In *Table-type* Recordset you can add, change, or delete records from a single database table.

Dynaset-type Recordset is the result of a query that can have updatable records. A dynaset-type Recordset object is a dynamic set of records that you can use to add, change, or delete records from an underlying database table or tables. A dynaset-type Recordset object can contain fields from one or more tables in a database.

Snapshot-type Recordset is a static copy of a set of records that you can use to find data or generate reports. A snapshot-type Recordset object can contain fields from one or more tables in a database but can't be updated.

2. **Exclusive:** This property has Boolean value i.e. true or false. That indicates whether the underlying database for a Data control is opened for single-user or multi-user access. If it is *true* the database is open for single-user access. No one else can open the database until it's closed or if it is *False* (Default) The database is open for multi-user access. Other users can open the database and have access to the data while it's open.
3. **BOFAction, EOFAction:** BOF stands for Beginning of the File and EOF stands for End of File. These properties determine what happens when the data control has taken you to the beginning and end of the database. The choices you have are to stay at the beginning or move to the first record or actually add new record when you are at the end.
4. **Default Type:** This property specifies whether the JET engine or ODBC model is used.

When you add the controls on the form you may require changing the data in the databases. Therefore we'll use a textbox for each of the fields so that we can both display and enter data as needed. Each textbox will be a bound control, i.e. it is bound to a specific field from the database. When we navigate through the database using the arrow buttons the content of each textbox will always reflect the content of the current field. To bind the control to the database field you need to set some properties of the data control, they are as follows:

- i. *DataSource* is the name of the Data Control. Remember that the DC specifies the name of the database to use and the name of the table to access.

- ii *DataField* is the name of the field to bind. That field is selected from the content of the table.
- iii. *DatabaseName*: It names the location of source of data for a data control.

There are many methods of data control recordset that are discussed as follows:

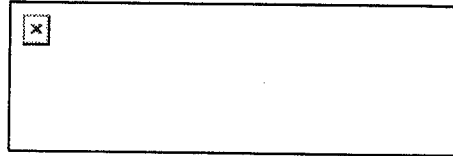


Figure 5.3 Data Control with 4 buttons

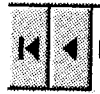


Figure 5.4 : First record and Previous

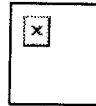


Figure 5.5 : Next and Last record

Four methods are connected with these buttons. Following methods are used to navigate records through code:

1. **MoveFirst**: Moves the pointer at first record from current location.
2. **MovePrevious**: Moves the pointer at previous record from current location.
3. **MoveNext**: Moves the pointer at Next record from current location.
4. **MoveLast**: Moves the pointer at last record from current location.

There are some methods that are used to add, delete, and search the records in the database:

- i. **AddNew**: This method is used to add a new record in the table.

Syntax

```
Data1.Recordset.AddNew
```

- ii. **Delete**: This method is used to delete current record from the table. You should use delete with MoveNext method. So that the user can understand the record is deleted. If you have deleted last record you can either move cursor on to the first record or display message about end of the record.

Syntax

```
Data1.Recordset.Delete
```

For example,

```
Data1.Recordset.Delete
```

```
Data1.Recordset.MoveNext
```

- iii. **Search:** There are four methods used for searching. Those are *FindFirst*, *FindNext*, *FindPrevious* and *FindLast*. These methods can be used to search any field in the recordset for a specific record.

Syntax

```
DataControl.Recordset.FindFirst "fieldname = 'searchstring'"
```

2.2 Connectivity with MS-Access and Operations of Database through Coding

Visual basic allows us to manage databases created with different database programs such as MS Access, Dbase, Paradox etc. In this unit we are not dealing with how to create database files but we will see how we can access their database files in VB. In following example, we will create a simple database application which enables one to browse Author's details. In this example we will see how you can perform different operations in database through coding. To create this application, insert the data control into the new form. Place the data control somewhere at the bottom of the form. To be able to use the data control, we need to connect it to any database. We can create a database file using any database application. Here in this example we will use the database files BIBLIO.MDB that comes with VB6.

Follow the steps given below:

- Step 1:** Open new VB project and start designing the form as shown in the *Figure 5.6*. In this example display a data control on the form (Data1), two labels, and four text boxes.

Set the properties of the data control *Data1*

Property	Value
Connect	Access (this is default).
DatabaseName	C:\Program Files\Microsoft Visual Studio\VB98\BIBLIO.MDB (Select any of the data base available on your system or create new).
RecordSource	Authors (Select any of the databases from the list).

Set the properties of the *Text1*

Property	Value
Data Source	Data1
DataField	AU_ID (Select field from the list)
Font	Bold ,12

Set the properties of the *Text2*

Property	Value
Data Source	Data1
DataField	Author (Select field from the list)
Font	Bold ,12

Set the properties of both the *labels*

Property	Value
Caption	Author Id for Label1 and Name for Label2
Autosize	True
Font	Bold ,12

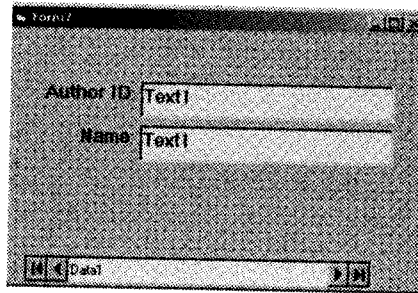


Figure 5.6

Step 2: Run this form now. You can see the records displayed in the text boxes and you can navigate the records using the arrow buttons of the data control.

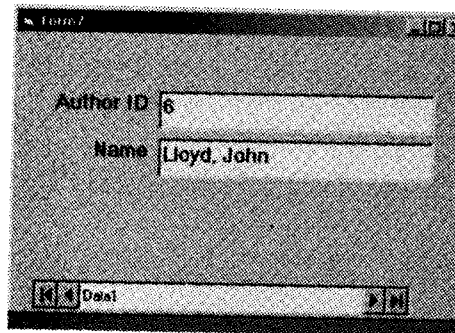


Figure 5.7: Run form showing Author record

Step 3: Now add command buttons to the form as shown in the *Figure 5.8*. Set the visible property of data control false so that it is not visible on the form at run time. Write the following code for the command buttons.

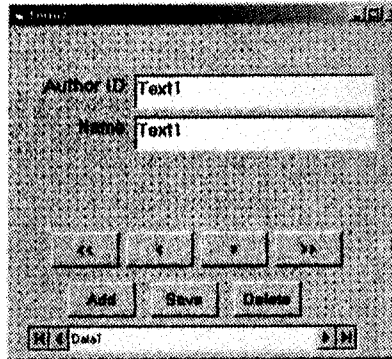


Figure 5.8: Adding command buttons on the form

```

Private Sub cmdAdd_Click()
    Data1.Recordset.AddNew
    Text2.SetFocus
End Sub
Private Sub cmdDelete_Click()
    Ans = MsgBox ("Are you sure you want to delete the
record?", vbYesNo, "Note")
    If Ans = vbYes Then
        Data1.Recordset.Delete
        Data1.Recordset.MoveNext
        If Data1.Recordset.EOF Then
            MsgBox "No more records in the database"
        End If
    End If
End Sub
Private Sub cmdFirst_Click(Index As Integer)
    Data1.Recordset.MoveFirst
End Sub
Private Sub cmdLast_Click()
    Data1.Recordset.MoveLast
End Sub
Private Sub cmdNext_Click()
    Data1.Recordset.MoveNext
    If Data1.Recordset.EOF Then
        Data1.Recordset.MoveFirst
    End If
End Sub
Private Sub cmdPrev_Click(Index As Integer)
    Data1.Recordset.MovePrevious
    If Data1.Recordset.BOF Then
        Data1.Recordset.MoveLast
    End If
End Sub
Private Sub cmdsave_Click()
    Data1.Recordset.Update
End Sub

```

Step 4: Save the project and run the form.

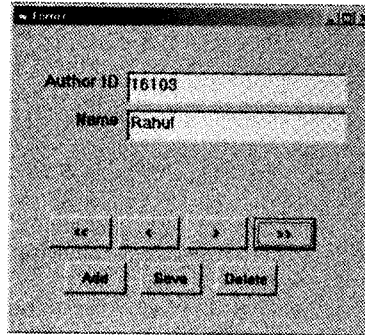


Figure 5.9

3. ADO Data Control

2

Apr. 2010–4M

Compare ADO and ADODC controls.

Apr. 2011–4M

Explain steps to connect MS-Access Database to ADO control.

Apr. 2012 – 4M

Explain ADO Data control.

Apr 2013 – 4M

Explain briefly ADO control.

