



ALAGAPPA UNIVERSITY

[Accredited with 'A+' Grade by NAAC (CGPA:3.64) in the Third Cycle
and Graded as Category-I University by MHRD-UGC]

(A State University Established by the Government of Tamil Nadu)

KARAIKUDI – 630 003



Directorate of Distance Education

B.Sc. [Computer Science]

VI - Semester

130 62

VISUAL BASIC PROGRAMMING

Authors:

"The copyright shall be vested with Alagappa University"

All rights reserved. No part of this publication which is material protected by this copyright notice may be reproduced or transmitted or utilized or stored in any form or by any means now known or hereinafter invented, electronic, digital or mechanical, including photocopying, scanning, recording or by any information storage or retrieval system, without prior written permission from the Alagappa University, Karaikudi, Tamil Nadu.

Information contained in this book has been published by VIKAS® Publishing House Pvt. Ltd. and has been obtained by its Authors from sources believed to be reliable and are correct to the best of their knowledge. However, the Alagappa University, Publisher and its Authors shall in no event be liable for any errors, omissions or damages arising out of use of this information and specifically disclaim any implied warranties or merchantability or fitness for any particular use.



VIKAS®

Vikas® is the registered trademark of Vikas® Publishing House Pvt. Ltd.

VIKAS® PUBLISHING HOUSE PVT. LTD.

E-28, Sector-8, Noida - 201301 (UP)

Phone: 0120-4078900 • Fax: 0120-4078999

Regd. Office: A-27, 2nd Floor, Mohan Co-operative Industrial Estate, New Delhi 1100 44

• Website: www.vikaspublishing.com • Email: helpline@vikaspublishing.com

Work Order No. AU/DDE/DE-12-27/Preparation and Printing of Course Materials/2020 Dated 12.08.2020 Copies -

SYLLABUS

Visual Basic Programming

BLOCK I: VISUAL BASIC CONCEPTS

UNIT 1 - Introduction to GUI - Visual Basic: Starting and Exiting Visual Basic Project Explorer, Working with Forms, Properties Window.

UNIT 2 - Using the Toolbox, Toolbars, Working with Projects, Programming Structure of Visual Basic Applications, Event and Event Driven Procedures.

UNIT 3 - Program Design: Form and Controls, Writing the Code, Saving, Running and Testing, Making EXE File, Printouts.

Unit 1: Introduction to Graphical user Interface
(Pages 1-26);

Unit 2: Using the Toolbox
(Pages 27-46);

Unit 3: Program Design
(Pages 47-60)

BLOCK II: VISUAL BASIC CODE, EVENTS AND CONTROLS

UNIT 4 - Adding Code and Using Events: Using Literals Data Types, Declaring and Using Variables, Using the Operator Subroutines and Functions.

UNIT 5 - Looping and Decision Control Structures: If...Then...Else, Structure, Select Structure, For...Next, Do...Loop and While...Wend.

UNIT 6 - Using Intrinsic Visual Basic Controls with Methods and Properties: Label, Text Box, Command Button, Frame, Check Box, Option Button, List Box, Combo Box, Drive List Box, Directory List Box and File List Box, Formatting Controls, Control Arrays, Tab Order.

Unit 4: Adding Code and Using Events
(Pages 61-90);

Unit 5: Looping and Decision Control Structures
(Pages 91-100);

Unit 6: Using Intrinsic Visual Basic Controls with Methods and Properties
(Pages 101-128)

BLOCK III: VISUAL BASIC PROCEDURES, FUNCTIONS AND ARRAYS

UNIT 7 - Creating Procedures, Functions, String Functions, Date and Time Function, Numeric Functions, Recursive Functions.

UNIT 8 - Multiple Forms, Startup Forms, SubMain Procedure.

UNIT 9 - Arrays, Control Arrays, Indexing and Event Handling, Graphics.

Unit 7: Creating Procedures and Functions
(Pages 129-146);

Unit 8: Multiple Forms
(Pages 147-162);

Unit 9: Arrays
(Pages 163-186)

BLOCK IV: MENUS AND MDI FORMS

UNIT 10 - Menus: Creating Menus, Adding Code to Menus.

UNIT 11 - Using MDI Forms: MDI Form, Basic Building MDI Form, Creating MDI Child Forms.

Unit 10: Menus
(Pages 187-218);

Unit 11: Using MDI Forms
(Pages 219-236)

BLOCK V: DATA ACCESS OBJECT (DAO) AND PROPERTIES

UNIT 12 - Database Object (DAO) and Properties: Accessing Recordset Objects; MoveFirst, MoveLast, MovePrevious and MoveNext Methods; Begin, Commit and Rollback Transaction, Accessing Microsoft Access Files.

UNIT 13 - Active Data Objects (ADO): ADO and OLE DB and ADO Primer, What are OLE DB and ADO? ADO Object Model, Converting DAO Code to Use ADO.

UNIT 14 - Connecting to the Database, Retrieving a Recordset, Creating a Query Dynamically, Using a Parameterized Query, Using Action Queries, Adding Records, Editing Records, Closing the Database Connection.

Unit 12: Data Access Object (DAO) and Properties
(Pages 237-254);

Unit 13: Active Data Objects (ADO) and ADO Primer
(Pages 255-276);

Unit 14: Connecting to the Database
(Pages 277-308)

CONTENTS

INTRODUCTION

BLOCK I: VISUAL BASIC CONCEPTS

UNIT 1 INTRODUCTION TO GRAPHICAL USER INTERFACE 1-26

- 1.0 Introduction
- 1.1 Objectives
- 1.2 Visual Basic
 - 1.2.1 Features of Visual Basic
 - 1.2.2 Visual Basic 6.0 Versus Earlier Versions of Visual Basic
 - 1.2.3 Event-Driven Programming
 - 1.2.4 Properties, Methods and Events
 - 1.2.5 Getting Started with Visual Basic 6.
- 1.3 Understanding Visual Basic Projects
 - 1.3.1 Visual Basic Environment (Integrated Development Environment)
 - 1.3.2 Visual Basic Program Development Process
 - 1.3.3 Opening/Saving Running a Visual Basic Project
- 1.4 Answers to Check Your Progress Questions
- 1.5 Summary
- 1.6 Key Words
- 1.7 Self-Assessment Questions and Exercises
- 1.8 Further Readings

UNIT 2 USING THE TOOLBOX 27-46

- 2.0 Introduction
- 2.1 Objectives
- 2.2 Working with Toolbar
- 2.3 Use of the Toolbox
- 2.4 Project Programming Structure in Visual Basic Application
- 2.5 Event and Event Driven Procedures
- 2.6 Answers to Check Your Progress Questions
- 2.7 Summary
- 2.8 Key Words
- 2.9 Self-Assessment Questions and Exercises
- 2.10 Further Readings

UNIT 3 PROGRAM DESIGN 47-60

- 3.0 Introduction
- 3.1 Objectives
- 3.2 Forms and Controls
 - 3.2.1 Creating and Saving a New Program
 - 3.2.2 Restoring/Opening an Existing Program
 - 3.2.3 Running the Program
 - 3.2.4 Stopping the Program
 - 3.2.6 Printing Visual Basic Project
 - 3.2.7 Exiting Visual Basic
- 3.3 Making Exe Files
- 3.4 Answers to Check Your Progress Questions
- 3.5 Summary
- 3.6 Key Words