Another database access control in Visual basic is the ActiveX Data Object Data Control (ADODC). This control lets you access data in a database server through any OLE DB provider. ADO gives you a consistent interface for working with a wide variety of data sources from text files to ODBC relational databases. This control is similar to Data control which we have discussed in the above section. ADO Data Control consists of three objects which are necessary to establish connection.

1. **Connection String:** Establish the connection by setting the *ConnectionString* property.
2. **Data command:** It defines access to database objects such as Tables, stored procedures or SQL queries.
3. **Recordset:** Specify how to derive a Recordset by setting the *RecordSource* property.

ADO is OLE support object. OLEDB is a set of COM interfaces that provide applications with uniform access to data stored in different types of data source. ADO can be used to access unstructured data.

ActiveX data control by combining with data bound control like DataGrid or Bound Control can build quick interface for working with a database.

The difference between Data control and ADO is that the DC uses Data AccessObjects technology to connect to a database, whereas ADO uses.

ActiveX DataObjects, which is a newer technology and generally regarded as better than DAO. It is advisable to use ADO technology as it is newer and better than DAO.

The detailed steps are as follows:

Step 1: Add the Microsoft ADO DataControl 6.0 (OLEDB) from the Project → Components menu dialog box, (as shown in the *Figure 5.10*). Now you can see the ADO Data Control icon in the VB toolbox.

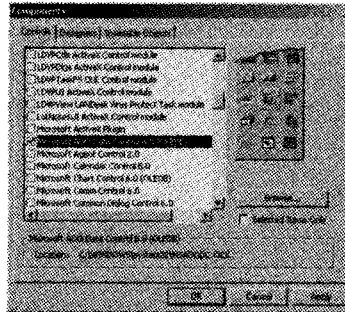


Figure 5.10: Adding ADO data Control

Step 2: Display ADO Data Control on the form (*Figure 5.11*).

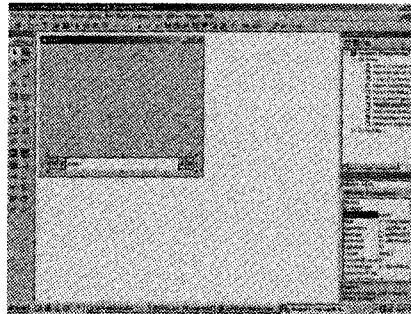


Figure 5.11: ADO data control on the form

Step 3: Change the Name and Caption property of the ADO control.

Step 4: Now follow the step 5-9 to set the connectionString property.

Step 5: Right click the ADO control and select the last option i.e. Properties. This will open the properties page of the ADO control. (*Figure 5.12*)

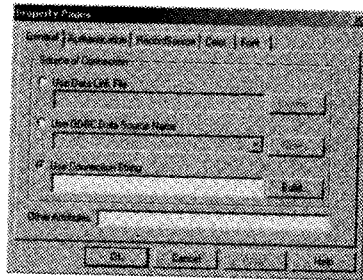


Figure 5.12: The first and only Property Page dialog box for the ADO Data Control's Connection string property

As Source of Connection, choose one of the following three options

- Use Data Link File:* If you choose this option, you will be able to click the Browse button to specify an existing *.UDL file.
- Use ODBC Data Source Name:* This option enables you to choose an existing ODBC DSN from the drop-down list, or you can create a new DSN by clicking the New button.
- Use Connection String:* This option allows you to click the Build button to bring up the Data Link Properties tabbed dialog box.

In our case we will select third option i.e. Use Connection string and click on the Build... button. This will open the Data Link Properties Page. On the *Provider* tab of the Data Link Properties tabbed dialog box, choose an OLE DB data provider, such as Microsoft Jet 3.51 OLE DB (Figure 5.13).

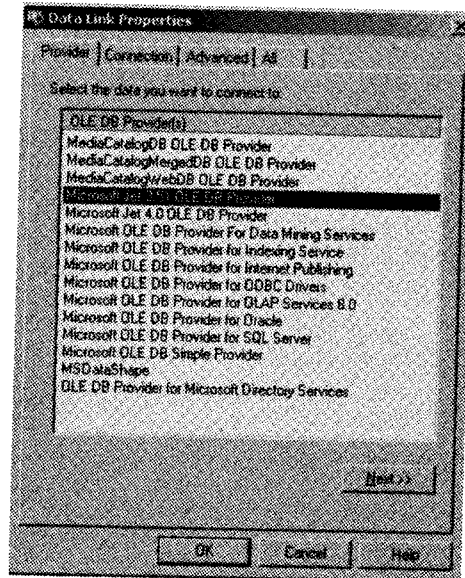


Figure 5.13

Step 6: The *Connection* tab of the Data Link Properties tabbed dialog box selects the database name. In our case we will select Access database (*Figure 5.14*). Select BIBLIO.MDB database which is available with VB6.

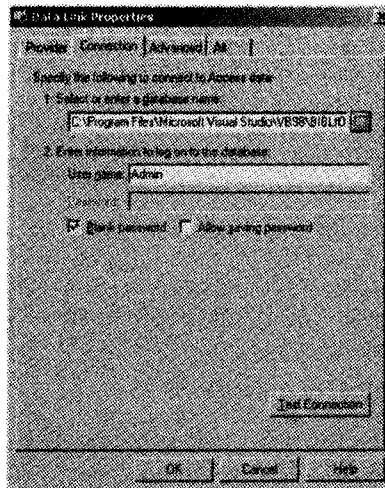


Figure 5.14: Setting connection through Jet provider

Step 7: Now you have built `ConnectionString`; Click OK to accept it. You can check the connection by clicking Test Connection button. If every thing is ok it displays one message "Test Connection Succeeded". This means you have successfully established the connection.

Step 8: Now in the ADO Data Control's Properties window, select the *RecordSource* property. Select the command type from the list given with command Type (the list displays the different options like `adCmdUnknown`, `adCmdText`, `adCmdTable`, `adCmdStoredProc`). (*Figure 5.14*) In this example select `adCmdTable`. (If you select `adCmdUnknown` or `adCmdText` command types you need to write specify SQL query to access records whereas in case of `adCmdStoredProc` select the stored procedure if it is.)

Step 9: The Select Table or Procedure name list box automatically displays the table list available in the database. In our example we have selected VB6 BIBLIO.MDB database. Therefore the list will show all the tables available in this database. Select Authors Table.

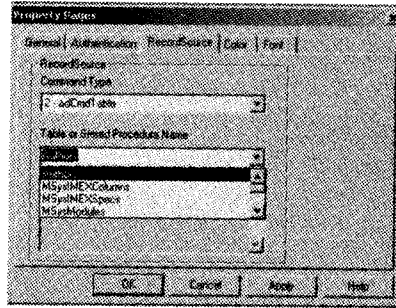


Figure 5.15: property page dialog box for ADO control

Step 10: Click OK to close the RecordSource dialog box. This way you have set the ADO Data Control to a Recordset, now you can bind VB controls to the ADO Data Control, as we have discussed in the above section in data control.

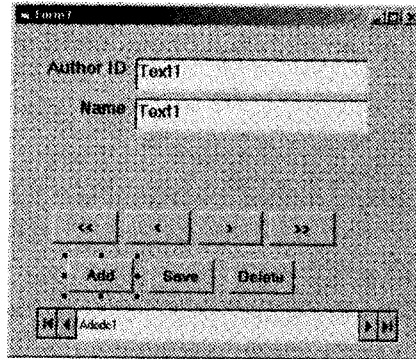


Figure 5.16

Step 11: Design the form as shown in the *Figure 5.16*. Write the following code.

```
Private Sub cmdAdd_Click()
    Adodc1.Recordset.AddNew
    Text2.SetFocus
End Sub

Private Sub cmdDelete_Click()
    Ans = MsgBox ("Are you sure You want to delete the record?",
vbYesNo, "Note")
    If Ans = vbYes Then
        Adodc1.Recordset.Delete
        Adodc1.Recordset.MoveNext
        If Adodc1.Recordset.EOF Then
            Adodc1.Recordset.MoveFirst
        End If
    End If
End Sub
```

```
End If
End Sub
Private Sub cmdFirst_Click(Index As Integer)
    Adodc1.Recordset.MoveFirst
End Sub
Private Sub cmdLast_Click()
    Adodc1.Recordset.MoveLast
End Sub
Private Sub cmdnext_Click ()
    Adodc1.Recordset.MoveNext
    If Adodc1.Recordset.EOF Then
        Adodc1.Recordset.MoveFirst
    End If
End Sub
Private Sub cmdPrev_Click(Index As Integer)
    Adodc1.Recordset.MovePrevious
    If Adodc1.Recordset.BOF Then
        Adodc1.Recordset.MoveLast
    End If
End Sub
Private Sub cmdSave_Click()
    Adodc1.Recordset.Fields("Au_ID") = Text1.Text
    Adodc1.Recordset.Fields("Author") = Text2.Text
    Adodc1.Recordset.Update
End Sub
```

Step 12: Save the project and check all the operations written for the buttons.