- 3.7 Self-Assessment Questions and Exercises
- 3.8 Further Readings

BLOCK II: VISUAL BASIC CODE, EVENTS AND CONTROLS

UNIT 4 ADDING CODE AND USING EVENTS

61-90

- 4.0 Introduction
- 4.1 Objectives
- 4.2 Data Types
- 4.3 Declaring and Using Variables
- 4.4 Introducing Operators
 - 4.4.1 Arithmetic Operators
 - 4.4.2 Relational Operators
 - 4.4.3 Concatenation Operators
 - 4.4.4 Logical Operators
- 4.5 Answers to Check Your Progress Questions
- 4.6 Summary
- 4.7 Key Words
- 4.8 Self-Assessment Questions and Exercises
- 4.9 Further Readings

UNIT 5 LOOPING AND DECISION CONTROL STRUCTURES

91-100

- 5.0 Introduction
- 5.1 Objectives
- 5.2 Looping and Decision Control Structures
 - 5.2.1 Decision Structure
 - 5.2.2 Loop Structure
- 5.3 Answers to Check Your Progress Questions
- 5.4 Summary
- 5.5 Key Words
- 5.6 Self-Assessment Questions and Exercises
- 5.7 Further Readings

UNIT 6 USING INTRINSIC VISUAL BASIC CONTROLS WITH METHODS AND PROPERTIES

101-128

- 6.0 Introduction
- 6.1 Objectives
- 6.2 Basic Controls
- 6.3 Control Array
- 6.4 Answers to Check Your Progress Questions
- 6.5 Summary
- 6.6 Key Words
- 6.7 Self-Assessment Questions and Exercises
- 6.8 Further Readings

BLOCK III: VISUAL BASIC PROCEDURES, FUNCTIONS AND ARRAYS

UNIT 7 CREATING PROCEDURES AND FUNCTIONS

129-146

- 7.0 Introduction
- 7.1 Objectives
- 7.2 Creating Procedures and Functions

- 7.3 Answers to Check Your Progress Questions
- 7.4 Summary
- 7.5 Key Words
- 7.6 Self-Assessment Questions and Exercises
- 7.7 Further Readings

UNIT 8 MULTIPLE FORMS

147-162

- 8.0 Introduction
- 8.1 Objectives
- 8.2 Startup Forms
- 8.3 Submain Procedure
- 8.4 Answers to Check Your Progress Questions
- 8.5 Summary
- 8.6 Key Words
- 8.7 Self-Assessment Questions and Exercises
- 8.8 Further Readings

UNIT 9 ARRAYS

163-186

- 9.0 Introduction
- 9.1 Objectives
- 9.2 Arraya and Control Arrays
 - 9.2.1 Fixed Size Arrays
 - 9.2.2 Dynamic arrays
 - 9.2.3 Array Characterstics
 - 9.2.4 Processing Array Elements
 - 9.2.5 Control arrays
- 9.3 Indexing and Event Handling
- 9.4 Graphics
- 9.5 Answers to Check Your Progress Questions
- 9.6 Summary
- 9.7 Key Words
- 9.8 Self-Assessment Questions and Exercises
- 9.9 Further Readings

BLOCK IV: MENUS AND MDI FORMS

UNIT 10 MENUS

187-218

- 10.0 Introduction
- 10.1 Objectives
- 10.2 Menus
 - 10.2.1 Using the Visual Basic Application Wizard
 - 10.2.2 Using the Visual Basic Menu Editor
- 10.3 Answers to Check Your Progress Questions
- 10.4 Summary
- 10.5 Key Words
- 10.6 Self-Assessment Questions and Exercises
- 10.7 Further Readings

UNIT 11 USING MDI FORMS

219-236

- 11.0 Introduction
- 11.1 Objectives

- 11.2 Multiple Document Interface (MDI) Forms
 - 11.2.1 Accessing Child Forms
 - 11.2.2 Adding, Loading and Unloading Forms
- 11.3 Answers to Check Your Progress Questions
- 11.4 Summery
- 11.5 Key Words
- 11.6 Self-Assessment Questions and Exercises
- 11.7 Further Readings

BLOCK IV: DATA ACCESS OBJECT (DAO) AND PROPERTIES

UNIT 12 DATA ACCESS OBJECT (DAO) AND PROPERTIES 237-254

- 12.0 Introduction
- 12.1 Objectives
- 12.2 Data Access Object (DAO)
- 12.3 Accessing Data Through Transaction Method
- 12.4 Answers to Check Your Progress Questions
- 12.5 Summary
- 12.6 Key Words
- 12.7 Self-Assessment Questions and Exercises
- 12.8 Further Readings

UNIT 13 ACTIVE DATA OBJECTS (ADO) AND ADO PRIMER 255-276

- 13.0 Introduction
- 13.1 Objectives
- 13.2 Active Data Objects (ADO): An Introduction
- 13.3 Answers to Check Your Progress Questions
- 13.4 Summary
- 13.5 Key Words
- 13.6 Self-Assessment Questions and Exercises
- 13.7 Further Readings

UNIT 14 CONNECTING TO THE DATABASE 277-308

- 14.0 Introduction
- 14.1 Objectives
- 14.2 Database Connectivity
- 14.3 Retrieving a Recordset
 - 14.3.1 What is Data Provider?
 - 14.3.2 What is OLE DB?
 - 14.3.3 Accessing Database using ADO Control
- 14.4 Working with Queries
 - 14.4.1 Parameterized Queries
 - 14.4.2 Action Query
- 14.5 Adding Records and Editing Records
 - 14.5.1 ADO: Adding a Record to a Record Set
 - 14.5.2 ADO: Editing a Record in a Record Set
- 14.6 Closing the Database Connection
- 14.7 Answers to Check Your Progress Questions
- 14.8 Summary
- 14.9 Key Words
- 14.10 Self-Assessment Questions and Exercises
- 14.11 Further Readings

INTRODUCTION

NOTES

Visual Basic (VB) is a third generation event driven programming language from Microsoft known for its Component Object Model (COM) programming model first released in 1991 and declared legacy during 2008. Microsoft's extended support ended in March 2008 and the designated successor was Visual Basic.NET now acknowledged simply as Visual Basic.

Microsoft intended Visual Basic to be relatively easy to learn and use. Visual Basic was derived from BASIC (Beginners' All-purpose Symbolic Instruction Code) and enables the Rapid Application Development (RAD) of Graphical User Interface (GUI) applications, access to databases using Data Access Objects (DAO), Remote Data Objects (RDO), or ActiveX Data Objects (ADO), and creation of ActiveX controls and objects. A programmer can create an application using the components provided by the Visual Basic program itself. Programs written in Visual Basic can also make use of the Windows API, which requires external functions declarations.

This book, *Visual Basic Programming*, is divided into five blocks, which are further subdivided into fourteen units. This book provides a basic understanding of the subject and helps to grasp its fundamentals. In a nutshell, it explains various aspects, such as Visual Basic concepts, introduction to GUI, working with Forms, Properties Window, Toolbox, Toolbars, working with Projects, programming structure of Visual Basic Applications, event and event driven procedures, program design, writing the code, using Literals Data Types, declaring and using Variables, using the Operator Subroutines and Functions, looping and decision control structures, intrinsic Visual Basic controls with methods and properties, creating procedures, functions, string functions, date and time function, numeric functions, recursive functions, arrays, indexing and event handling, graphics, Menus and MDI Forms, Active Data Objects (ADO) and properties.

The book follows the Self-Instructional Mode (SIM) wherein each unit begins with an 'Introduction' to the topic. The 'Objectives' are then outlined before going on to the presentation of the detailed content in a simple and structured format. 'Check Your Progress' questions are provided at regular intervals to test the student's understanding of the subject. 'Answers to Check Your Progress Questions', a 'Summary', a list of 'Key Words', and a set of 'Self-Assessment Questions and Exercises' are provided at the end of each unit for effective recapitulation. This book provides a good learning platform to the people who need to be skilled in the area of operating system functions. Logically arranged topics, relevant examples and illustrations have been included for better understanding of the topics and for effective recapitulation.

BLOCK I

VISUAL BASIC CONCEPTS

*Introduction to
Graphical user
Interface*

UNIT 1 INTRODUCTION TO GRAPHICAL USER INTERFACE

NOTES

- 1.0 Introduction
- 1.1 Objectives
- 1.2 Visual Basic
 - 1.2.1 Features of Visual Basic
 - 1.2.2 Visual Basic 6.0 Versus Earlier Versions of Visual Basic
 - 1.2.3 Event-Driven Programming
 - 1.2.4 Properties, Methods and Events
 - 1.2.5 Getting Started with Visual Basic 6.
- 1.3 Understanding Visual Basic Projects
 - 1.3.1 Visual Basic Environment (Integrated Development Environment)
 - 1.3.2 Visual Basic Program Development Process
 - 1.3.3 Opening/Saving Running a Visual Basic Project
- 1.4 Answers to Check Your Progress Questions
- 1.5 Summary
- 1.6 Key Words
- 1.7 Self-Assessment Questions and Exercises
- 1.8 Further Readings

1.0 INTRODUCTION

Visual Basic (VB) is a programming language for developing sophisticated professional applications for Microsoft Windows. It utilizes a Graphical User Interface (GUI) to create powerful applications. The use of illustrations for text by GUI helps users in interacting with an application. This feature helps in easy and quick comprehension. The VB programming language is event driven and programming takes place in a graphical environment, whereas in its previous version, BASIC, programming occurs in a text only environment which is sequentially executed for controlling user interface. VB is an attractive tool to work with because of features, such as faster application development, easier comprehension, user-friendliness, and Internet and ActiveX technology. VB programming language combines event driven and object-oriented programming that supports GUI effects.

Visual Basic is an event driven programming language. The 'Visual' part refers to the method used to create the Graphical User Interface (GUI).

NOTES

The “Basic” part refers to the BASIC language that is used by most of the programmers in the history of computing. Visual Basic has evolved from the BASIC language and now contains several hundred statements, functions and keywords, many of which are directly related to the Windows GUI. Visual Basic allows professionals to implement anything that can be implemented using any other Windows programming language. The Visual Basic programming language is not unique to Visual Basic. The Visual Basic programming system, Applications Edition included in Microsoft Excel, Microsoft Access and many other Windows applications use the same language. The Visual Basic programming system, Scripting Edition (VBScript) for Internet programming is a subset of the Visual Basic language. The effort you make in learning Visual Basic will carry over to these other areas.

In this unit, you will study about the starting and exiting Visual Basic project explorer and working with Forms properties Windows.

1.1 OBJECTIVES

After going through this unit, you will be able to:

- Know about Visual Basic
- Understand the Visual Basic Projects
- Elaborate on the Features of Visual Basic
- Explain about the Visual Basic environment
- Define the basic terms used in Visual Basic
- Able to open/save and run a Visual Basic Project

1.2 VISUAL BASIC

Visual Basic is a third-generation event driven programming language from Microsoft known for its Component Object Model (COM) .Programming model first released in 1991 and declared legacy during 2008. Microsoft intended Visual Basic to be relatively easy to learn and use. Visual Basic was derived from BASIC and enables the Rapid Application Development (RAD) of Graphical User Interface (GUI) applications, access to databases using Data Access Objects (DAO), Remote Data Objects (RDO), or ActiveX Data Objects (ADO), and creation of ActiveX controls and objects.

1.2.1 Features of Visual Basic

The Visual Basic (VB) programming environment provides all the features that are required to develop a Graphical User Interface (GUI). Some important ones are listed below:

1. It is the successor of BASIC 'Beginner' ALL-Purpose Symbolic Instruction Code).
2. It provides a common programming platform across all MS-Office applications.
3. It offers many tools that provide a quick and easy way to develop an application.
4. It also provides many wizards that can automate tasks or even automate coding.
5. It supports ActiveX Control, with which you can create your own ActiveX control and use it in your application.
6. It has an n -tier architecture.
7. It offers quick error detection/correction.
8. It comes with a full set of objects for 'Drawing' the application.
9. It responds to the action of the mouse and the keyboard.
10. It has access to the clipboard and printer.
11. It comes with a full array of string handling, mathematical and graphics functions.
12. It can handle dynamic and fixed variables and control arrays.
13. It supports sequential and random access files.
14. It has powerful tools for database access.
15. It has a package and deployment wizard to make distribution of applications easy.

1.2.2 Visual Basic 6.0 versus Earlier Versions of Visual Basic

The original Visual Basic for DOS (Disk Operating System) and Visual Basic for Windows were introduced in 1991.

Visual Basic 3.0, an improved version, was released in 1993.

Visual Basic 4.0 added 32-bit application support and was released in late 1995.

Visual Basic 5.0 was released in late 1996. It supported the creation of ActiveX controls, provided a new environment and deleted 16-bit application support.

Visual Basic 6.0 was released in mid-1998. Its new features are:

- New ActiveX data control object
- Faster compiler
- Integration of database with large variety of applications
- New package and deployment wizard
- New data report designer
- Additional Internet capabilities

NOTES

NOTES

Visual Basic 3.0 was a powerful language but it was reasonably small. The addition of classes to the language in Visual Basic 4.0 made it much more complex. Though more support for database programming and other topics like custom controls in Versions 4.0, 5.0 and 6.0 made it even more complex, Visual Basic remained an easy to understand language.

Visual Basic .NET speeded up the expansion of Visual Basic immensely. The .NET framework added various powerful new tools. Various allied technologies have been added to the language. Today, it is not possible for anyone to be an expert on each topic dealing with Visual Basic (VB).

Its system requirements are dependent on the version of the software. Visual Basic 6.0 for Windows requires Microsoft Windows 95/Windows NT 3.51, a 486 processor and a minimum RAM of 16 MB. The Enterprise Edition, the most powerful version of Visual Basic 6.0, requires more than 250 MB of hard disk space for a complete installation.