3.1 Connecting with Oracle

In the previous session we have discussed how to connect to the MS access database in VB6 and perform different operations. Now we will see how to establish connection with Oracle. There are two ways to get connected with oracle and get the records. One is through ADODC control and other is without ADO control i.e. through coding. Here in the following session we will discuss both the ways of connecting to Oracle. Since we need User DSN (Data Source Name) in both the cases first we will understand how to create Data Source Name.

If you are connecting to SQL Server you'll need DSN. ODBC has three different types of DSNs:

1. User DSN
2. System DSN
3. File DSN

Each DSN category serves a specific purpose and has a specific scope:

1. **User DSN:** A user DSN is for a specific user. If you create a user DSN under user account, no other user can see it or use it. If you need a connection to a data source that only you should use, create a user DSN.
2. **System DSN:** A system DSN is seen by the entire system. Any user, process or service can see it. If you need a data source connection that should be seen more than just your user account, choose to use a system DSN.
3. **File DSN:** A file DSN is the connection settings written to a file. A file DSN is useful if you want to distribute a data source connection to multiple users on different systems without having to configure a DSN for each system.

Here we will see how to create User DSN, but the process is basically the same for a file or system DSN. The only difference with a file DSN is you'll be prompted to save the DSN.

Follow the steps given below:

Step 1: Open the *Control Panel* → Select *Administrative Tools* → Open the *Data Sources (ODBC)* it will popup one window where you can choose appropriate DSN you want to create. Here we will select User DSN Tab. Click on the *Add...* button as shown in the following figure.

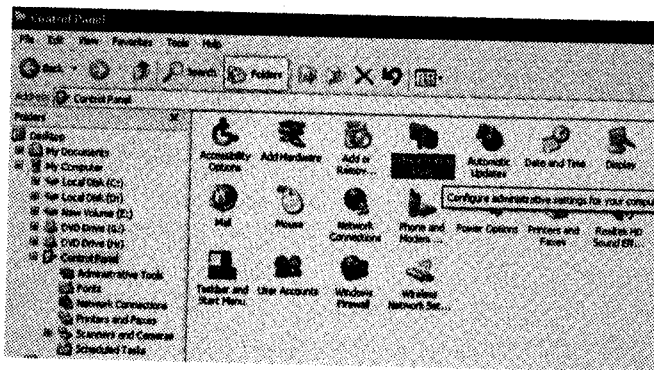


Figure 5.17

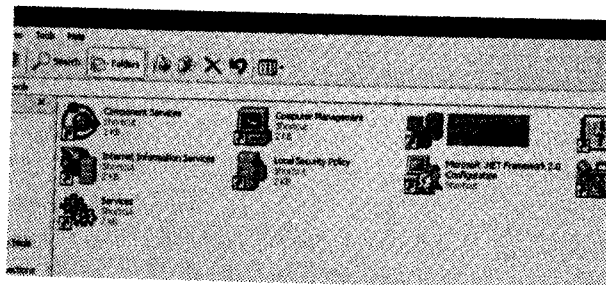


Figure 5.18

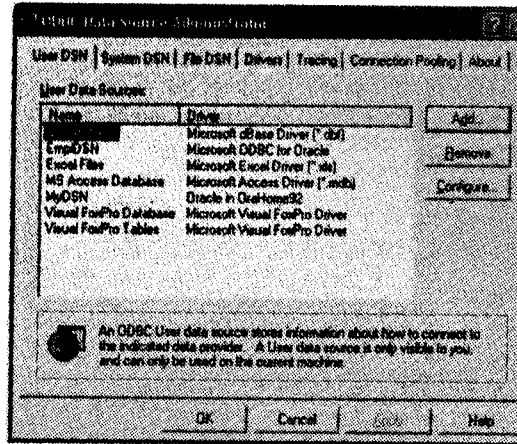


Figure 5.19

Step 2: Now select the proper driver. Since we are connecting to oracle data base we will choose Oracle in OraHome92 driver. As shown in the following figure. And select Finish Button to end the process.

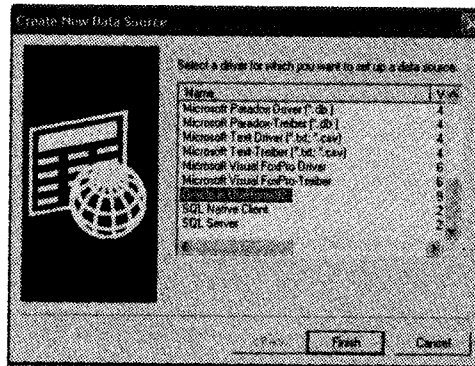


Figure 5.20

Step 3: Once you select the driver it will ask you to give some name to your DSN. Here "MyDSN" "is the name of our User DSN. Now Click Test Connection Button to verify the connection. It will ask for Service Name, User Name and Password. Once you give that value select OK button. It shows the "Connection Successful" message box. If this message box is displayed it means you have successfully created User DSN. And now you can use it.

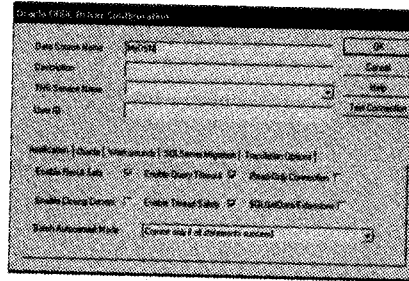


Figure 5.21

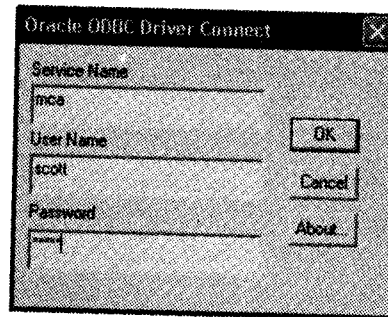


Figure 5.22

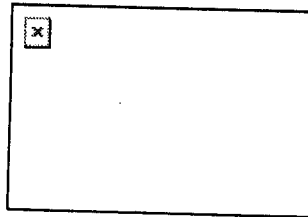


Figure 5.23

Now we will discuss the ADO Data Control connectivity without using ADO Data Control. In this session, we will create an ADO Connection Object and then use an ADO Command Object to return the results of a query to an ADO Recordset object and without the ADO Data Control we'll populate the records in a DataGrid.

Follow the steps given below:

1. In order to use ADO Objects we have to set a reference to the ADO Object Library. To add a reference, select *Project* → *References* from the Visual Basic Menu, then select *Microsoft ActiveX Data Objects 2.6 library*. Object libraries do not appear in the components as they do not have visible interface. Click on Ok button.

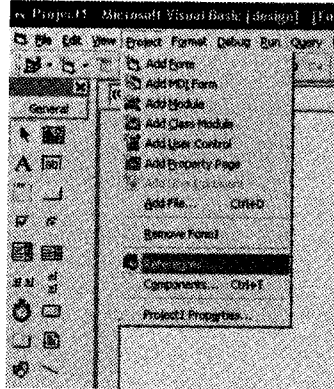


Figure 5.24

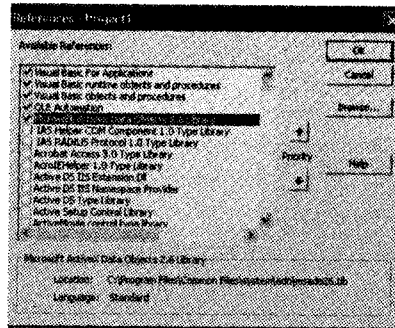


Figure 5.25

- Now we will add Data Grid control to our form. Select Project → Components menu and select Microsoft DataGrid Control 6.0. Click on Ok button.

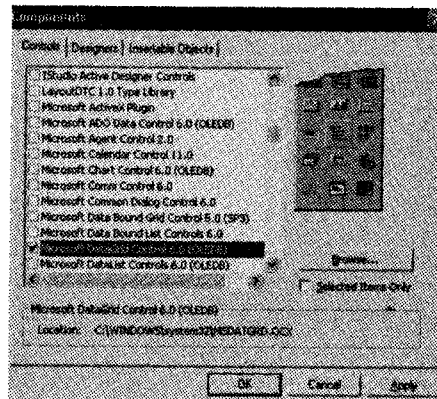


Figure 5.26

The DataGrid control will be added to the toolbox as shown in the following figure.