1.2.3 Event Driven Programming

In a VB project, the occurring processes have to be associated with events. An event is something that occurs when button being clicked by the user or a form being opened. The operation is driven by event because every execution is a result of some event. The programmer role is to anticipate the events and to write the code for execution during the occurrence of the event. A VB application is interactive because of the constant interaction of the user with the program. VB programs are built around events, which are different incidents that can occur in a program. This will become clearer when VB is compared to procedural programming. In procedural languages, an application is written and executed by a logical checking of the program through the program statements, one after another. The control can be transferred to some other point in a program on a temporary basis. In applications that are event-driven, the execution of the program statements occurs only when a certain event calls a particular part of the code assigned to it.

We can look at a TextBox control and a few of its related events to understand the concept of event-driven programming. The TextBox control supports several events, such as Change, Click, MouseMove and others, that will be listed in the drop-down list of Properties in the code window for the TextBox control. Let us look at a few of them:

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

1.2.4 Properties, Methods and Events

Put simply, objects are described by properties. Methods make an object do something. Events are what occur when something is done by an object. Every object, like control or a form, has certain properties describing it. This set is not equal for all objects.

1.2.5 Getting Started with Visual Basic 6.0

Visual Basic 6.0 (VB 6.0) was released in mid-1998. It is started either by clicking the VB icon or Programs → Microsoft -VB 6.0 → VB 6.0. After clicking the VB icon, a new screen is opened to users- friendly with additional features, such as MenuBar, ToolBar and the New Project dialog box. The interface elements facilitate the user to develop enhanced VB 6.0 in associated windows with the project providing single container, popularly known as the parent. The main container form contains code and form-based windows. Getting started with Visual Basic 6.0 (VB 6.0) provides additional features, for example, Project Explorer, Properties Windows and Object Browser. Basically, VB 6.0 for Windows requires Microsoft Windows 95/Windows NT 3.51, a 486 processor and a minimum RAM of 16 MB. The Enterprise Edition, the most powerful version of VB 6.0, requires more than 250 MB of hard disk space for a complete installation.

The Project Explorer window provides forms, classes and modules collectively in a group as elements to the programmer. In fact, a sample project typically contains one form, where as a program might contain single form. It contains the Properties Window that exposes the various characteristics, for example, window color and size, but the characteristics and elements are considered as objects. The form contains properties and controls. The Object Browser facilitates user to browse the properties, events and methods. This browser can be selected either by pressing the function key [F2] from the keyboard or from View menu.

Its new features are as follows:

- New ActiveX data control object
- Fast compiler
- Integration of database with large variety of applications
- New package and deployment wizard
- New data report designer
- Additional Internet capabilities

You can start VB 6.0 by the following three steps:

Click at Program → Microsoft VB Studio → Microsoft VB6.0

NOTES

NOTES

Check Your Progress

1. What is Visual Basic?
2. Which type of applications can be developed using Visual Basic?
3. Explain the features of Visual Basic.
4. List out the different versions of Visual Basic.
5. Which is the most powerful version of VB?
6. Explain about the event in Visual Basic.
7. Elaborate on the TextBox events.
8. Write down the steps for starting Visual Basic (VB) 6.0.

1.3 UNDERSTANDING VISUAL BASIC PROJECTCS

For opening a Visual Basic (VB) environment and working with it, select and click on Microsoft Visual Basic 6.0 in the start menu. After VB is loaded, the New Project dialog, shown in Figure 1.1, will be displayed with the types available in VB. You can see that **Standard Exe** is highlighted by default. **Standard Exe** helps the user in creating a standard executable. Standard executable is a category that has most of the common features of VB.

A **project** in VB is a collection of several types of files that make up your program. An **application** is the final program that is used by people.



Fig. 1.1 The Visual Basic Startup Dialog Box

Visual Basic Project

Various types of projects that you can create are listed in Table 1.1 as follows:

Table 1.1 Different Types of Projects Supported by Visual Basic 6.0

Project Name	Description
1 Standard EXE	For creating a stand alone application.
2 ActiveX EXE	For creating an ActiveX executable component that can be executed from other applications.
3 ActiveX DLL	For creating ActiveX dynamic link library.
4 ActiveX Control	For creating your own ActiveX control. An ActiveX control is a basic element of user interface. For Example, a TextBox or a command button, etc.
5 VB Application Wizard	For setting up the skeleton of a new application.
6 VB Wizard Manager	For collecting information from the user. After the user fills out all the required information, the wizard proceeds to carry out a specified task.
7 Data Project	For creating a data project, which is a combination of standard EXE project and various data access controls.
8 Internet Information Server (IIS) Application	For creating an Internal Information Server (IIS) application that can run on a Web server.
9 Add In	For creating your own Add in for the Visual Basic Integrated Development Environment (VBIDE).
10 ActiveX Document DLL	For creating ActiveX documents in DLL form.
11 ActiveX Document EXE	For creating ActiveX documents in EXE form.
12 DHTML Application	For building dynamic HTML pages.

NOTES

1.3.1 Visual Basic Environment (Integrated Development Environment)

The working environment in Visual Basic (VB) incorporates several different functions such as editing, designing, compiling and debugging within a common environment. For this reason, the working environment of VB is also referred to as Integrated Development Environment (IDE) as shown in Figure 1.2.

To start up VB, you need to follow these steps:

1. Click at Start button.
2. Click at Program → Microsoft Visual Basic Studio → Microsoft VB 6.0

This will open a New Project dialog box, as shown in Figure 1.1, which is the startup dialog box. From this dialog box, you can select the desired project

NOTES

type. When you select, for example, a Standard EXE from the New Project dialog box, VB starts a new project and shows the screen as shown in Figure 1.2. The following screenshot shows the involvement of Integrated Development Environment (IDE) in Visual Basic.

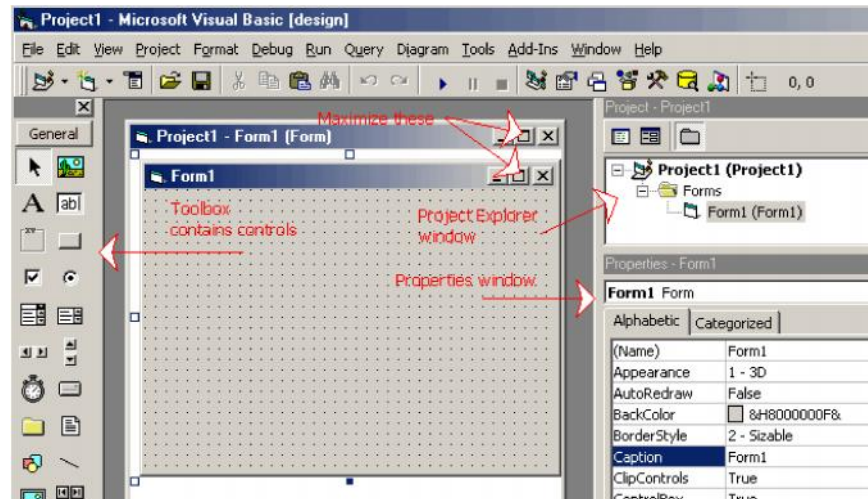


Fig. 1.2 A Standard EXE Project

Elements of VB IDE

The Visual Basic Integrated Development Environment (VB IDE) consists of the following elements:

Title Bar: It is the topmost bar displaying the title of the project. The window titled **Project 1** is the name of project containing the project files.

Form: It is the main feature of the VB application; it is the 'Window' or 'Screen' that users interact with. It can be considered as a 'Canvas' on which the user places the objects that form an application. When a new project is started, VB automatically supplies one form to work with; more forms can be added as the need arises. Forms are used to display things, such as TextBox, Label, CommandButton, Graphics, and so on.

Toolbox Window: It has a set of controls used for customizing forms. These controls help the user in creating an interface between the user and the application.

Properties Window: It helps in changing the property settings or characteristics of the form itself and also the elements of visual interface on the form. There are two columns in the Properties window: the first is the property name that cannot be changed and the second is the setting of the property that can be changed.

Project Explorer or Project Window: It shows the list of forms and modules in a project. A VB Project Explorer consists of a number of forms, modules and controls that make up an application.

NOTES

Form Layout Window: It shows how big a form is in relation to the screen. It also displays the position of the window where it will be displayed when the project is run.

Code Editor Window: It is the place for writing VB code for your application. By code we mean language statements, declarations and constants. For entering application code, the code editor window serves as an editor. A separate code editor window is created for each form or control that you create in your application. With the use of the code editor window, any code in the application can be quickly viewed and edited.

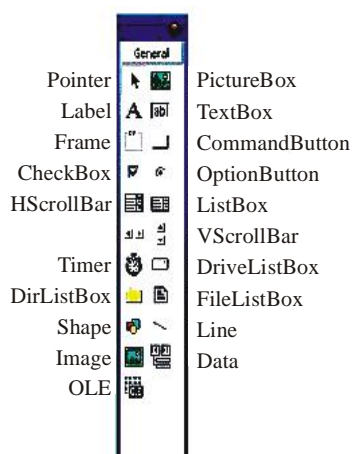


Fig. 1.3 (a) Toolbox Window



Fig. 1.3 (b) Properties Window

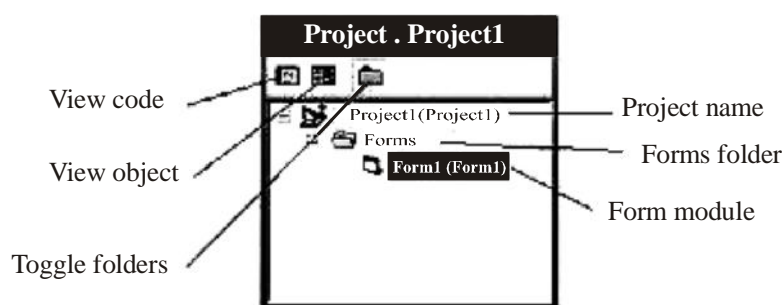


Fig. 1.3 (c) Project Window

The following is a list of common controls with their description:

Table 1.2 Different Types of Intrinsic Controls and their Description

Control	Description
Pointer	Used for interacting with the controls on the form
PictureBox	Used for displaying images
TextBox	Used for accepting user input that can display only editable text
Frame	Used for grouping other controls

NOTES

CommandButton	Used for initiating an action by pressing on the button
CheckBox	Used for making a choice for user (checked or unchecked)
OptionButton	Used in groups where one at a time can be true
ListBox	Used for providing a list of items
ComboBox	Used for providing a short list of items
HScrollBar	Used to scroll horizontally
VScrollBar	Used to scroll vertically
Timer	Used for performing tasks at particular intervals
DriveListBox	Used for accessing the system drives
DirListBox	Used for accessing the directories on the system
FileListBox	Used for accessing the files in the directory
Shape	Used for drawing circles, squares, rectangles, ellipses
Line	Used for drawing lines
Image	Used for displaying images but has less capability than the PictureBox
Data	Used for connecting a database
OLE	Used for interacting with other application of Windows
Label	Used for display texts that cannot be edited

VB IDE Modes

A VB application works in either of the following three modes:

- Design Mode
- Run Mode
- Break/Suspended Mode

While an application is being created or designed, it is in the *design mode*. When the application is executing, it is said to be in the *run mode*. And while an application is in a state of *suspension*, it is said to be in the *break mode*.

Definition of Basic Terms

Application: An application is an assortment of objects that work together for accomplishing something useful. The application in VB is called **Project**. A project could be calculation of mortgages, management of a video store, the payroll for 1000 employees or a dating service.

Object: An object is a part of software with properties and functions that can be changed. A window is an **object** with **properties** like color, size, position on the screen, etc. The function of a window, also known as **methods**, can be manipulated, move it around, change the size, open it and close it. There is no need for writing the code for resizing a window. Clicking and dragging will do. The code was written by somebody and put together in a small package called **window object**. Whenever there is a need of a window in a project, the window object can be copied and pasted wherever it is needed by changing its properties for color or size very easily. Its built-in methods can be used for opening and closing it or for resizing it whenever

required. When an application is created using objects and combining them for producing results, it means the user is working in an **object-oriented** environment.

Help Topics Dialog Box: The new Help Topics Dialog Box is the 'Way in' to help information in an application. To invoke the Help Dialog Box, choose the **content** command from the **Help** menu. It provides three tabs to help the user to find the required information. The information displayed on these tabs is totally dependent on the way the Help topic files and the content file are created. Each tab has its own unique features.

Contents Tab: This helps in displaying the help topics by title in the categories set up by you. Jumps to topics as well as the capability of running macros can be created in other Help files. Additionally, the Contents tab topics can be automatically updated whenever the latest release of the application is installed on the PC of the user. Jumps to topics or macros are displayed as page icons, while headings are displayed as book icons on the display.

Index Tab: The old WinHelp 3.1 Search Dialog Box has been changed and replaced by the Index tab. The Index tab provides you the capability of looking for topics based on the keywords that you add to your Help file. These keywords are then sorted automatically from A to Z and any second-level entry is indented.

Find Tab: This is used for searching the help file for topics containing a particular word or phrase that has been specified in the top input box on the tab. The distinctive feature of the Find tab is that the word list the Find tab works on is created when the user clicks the Find tab. This means that there is no need of building and sending a large word list files on the program disks for this to work.

1.3.2 Visual Basic Program Development Process

Generally, the following steps are required for building a VB application:

- Step 1: Designing the Interface
- Step 2: Setting Properties of the Controls (Objects)
- Step 3: Writing the Procedures of the Events

Step1: Designing the Interface

In our first example, there is a need of 6 **Labels** and 2 **Command buttons**. These objects that are put on a Form are called **controls**. For getting a control, go to the **Toolbox**, then click on the control you need, return to the Form and click and drag the control to the size and position you want. Position the controls as in the following diagram:

NOTES

NOTES

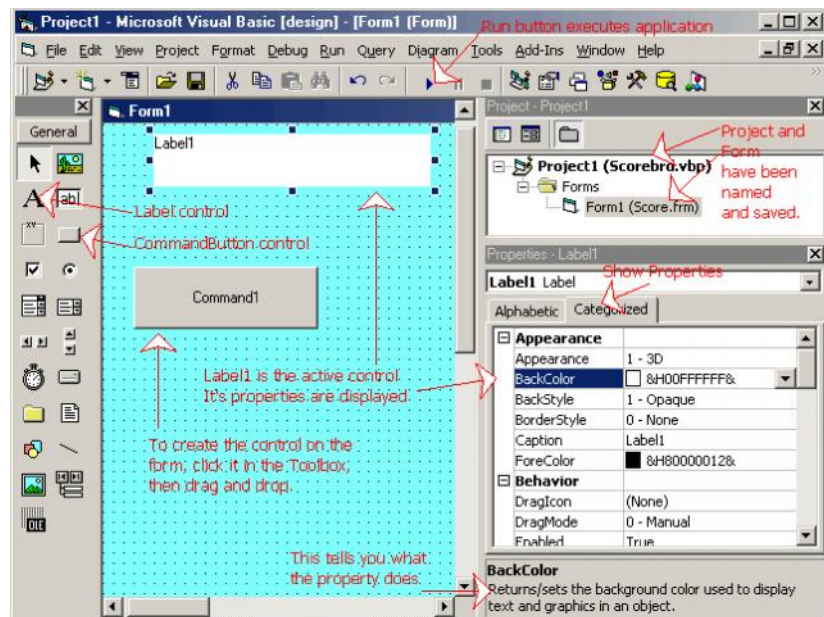


Fig. 1.4 Visual Basic IDE

We can make the bunch of controls on the form more attractive by changing the **Properties** of the controls in the **Properties Window**. Each control has a whole series of properties. But right now, we only need the following ones:

Alignment = How text aligns in the control

BackColor = The color of the background

Caption = The text that will appear in the control

Font = The font type and size

ForeColor = The color of the text (foreground)

Just as in all Windows applications, you can select multiple controls with (Ctrl)+(Click) for changing a property for all of them at once. For instance, if there are all white backgrounds, select all controls, change ForeColor to white and all of them will be modified. Change the form to look like the one below. Note that there is no need to change the captions for Label4, Label5 and Label6 and the color of the buttons cannot be changed. The color of the buttons is what was earlier called 'IBM Grey'. Remember to save your project as often as you can.

Step 2: Setting Properties of the Controls (Objects)

NOTES

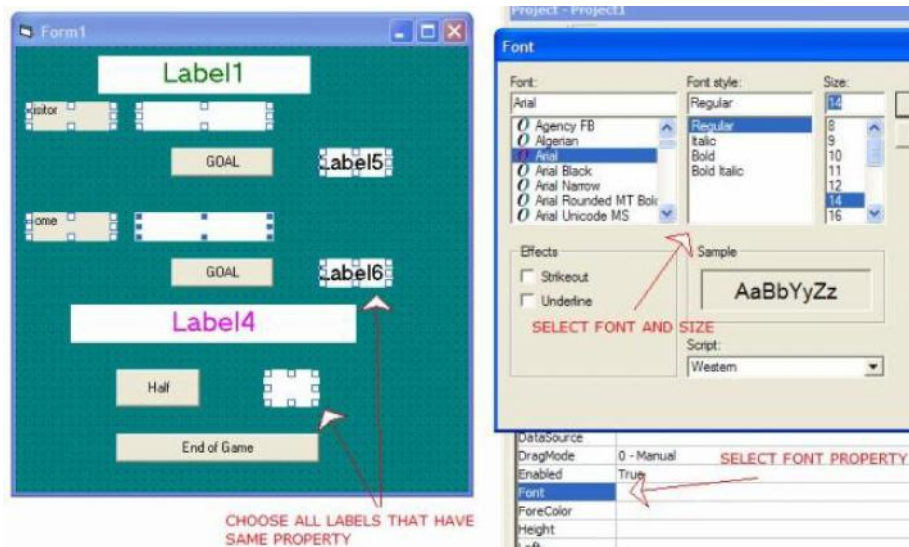


Fig. 1.5 Design Mode

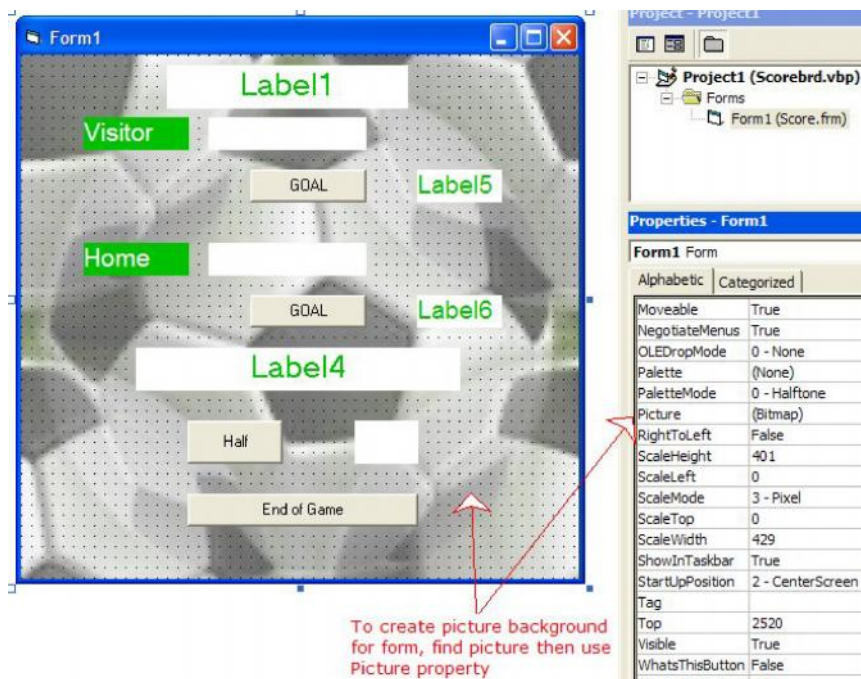


Fig. 1.6 Final Display

At this point, if you **Run** the application, your Form will appear just the way it was created. However, **absolutely nothing happens** if you click on any of the controls. Some **events** occur; the form opens, a button is clicked, etc. But nothing tells the form what should be done when it sees an event. This is the reason that we have to write code, which is also known as **script**.

NOTES

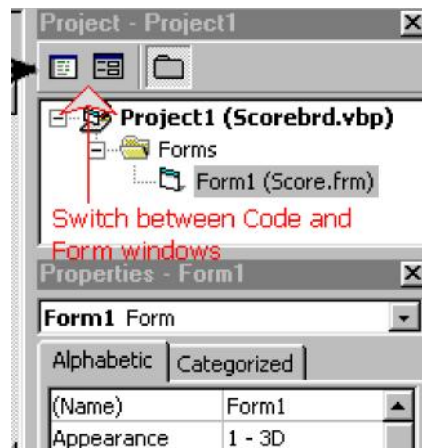


Fig. 1.7 (a) Project Explorer Window

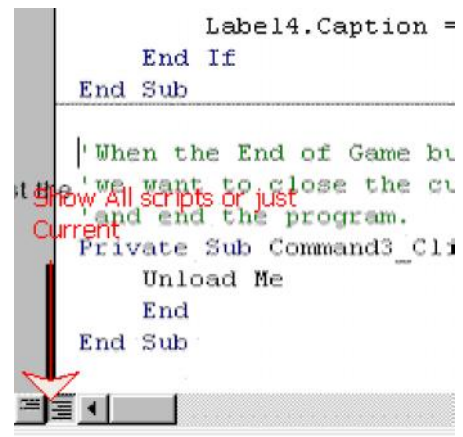


Fig. 1.7(b) Code Editor Window

For switching between the Code window and the Form window, the buttons just over the Project Explorer window [(Figure 1.7 (a))].