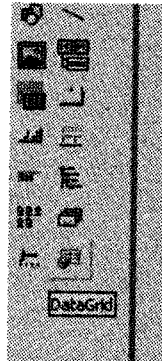


Figure 5.27

3. Add the DataGrid control on your form and place the following code on the form load event. Save the form and run.

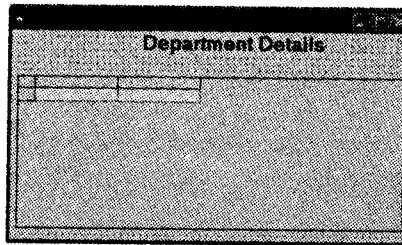


Figure 5.28

```
Private Sub Form_Load ()
    Dim conn As New ADODB.Connection
    Dim rs As New ADODB.Recordset
    Dim strSQL As String
    strSQL = "SELECT * FROM DEPT"
    Set conn = New ADODB.Connection
    conn.Open "Provider=MSDAORA.1; User ID=scott; Data
    Source=mca;Persist Security
    Info=False; password=tiger"
    rs.CursorType = adOpenStatic
    rs.CursorLocation = adUseClient
    rs.LockType = adLockOptimistic
    rs.Open strSQL, conn, , , adCmdText
    Set DataGrid1.DataSource = rs
End Sub
```

Let's understand the code.

- i. These first two lines of code declare two object variables. The first is an ADO Connection Object and second is an ADO Recordset Object

```
Dim conn As New ADODB.Connection
Dim rs As New ADODB.Recordset
```

- ii. These declarations will build the Recordset which is a virtual representation of an actual Oracle table in our Database.

```
Dim strSQL As String
```

strSQL variable in the above line is a String variable that we will use to store the SQL statement used to build our Recordset. Once you declared the strSQL variable, you can assign a SQL statement to it. It is used to retrieve every record from the DEPT table in Oracle database.

```
strSQL = "SELECT * FROM DEPT"
```

- iii. The next statement will open ADO Connection. We do that by executing the Open method of our Connection Object. The Open method requires four parameters: the Provider name, the Data Source (or HostName), the User ID and the Password of the database.

```
conn.Open "Provider=MSDAORA.1; User ID=scott;
Data Source=mca;Persist Security
```

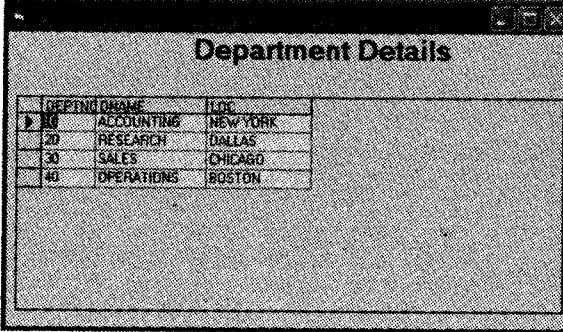
Note: The Connection String that the ADO Data Control wizard built for us could be used to open a Connection in code. You can use the ADO Data Control wizard to build Connection Strings, and then copy and paste them into the code window. Before we build the Recordset object we need to adjust three properties of the Recordset object, the CursorType, CursorLocation and LockType.

```
rs.CursorType = adOpenStatic
rs.CursorLocation = adUseClient
rs.LockType = adLockOptimistic
rs.Open strSQL, conn, , , adCmdText
```

- iv. We now have a Recordset object built containing all of the data in the DEPT table of Oracle database. We can use the Set statement to assign the Recordset object to the DataSource property of DataGrid.

```
Set DataGrid1.DataSource = rs
```

- v. If we now run the program, the code in the Load Event procedure of the form executes. A Connection object is created, initiating the Connection to Oracle Database. Then a Recordset object is created, retrieving DEPT Table's records. Finally, the DataSource property of the DataGrid is set to point to the Recordset object. The following form is the result of this entire process.



OFFNO	DEPARTMENT	JOB
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Figure 5.29

This result we have achieved without using ADO data Control. You can also use ADO data control and display data.

Follow the steps given below:

1. Select Project → Component Menu and Select Microsoft ADO Data Control 6.0. This will add the ADO control on the tool box as shown in the following figure.



Figure 5.30

2. Now display the Data Grid and ADO control on the form. The screen should look like following figure.

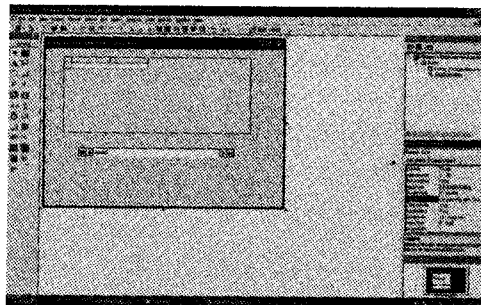


Figure 5.31

3. At this step you need to set some properties of the Data Grid and ADO control. First we will set ADO data control properties. Right click on ADODC control to open ADODC property page.

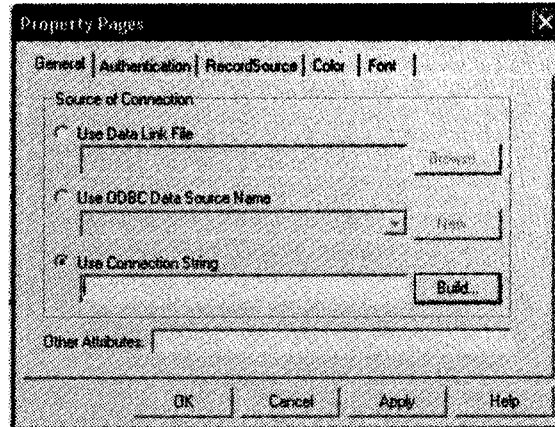


Figure 5.32

4. You can see three different sources of connection on the General Tab on the Property page. Select Use Connection String option button and click on the **Build...** button to create connection string. After clicking on Build button VB will open Data Link Property page as shown in the following figure. It will ask to choose proper driver Here we will select "Microsoft OLEDB provider for Oracle" drivers. Then click on the Next button.

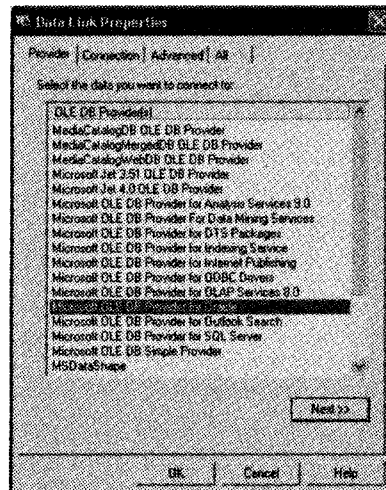


Figure 5.33

5. Now enter Server name, User Name and Password to get connection with oracle. Then click on Test connection button to verify the connection. If the connection is created successfully it shows the "Test Connection Successful" message box.

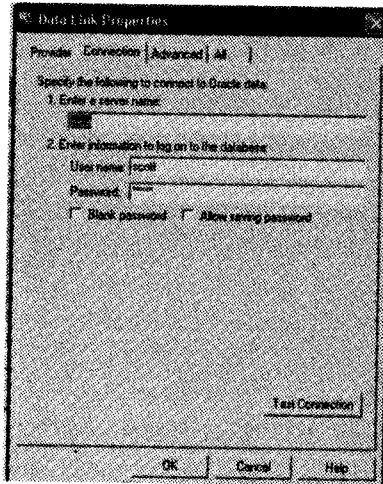


Figure 5.34

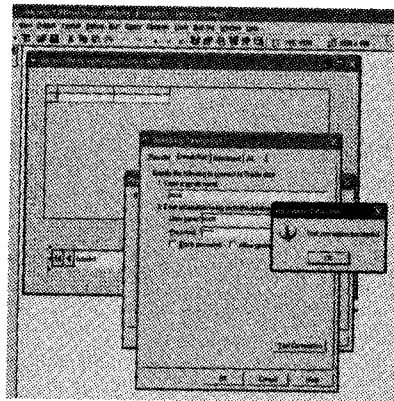


Figure 5.35

6. You have successfully created connection with oracle. Your connection string has been created now. Click on Ok button to return to ADODC property page. You can see the string in the text box with the Use Connection String option button.