Once you are in the Code window, you can see all the codes for the project or the code for one event at a time. Use the buttons in the lower left-hand corner (Figure 1.7 (b)).

For selecting the object and the event you want to code, use the two List boxes given at the top of the Code window: the left button is for the object and the right button is for the event. Start with **General ... Declarations** and then **Form ... Load**, etc.

Step 3: Write the Procedures of the Events

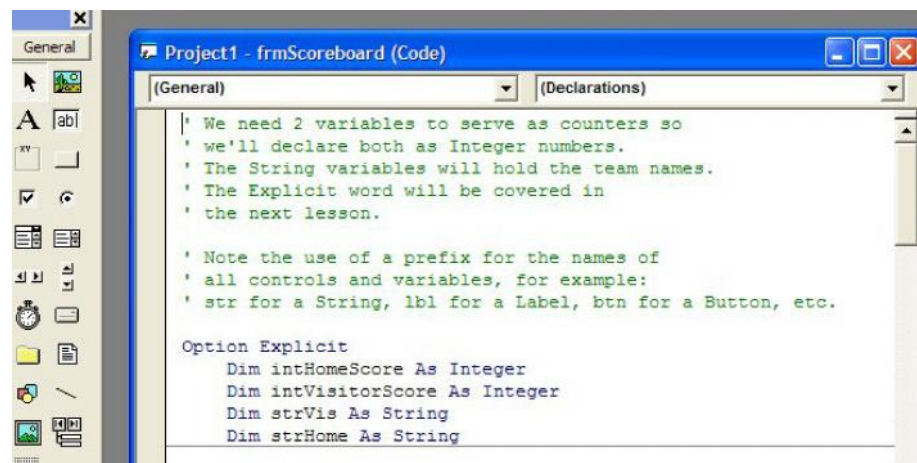


Fig. 1.8 Showing the Code

NOTES

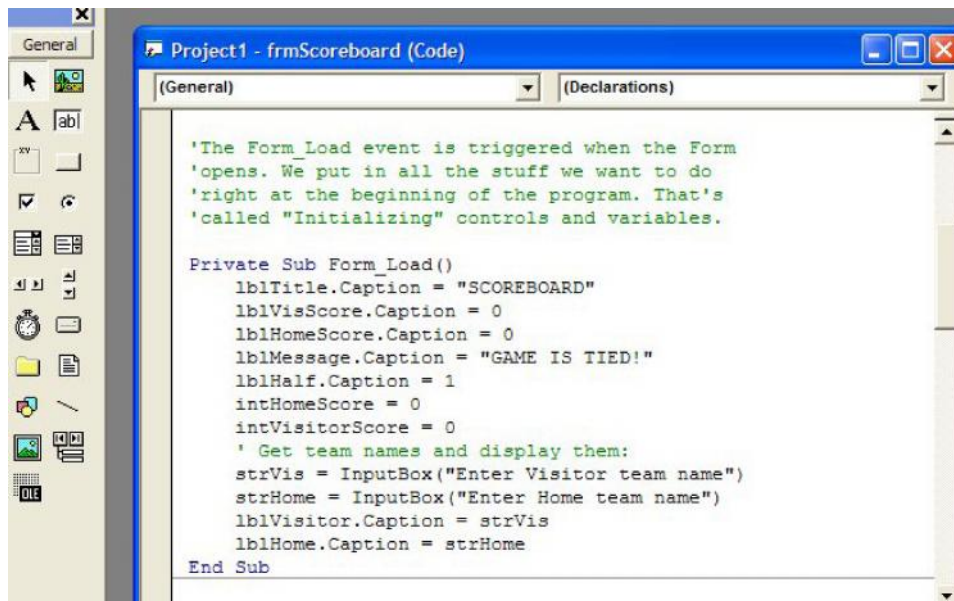


Fig. 1.9 Showing the Code of Form_Load Event

Now we can **Run** it and find something happening. When the Form loads, it will initialize the fields that were specified in the code.

1.3.3 Opening/Saving Running a Visual Basic Project

Creating and Saving a New Program

1. Start VB from the Windows start menu.

The large VB window appears with a **New Project** dialog box (as shown in Figure 1.1). (In case there is no dialog box, bring it up through **File > New Project**).

Select **Standard Exe**. The VB window will show **Project1** with an empty **Form1** (as shown in Figure 1.2).

2. Name the project.

Select **Project > Project1 Properties**. In the **Project Properties** dialog box, change **Project1** to your project name, say Sample.

The title bar on the VB window changes to match.

3. Name the form and set the form caption.

The **Properties** window at the right side of the VB window shows the properties for the form. The first property at the top of the list is **Name**. Change the name to, say **frmSample1**.

The title bar in the Form window changes to match as shown in Figure 1.10.

NOTES



Fig. 1.10 Showing Change in the Title Bar

Scroll down to **Caption** still remaining in the Properties window. Rename the caption with the same name as the project, say **Sample**.

The title bar on the form will change automatically to match.

4. Save the form (you should always name and save the form *before* you save the project).

Select **File** → **Save frmSample1 As ...**. The **Save File As** and the **Save Project As** dialog boxes appear (as shown in Figures 1.11 and 1.12 respectively). The **Save in** textbox shows the folder's name where the form will be saved. The form file name **frmSample1.frm** should appear in the **File name** box (if not, type in the correct name). Click on **Save**.

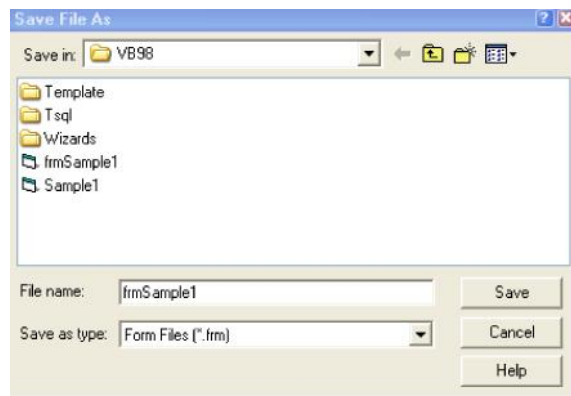


Fig. 1.11 Save File as Dialog Box (to save form file as frmSample1.frm)

The form is saved.

5. Save the project (be sure to name and save the form *before* you save the project).

Select **File** → **Save Project As ...**. The **Save Project As** dialog box appears (Figure 1.12). The **Save in** TextBox should show the correct folder (the one you just created for the form). The **File name** box should show the correct name (**Sample.vbp** in this example). Click on **Save**.

NOTES

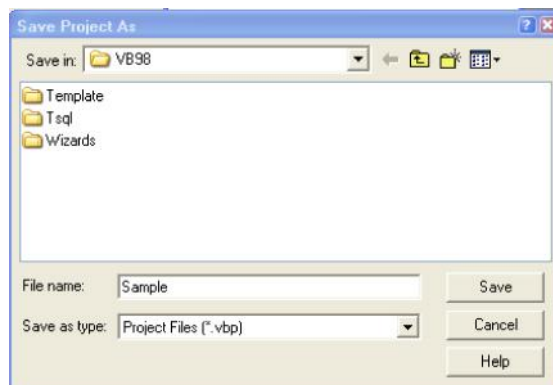


Fig. 1.12 Save Project as Dialog Box (to save project file as Sample.vbp)

The project is saved.