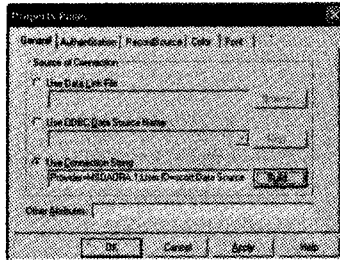


Figure 5.36

- Now we have access to oracle database. Select RecordSource tab on the Property page. This will ask for the command Type. Select 2-adCmdTable command type so that you can see the list of oracle tables created in your account.

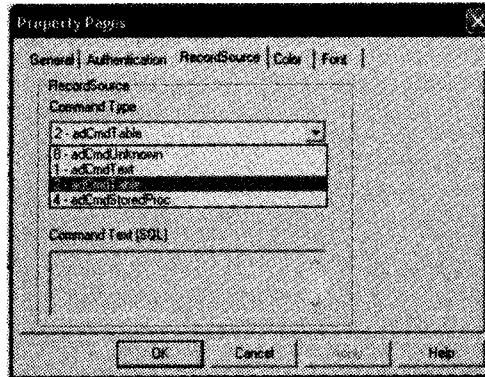


Figure 5.37

- Once you select Table command type, all the available table will be listed in the Table or Stored Procedure Name list box. As shown in the following figure. Select any table. In this example I have selected DEPT table and click on the Ok button to finish the procedure.

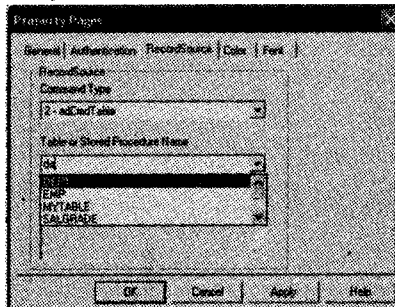


Figure 5.38

9. Now we have to set the properties of Data grid control. We have already displayed Data Grid control on the form. Set the Data Source property of Data Grid control. When you select the property it will automatically gives the list of available data source control list on the form. In this example there is only one data source control available on the form i.e. Adodc1. You need to select that option. Save the project and check the operations. It displays the result as shown in the following *figure*.

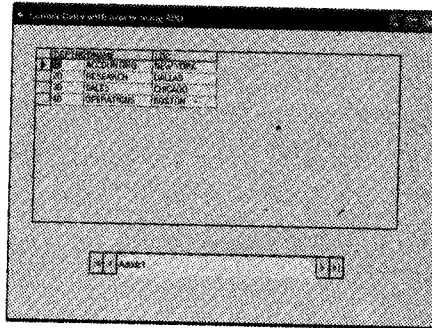


Figure 5:39

3.2 Report Generation

Report is nothing but the formatted and organized presentation of data. Most database management systems include a report writer that enables you to design and generate reports. Data Report facility in Visual Basic allows you to design professional reports in VB. The Data Report Designer creates hierarchical reports. These are the most common type of database report; the reports designed in Data report have headings, subheadings, details, and summaries organized in a hierarchical manner. A Data Report is similar to a VB form in that it has a visual designer and a code module. Using the visual designer, you can divide the report into two or more sections, each with its own headings. Each section can contain controls to display the report details. The design of the details sections is simplified by drag-and-drop functionality. The available controls are distinct from VB controls but have similar functionality. In particular, the Function control lets you easily perform calculations on field data (sum, average, minimum, and maximum) and display the results as the report is generated. Headers and footers can be defined for the report as a whole and for each page of the report. The Data Report Designer is a very useful facility of VB. Its not suitable for every type of report, but when it fits your needs, it can save you a tremendous amount of time. VB Data report designer is very much similar to Microsoft Access.

To understand the data report we will discuss one example. You just need to follow the instruction given below:

The Data Report Designer builds reports from database tables. The example in this chapter uses the BIBLIO.MDB database, which is included in the main Visual Basic directory.

Step 1: From the *Project* menu, select *Data Environment* option.

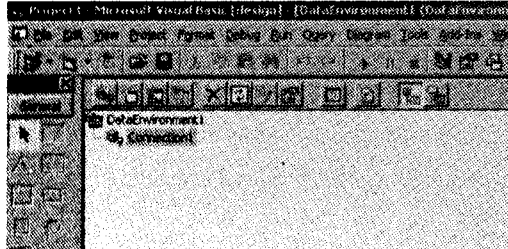


Figure 5.40

Step 2: To make a connection to the BIBLIO.MDB database, right-click *Connection1* and select *Properties* from the popup menu.

Step 3: To connect with Access database, select *Microsoft Jet 3.51 OLE DB Provider* and then click *next*.

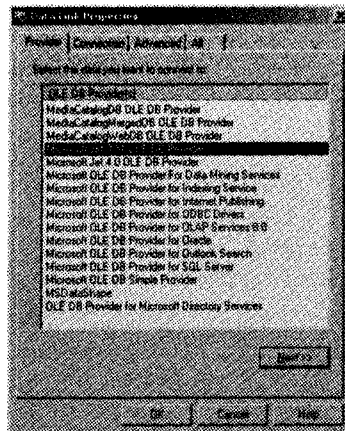


Figure 5.41

Step 4: Enter or select the database filename to use for this connection. Here we will select BIBLIO.MDB database which comes with VB. For Biblio Database no need to set any security option. If you were using another database that had security, you would have to specify a user ID and password.

Apr.11, 13 – 4M

Oct.10, 12 – 4M

Write short note on: Data Reports.

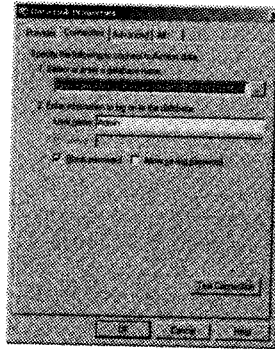


Figure 5.42

Step 5: You can verify the connection by clicking the *Test Connection* button. If you receive a message indicating a successful test, click *OK* to continue. Otherwise, go back and make sure that you followed the steps correctly.

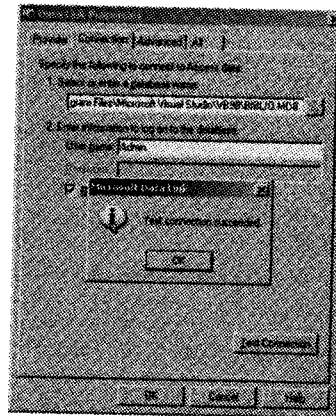


Figure 5.43

Step 6: Rename the Connection1 item, click the *Connection1* item and wait a while and then rename the connection with some meaningful name such as conBiblio. Renaming Connection item helps you when you have many Connections in your data environment. By the name you can easily identify that which data base is connected with the connection item.

Step 7: Same way you can also rename Data Environment also. If you have multiple data environments in your project, it helps you to identify the control.

Now you have data environment created successfully. Save your work. Yet we have not finished with reports. With the data environment created, you can now create the query that will retrieve Author information from the database.

Step 8: Now Select *Project* → *Add Data Report* option. The data report screen looks like following screen shot.

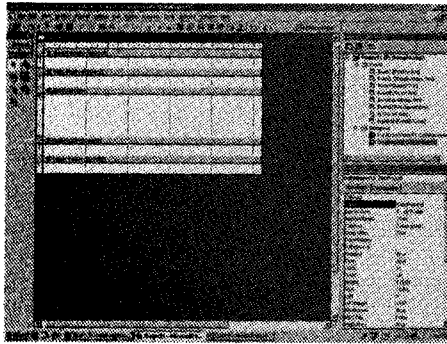


Figure 5.44

Step 9: Give some meaningful title to the report. This title can be both in the Report Header, and on the first page of the report, or in the Page Header and on the top of every report. Add RptLabel control to create a title for your report, or simply draw the RptLabel control where you want it and set the Caption property to change the text that should be displayed.

You can also set the Caption property of the Report itself by using the Properties window when the Report is selected. Also, any blank space you leave around the controls will be repeated whenever the report is shown, so be sure to place the controls correctly and to close up any blank space around them. You can add other labels or graphics to the report as you want.

Step 10: Right-click *Commands* and select *Add Command* from the popup menu. The Data Environment Designer will add a new command to the environment. Right-click it and select *Properties*. It will open command properties page (Figure 5.45). General Tab is selected. Give the command a useful name, such as *cmdAuthor*.

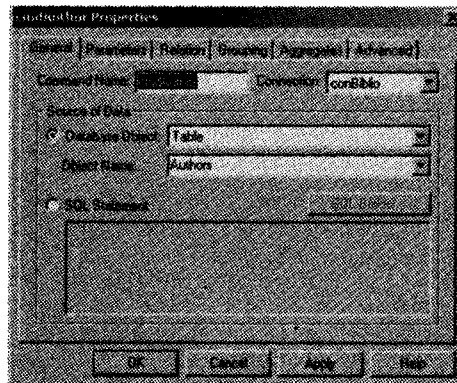


Figure 5.45: Command property page

Step 11: Select the connection *conBiblio* from the *Connection* drop-down list. Now select source of data. You can add table from database object dropdown list and then select the precise table name from object name list. You can use the SQL Builder if you want to create complex joins and don't want to type the SQL yourself. SQL statement helps you to write SQL query, so that you can retrieve your data.

Step 12: Click *OK*. Now after setting command item your screen should look like following screen (*Figure 5.46*). If you explore command i.e. *cmdAuthor* it will show you all the fields of author table. Be sure to save your work because you have now finished building your data connections.

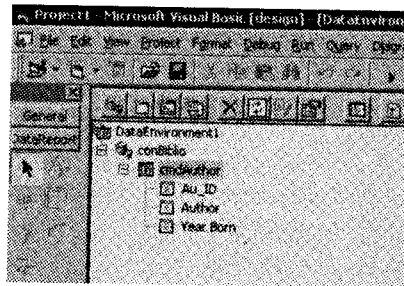


Figure 5.46

Step 13: We have already added Data Report. Now set the following properties of the Data Report.

Data Member: *cmdAuthor* (Command name)

Data Source: *DataEnvironment1* (DataEnvironment name)

If you don't set these properties, you'll receive errors when you try to run the report.

Step 14: To add data to the report, take a field from the Data Environment Designer window, such as *Au_ID*, and drag it to the Detail section of your report. VB will automatically draw a *RptTextBox*, along with a *RptLabel* control on the report. The Detail section is very narrow because any blank space in this section will be repeated for every row in the report. You can decorate your report if you wish. Use line and shape control to elaborate your report. With line control you can draw line. After dragging all the fields in to the data report the screen should look like *Figure 5.47*.

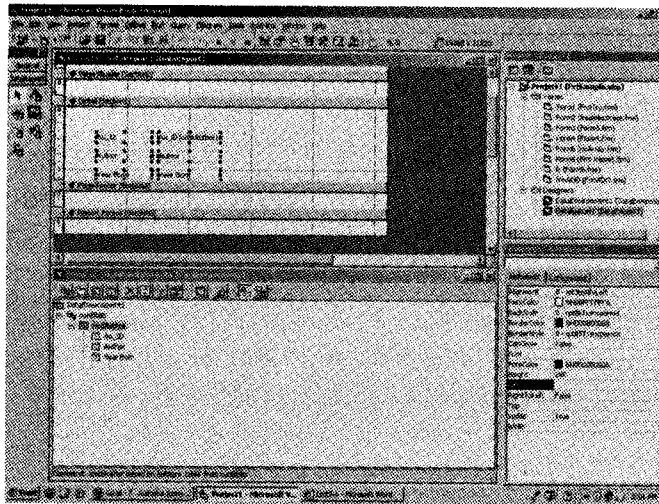


Figure 5.47

Step 15: To make this report column-based, drag the RptLabel into the PageHeader section. Place the RptTextBox beneath the RptLabel but leave the RptTextBox in the Detail section. Repeat this process for all the fields. Now your report should look like *Figure 5.48*.

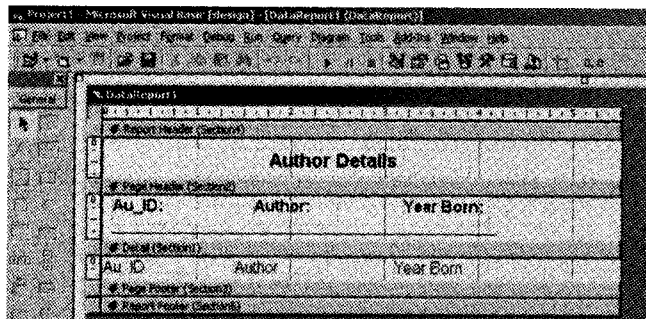


Figure 5.48

Step 16: To include a page number and total number of pages, create page footer. This is very easy to do with RptLabel control, which supports a number of substitutions so that you don't have to write code to put page numbers in your report. The RptLabel control supports these substitutions.

Step 17: Now that the report is complete you can run it. The report is run just like any other form in your project. Choose *Properties* from the *Project* menu and use this report as your startup

form. Or you can display report using code. Use the standard Show method. When you run your program, your report will be displayed as shown in *Figure 5.49*.

You can write following code for the button to run your report.

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

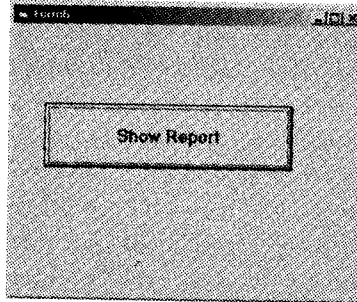


Figure 5.49

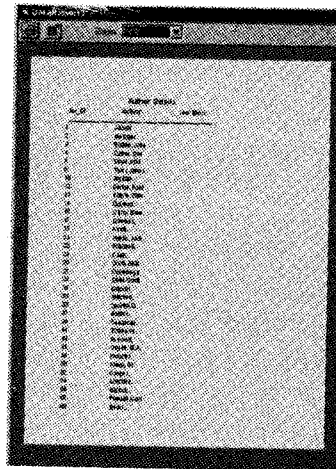


Figure 5.50

4. Developing ADO Application through ADODC and Coding

In this example we will create a small tutor. This tutor has one database and five forms. It is assumed that you know how to create database in Microsoft access or Oracle. We have already discussed the process to connect oracle and access database to the visual basic using ADO data control.

Follow the steps given below to create your tutor

1. Create the following tables in MS Access or oracle database.

Chapter

Cno	Number
ChapName	Text

Exam

Queno	Number
A	Text
B	text
C	
D	
Ans	
UAns	

Topics

No	Number
Topic	Text
Description	text

2. Open Visual Basic IDE and add four forms to your project. Design and write the code for all the forms as shown below:

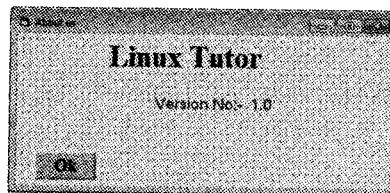


Figure 5.51: About Form

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

```
Private Sub Form_Load()  
    Me.Left = 1500  
    Me.Top = 2000  
End Sub
```

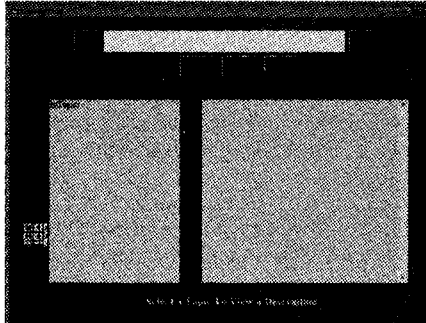


Figure 5.52: Main Form

```
Dim qry As String  
Private Sub cmdAbout_Click()  
    frmAbout.Show  
End Sub  
Private Sub cmdExit_Click()  
    End  
End Sub  
Private Sub cmdNext_Click()  
On Error Resume Next  
    Adodc1.Recordset.MoveNext  
    lblChName.Caption = Adodc1.Recordset.Fields(1)  
    Start  
End Sub  
Private Sub cmdPrevious_Click()  
On Error Resume Next  
    Adodc1.Recordset.MovePrevious  
    lblChName.Caption = Adodc1.Recordset.Fields(1)  
    Start  
End Sub  
Private Sub cmdTest_Click()  
    frmTest.Show  
End Sub  
Private Sub Form_Load()  
    Adodc1.Refresh  
    lblChName.Caption = Adodc1.Recordset.Fields(1)
```



```
    Start
End Sub
Private Sub Start()
    qry = "select * from Topics where ChNo=" &
Adodc1.Recordset.Fields(0) & ""
    Adodc3.RecordSource = qry
    Adodc2.RecordSource = qry
    Adodc2.Refresh
    Adodc3.Refresh
    lstTopics.Clear
    txtDescription.Text = ""
    While Not Adodc2.Recordset.EOF
        lstTopics.AddItem Adodc2.Recordset.Fields(1)
        Adodc2.Recordset.MoveNext
    Wend
End Sub
Private Sub lstTopics_Click()
    Dim c, i As Integer
    c = lstTopics.ListIndex + 1
    Adodc3.Recordset.MoveFirst
    For i = 1 To c - 1
        Adodc3.Recordset.MoveNext
    Next
    txtDescription.Text = Adodc3.Recordset.Fields(2)
End Sub
Private Sub txtDescription_GotFocus()
    lstTopics.SetFocus
End Sub
```

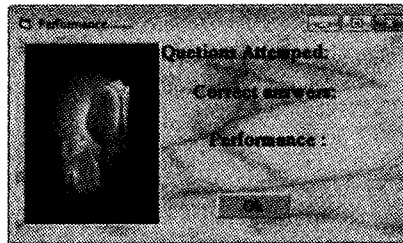


Figure 5.53: Source Form

```
Private Sub cmdOk_Click()
    Unload Me
    Unload frmTest
End Sub
```

```

Private Sub Form_Load()
On Error Resume Next
    Me.Left = 3000
    Me.Top = 3000
    lblQueAttemped.Caption = frmTest.lblQNo1.Caption
    lblCorrectAns.Caption = frmTest.lblScore1.Caption
    lblPerformance.Caption = CInt(frmTest.lblQNo1.Caption)
        * 100 / CInt(frmTest.lblScore1.Caption) & "
End Sub

```

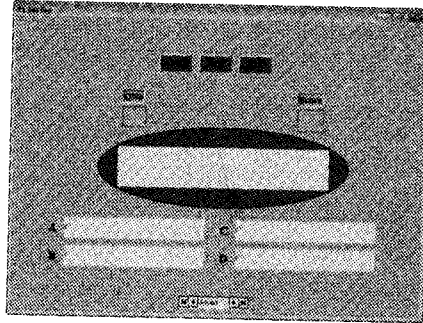


Figure 5.54: Test Form

```

Dim score As Integer
Private Sub Display_Data()
    lblQuestion.Caption = ADO.Recordset.Fields(5)
    lblQNo1.Caption = ADO.Recordset.Fields(0)
    optA.Caption = ADO.Recordset.Fields(1)
    optB.Caption = ADO.Recordset.Fields(2)
    optC.Caption = ADO.Recordset.Fields(3)
    optD.Caption = ADO.Recordset.Fields(4)
End Sub
Private Sub ADO_WillMove(ByVal adReason As ADODB.EventReasonEnum,
adStatus As ADODB.EventStatusEnum, ByVal pRecordset As
ADODB.Recordset)
End Sub
Private Sub cmdAbout_Click()
    frmAbout.Show
End Sub
Private Sub cmdExit_Click()
    If (optA.Value = True And optA.Caption = answer) Or (optB.Value =
True And optB.Caption = answer) Or (optC.Value = True And
optC.Caption = ans) Or (optD.Value = True And optD.Caption = answer)
Then score = score + 1
    lblScore1.Caption = score
    frmScore.Show

```

```

Unload Me
End Sub
Private Sub cmdNext_Click()
    Dim answer As String
    answer = ADo.Recordset.Fields(6)
    If (optA.Value = True And optA.Caption = answer) Or (optB.Value = True
And optB.Caption = answer) Or (optC.Value = True And optC.Caption = ans) Or
(optD.Value = True And optD.Caption = answer) Then score = score + 1
    lblScore1.Caption = score
    ADo.Recordset.MoveNext
    If ADo.Recordset.EOF Then ADo.Recordset.MoveLast
    Display_Data
End Sub
Private Sub Form_Load()
    score = 0
    lblScore1.Caption = score
    ADo.Refresh
    ADo.Refresh
    Display_Data
End Sub

```

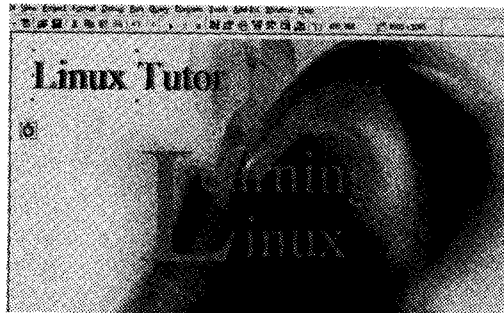


Figure 5.55: Wel-Come Form

```

Private Sub Timer1_Timer()
    Randomize
    lblTitle.ForeColor = QBColor(Rnd * 5)
    Randomize
    lblname.ForeColor = QBColor(Rnd * 5)
End Sub

```

Solved Programs

1. Draw an interface and code for the following. Also give property setting for appropriate controls. Write a program in VB to store a data into the database with the

fields' rollno, name, marks 1, marks 2, marks 3. Calculate average.

Solution

STUDENT RESULT

Roll No

Name

Mark1

Mark2

Mark3 Average

Step 1: Create New form.

Step 2: Display the following controls on the form and set the properties of the control as shown in the following table:

Control Name	Property Name	Value
Label1	caption	STUDENT RESULT
	Font size	18
	Font Bold	True
Label2	Caption	Roll No
Label3	Caption	Name
Label4	Caption	Mark1
Label5	Caption	Mark2
Label6	Caption	Mark3
Text1	Name	Txtrno
Text2	Name	txtname
Text3	Name	Txtm1
Text4	Name	Txtm2
Text5	Name	Txtm3
Command Button1	Name	cmdCalculate
	Caption	Calculate
Command Button2	Name	cmdSave
	Caption	Save
CommandButton3	Name	cmdClear
	Caption	Clear

```
Option Explicit
```

```
Dim avg as Integer
```

```
Private Sub cmdCalculate_Click()
```

```
    Avg = (m1+m2+m3) / 3
```

```
    Label7.caption = avg
```

```
End Sub
```

```
Private Sub cmdSave_Click()
```

```

ADODB.AddNew
ADODB.Recordset.Fields("RNo") = txtno.Text
ADODB.Recordset.Fields("name") = txtname.Text
ADODB.Recordset.Fields("mark1") = txtm1.Text
ADODB.Recordset.Fields("mark2") = txtm2.Text
ADODB.Recordset.Fields("mark3") = txtm3.Text
ADODB.Recordset.Fields("avg") = avg
ADODB.Recordset.Update
End Sub
Private Sub Form_Load()
Dim cnn As New ADODB.Connection
Dim rs As New ADODB.Recordset
Dim strSQL As String
Set cnn = New ADODB.Connection
cnn.Open "Microsoft.Jet.OLEDB.3.51;Persist Security Info=False;Data
Source=C:\Documents and Settings\Administrator\My Documents\TeacherDB.mdb"
rs.CursorType = adOpenDynamic
rs.CursorLocation = adUseClient
rs.LockType = adLockOptimistic
rs.Open strSQL, cnn, , , adCmdText
End Sub
Private Sub cmdClear_Click()
Txtno.text = ""
Txtname.text = ""
Txtm1.text = ""
Txtm2.text = ""
Txtm3.text = ""
End Sub

```

- 2. Write a program in VB to accept product details and store it into the database and display amount. The database fields are: Itemno, Itemname, Rate, Quantity.**

Solution

```

Option Explicit
Dim amount as Integer
Private Sub cmdCalculate_Click()
amount = rate * qty
Label1.caption = amount
End Sub
Private Sub cmdSave_Click()
ADODB.AddNew
ADODB.Recordset.Fields("ItemNo") = txtno.Text
ADODB.Recordset.Fields("Itemname") = txtname.Text
ADODB.Recordset.Fields("rate") = txtrate.Text
ADODB.Recordset.Fields("qty") = txtq.Text
ADODB.Recordset.Fields("amount") = label1.caption
End Sub
Private Sub Form_Load()

```

```

Dim cnn As New ADODB.Connection
Dim rs As New ADODB.Recordset
Dim strSQL As String
Set cnn = New ADODB.Connection
    cnn.Open "Microsoft.Jet.OLEDB.3.51;Persist Security
Info=False;Data Source=C:\Documents and
Settings\Administrator\My Documents\ProductDB.mdb"
    rs.CursorType = adOpenDynamic
rs.CursorLocation = adUseClient
rs.LockType = adLockOptimistic
rs.Open strSQL, cnn, , , adCmdText
End Sub

Private Sub cmdClear_Click()
    Txtno.text = ""
    Txtname.text = ""
    Txtrate.text = ""
    Txtqty.text = ""
End Sub

```

1

Oct.2011 - 8M

3. Write a program to accept the details of students from user and store the details along with total and percentage into the database. (Don't use Standard Control) Students having fields stud_rollno, stud name, stud_mark1, stud_mark2, stud_mark3.

Solution

Step 1: Create New form.

Step 2: Display the following controls on the form and set the properties of the control as shown in the following table:

Control Name	Property Name	Value
Label1	Caption	STUDENT RESULT
	Font size	18
	Font Bold	True
Label2	Caption	Roll No
Label3	Caption	Name
Label4	Caption	Mark1
Label5	Caption	Mark2
Label6	Caption	Mark3
Text1	Name	Txtrno
Text2	Name	txtname
Text3	Name	Txtrm1
Text4	Name	Txtrm2
Text5	Name	Txtrm3
Command Button1	Name	cmdSave
	Caption	Save
CommandButton2	Name	cmdClear
	Caption	Clear

```

Option Explicit
Private Sub cmdSave_Click()
    ADODB.AddNew
    ADODB.Recordset.Fields("RNo")=txtno.Text
    ADODB.Recordset.Fields("name")=txtname.Text
    ADODB.Recordset.Fields("mark1")=txtm1.Text
    ADODB.Recordset.Fields("mark2")=txtm2.Text
    ADODB.Recordset.Fields("mark3")=txtm3.Text
    ADODB.Recordset.Fields("avg")=avg
    ADODB.Recordset.Update
End Sub

Private Sub Form_Load()
    Dim cnn As New ADODB.Connection
    Dim rs As New ADODB.Recordset
    Dim strSQL As String
    Set cnn = New ADODB.Connection
    cnn.Open "Microsoft.Jet.OLEDB.3.51;Persist Security Info =
False;Data Source = C:\Documents and Settings\Administrator\My
Documents\TeacherDB.mdb"
    rs.CursorType = adOpenDynamic
    rs.CursorLocation = adUseClient
    rs.LockType = adLockOptimistic
    rs.Open strSQL, cnn, , , adCmdText
End Sub

Private Sub cmdClear_Click()
    txtno.Text = ""
    txtname.Text = ""
    txtm1.Text = ""
    txtm2.Text = ""
    txtm3.Text = ""
End Sub

```

4. Write a program to accept the details of customer from user and store that details in to the database. (Don't use standard control). Customer having fields custid, custname, custaddress.

Solution

```

Private Sub cmdSave_Click()
    ADODB.AddNew
    ADODB.Recordset.Fields("custid") = txtno.Text
    ADODB.Recordset.Fields("custname") = txtname.Text
    ADODB.Recordset.Fields("custadd") = txtadd.Text
    ADODB.Recordset.Update
End Sub

Private Sub Form_Load()
    Dim cnn As New ADODB.Connection

```

```

Dim rs As New ADODB.Recordset
Dim strSQL As String
Set cnn = New ADODB.Connection
cnn.Open "Microsoft.Jet.OLEDB.3.51;Persist Security Info=False;Data
Source=C:\Documents and Settings\Administrator\My
Documents\CustomerDB.mdb"
rs.CursorType = adOpenDynamic
rs.CursorLocation = adUseClient
rs.LockType = adLockOptimistic
rs.Open strSQL, cnn, , , adCmdText
End Sub

```

1

Apr.2013 - 8M

5. Write a VB program to accept the student details from user and store the details into the database (don't use standard control) student having rollnos, name, class.

Solution

```

Dim C As New Connection
Dim R As New Recordset
Dim S As String
Private Sub cmdAdd_Click()
    txtRno.Text = ""
    txtSname.Text = ""
    txtClass.Text = ""
    txtRno.SetFocus
End Sub

Private Sub cmdNext_Click()
    R.MoveNext
    If Not R.EOF Then

        txtRno.Text = R.Fields(0).Value
        txtSname.Text = R.Fields(1).Value
        txtClass.Text = R.Fields(2).Value
    Else
        MsgBox "No More Records!", vbInformation, "Student"
    End If
End Sub

Private Sub cmdPrev_Click()
    R.MovePrevious
    If Not R.BOF Then
        txtRno.Text = R.Fields(0).Value
        txtSname.Text = R.Fields(1).Value
        txtClass.Text = R.Fields(2).Value
    Else

```



```
MsgBox "No More Records!", vbInformation, "Student"
End If
End Sub
Private Sub cmdSave_Click()
    R.Close
    S = "Insert Into studData Values(" & Val(txtRno.Text) & ",'" &
txtSname.Text & "','" & txtClass.Text & "')"
    R.Open S, C, adOpenDynamic, adLockOptimistic
    S = "Select * From studData"
    R.Open S, C, adOpenDynamic, adLockOptimistic
    If Not R.BOF And Not R.EOF Then
        R.MoveFirst
        txtRno.Text = R.Fields(0).Value
        txtSname.Text = R.Fields(1).Value
        txtClass.Text = R.Fields(2).Value
    End If
MsgBox "Student record Added Successfully!", vbInformation, "Student"
End Sub
Private Sub Form_Load()
    S = "Select * From studData"
    C.Open "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=E:\VBSlipSol\Slip07\Ques-2\stud.mdb;Persist Security Info=False"
    R.Open S, C, adOpenDynamic, adLockOptimistic
    If Not R.BOF And Not R.EOF Then
        R.MoveFirst
        txtRno.Text = R.Fields(0).Value
        txtSname.Text = R.Fields(1).Value
        txtClass.Text = R.Fields(2).Value
    End If
End Sub
```



PU Questions

4 Marks

- | | |
|------------------------|--|
| <u>[Apr.2013 – 4M]</u> | 1. Explain briefly ADO control. |
| <u>[Apr.2013 – 4M]</u> | 2. Write a short note on: Data Reports |
| <u>[Oct.2012 – 4M]</u> | 3. Explain ADO Data Control. |
| <u>[Oct.2012 – 4M]</u> | 4. Write a short note on: Data Reports |
| <u>[Apr.2012 – 4M]</u> | 5. Compare ADO and ADODC Controls. |
| <u>[Apr.2011 – 4M]</u> | 6. Explain steps to connect MS - Access Database to ADO Control. |
| <u>[Apr.2011 – 4M]</u> | 7. Write short note on: Data Control |
| <u>[Apr.2011 – 4M]</u> | 8. Write short note on: Data Reports in VB |
| <u>[Oct.2010 – 4M]</u> | 9. Write short note on Popup Menu |
| <u>[Apr.2010 – 4M]</u> | 10. Compare ADO and ADODC Controls. |

8 Marks

- | | |
|------------------------|---|
| <u>[Apr.2013 – 8M]</u> | 1. Write a VB program to accept the student details from user and store the details into the database (don't use standard control) student having rollnos, name, class. |
| <u>[Apr.2012 – 8M]</u> | 2. Write a program to accept the details of customer from user and store that details in to the database. (Don't use standard control). Customer having fields custid, custname, custaddress. |
| <u>[Oct.2011 – 8M]</u> | 3. Write a program to accept the details of students from user and store the details along with total and percentage into the database. (Don't use Standard Control) Students having fields stud_rollno, stud_name, stud_mark1, stud_mark2, stud_mark3. |
| <u>[Apr.2011 – 8M]</u> | 4. Write a program in VB to accept product details and store it into the database and display amount. The database fields are: Itemno, Itemname, Rate, Quantity. |
| <u>[Oct.2011 – 8M]</u> | 5. Draw an interface and code for the following. Also give property setting for appropriate controls. Write a program in VB to store a data into the database with the fields' rollno, name, marks 1, marks 2, marks 3. Calculate average. |

Suggestive Readings:

1. Cornell, Gary. 1998. Visual Basic 6 from the Ground Up. New Delhi: Tata McGraw-Hill.
2. Warner, Scott L. 1998. Teach Yourself Visual Basic 6. New Delhi: Tata McGraw Hill.
3. Jerke, Noel. 1999. Visual Basic 6 - The Complete Reference. New York: McGraw
4. Smith, Eric A., Valor Whisler and Hank Marquis. 1998. Visual Basic 6 Programming Bible. New Jersey: John Wiley & Sons, Inc.
5. Azam, M. 2001. Programming with Visual Basic. New Delhi: Vikas Publishing House Pvt. Ltd.
6. Manchanda, Mahesh. 2009. Visual Programming. New Delhi: Vikas Publishing House Pvt. Ltd.
7. Balena, Francesco. 1999. Programming Microsoft Visual Basic 6.0. Bangalore: WP Publishers and Distributors (P) Ltd.
8. Petroustos, Evangelos. 1998. Mastering Visual Basic 6, 1st Edition. New Delhi: BPB Publications.
9. Deitel, Harvey M., Paul J. Deitel and T. Tem R. Nieto. 1999. Visual Basic 6: How to Program. New Jersey: Prentice-Hall.
10. Norton, Peter. 1998. Peter Norton's Complete Guide to Visual Basic 6. New Delhi: Techmedia.
11. Reselman, Bob and Richard A. Peasley. 1998. Using Visual Basic 6. New Jersey: Pearson Education (Que Publishing).
12. Donald, Bob and Oancea Gabriel. 1999. Visual Basic 6 from Scratch. New Delhi: Prentice-Hall of India.