After naming and saving the form and the project, it is better to exit and then restart VB for making sure that the project can be restored before investing a lot of work in it.

Restoring/Opening an Existing Program

1. Start VB.

Find Visual Basic on the Windows Start menu and start it.

The large VB window appears with a **New Project** dialog box. Choose the **Existing** tab (as shown in Figure 1.13). If there is no dialog box, bring it up by **File > Open Project**.

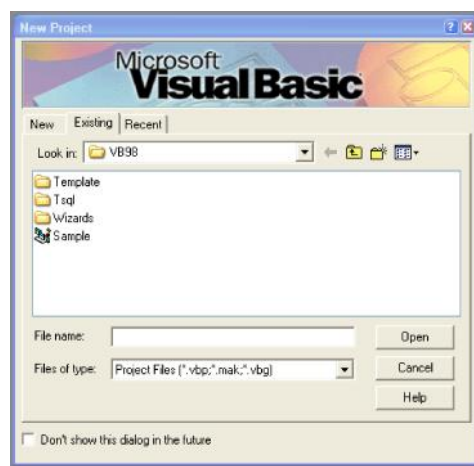


Fig. 1.13 New Project Dialog Box (to choose the Existing Project)

Navigate to the folder that was created earlier. There will be a **.vbp** file for the project created earlier (**Sample.vbp** in this example). Select that file.

Alternatively, for navigating to the project folder, you can use My Computer or Windows Explorer instead of the Start menu. The VB starts and opens that project when the **.vbp** file or the **.frm** file is double clicked.

NOTES

The restored project appears in the VB window. The project name appears in the title bar.

2. Open the form.

An icon for a folder of forms appears in the **Project** window on the right side of the VB window. Open this folder and select the form you saved. Click on the **View Code** and **View Object** icons for displaying the code or form layout windows.

Now you are ready to add controls and code to the form. At any time, you can use **File > Save frmSample1** and **File > Save Project** to save your changes to the form and the project without going through the **Save As ...** dialog.

Running the Program

There are various ways of running your program:

- Press the **F5** key.
- On the VB menu bar, go to Run and click Start (**Run > Start**).
- On the VB toolbar, click the VB Run icon (the arrow).

The program window appears and looks similar to the form that was designed. The window controls are active. The program behaves like any other window on the desktop while it is running; you can minimize it, move it, etc.

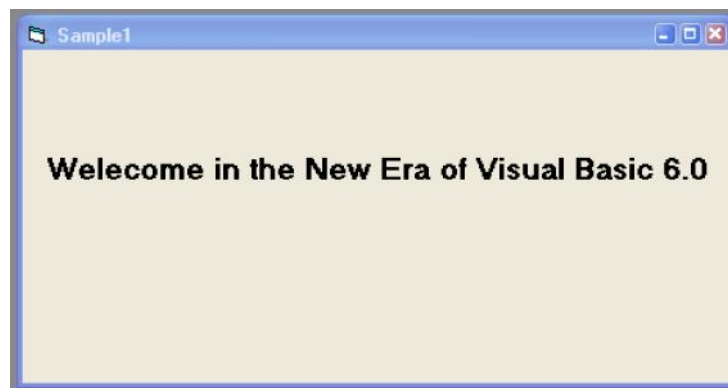


Fig. 1.14 Output of Sample Program

From the **design mode**, the indicator on the Visual Basic (VB) window title bar changes to the **run mode**. Several items in the VB window disappear during the run mode and several menu and ToolBar operations are not enabled.

Stopping the Program

There are various ways of stopping your program:

- In the window of your program, click the **Exit** button or menu (if you are provided one).

- In the title bar of your program window, click the **Close** button (VB always provides one).
- On the VB menu bar, go to Run and click Stop (**Run > Stop**).
- On the VB toolbar, click the VB **Stop** icon (the box).

From the **run mode**, the indicator on the VB window title bar changes to the **design mode**. Items in the VB window reappear, and menu and toolbar operations are enabled again.

Examining the Program in the Break Mode

As the program runs, there are several ways of put VB in the break mode:

- On the VB menu bar, go to Run and click Break (**Run > Break**).
- On the VB toolbar, click the **Break** icon (the vertical bars).
- Your program reaches a breakpoint or experiences an error.

Printing Visual Basic Project

Surprisingly, there is no Printer control. Unlike most of the things in VB, sending output to the printer can be a tiresome process. It requires sending a long list of instructions to the printer describing exactly the way the output should look like.

Despite the problems sometimes faced with printing, you can easily learn to control all the aspects of printing that include the font of individual characters that is sent by the application to the printer. The control needed for printing provides accuracy and it lets you control all the printing details.

Exiting Visual Basic

The following are the ways to exit VB:

- On the menu bar, go the File and click Exit (**File > Exit**).
- On the VB window title bar, click the **Close** button.

VB will give you a warning if you have forgotten to save your work.

Check Your Progress

9. Give the definition of project in Visual Basic (VB).
10. What is the full form of IDE?
11. Which window displays all forms and modules of the required application?
12. In which window we can write code?
13. In How many modes a VB application can work?
14. Explain the steps required for the Visual Basic application.

NOTES

1.4 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

NOTES

1. Visual Basic is a third-generation event driven programming language from Microsoft known for its Component Object Model (COM).
2. Graphical User Interface (GUI) applications are developed in Visual Basic.
3. Following are the features of Visual Basic :
 - It provides a common programming platform across all MS-Office applications.
 - It offers many tools that provide a quick and easy way to develop an application.
 - It also provides many wizards that can automate tasks or even automate coding.
 - It supports ActiveX Control, with which you can create your own ActiveX control and use it in your application.
 - It has an n -tier architecture.
 - It offers quick error detection/correction.
 - It comes with a full set of objects for 'Drawing' the application.
 - It responds to the action of the mouse and the keyboard.
 - It has access to the clipboard and printer.
 - It comes with a full array of string handling, mathematical and graphics functions.
 - It can handle dynamic and fixed variables and control arrays.
 - It supports sequential and random access files.
 - It has powerful tools for database access.
 - It has a package and deployment wizard to make distribution of applications easy.
4. There were a total of six different versions of Visual Basic released (1.0, 1.0 for MS-DOS, 2.0, 3.0, 4.0, 5.0, and 6.0). The last version of Visual Basic, version 6.0 was released in 1998. After Visual Basic 6.0, Microsoft moved Visual Basic to the .NET Framework.
5. The Enterprise Edition is the most powerful version of VB 6.0.
6. An event is something that occurs when button being clicked by the user or a form being opened. The operation is driven by event because every execution is a result of some event.
7. The TextBox control supports several events, such as Change, Click, MouseMove and others, that will be listed in the drop-down list of Properties in the code window for the TextBox control.

8. You can start VB 6.0 by the following three steps:
Click at Program → Microsoft VB Studio → Microsoft VB6.0
9. A project in VB is a collection of several types of files that make up your program. An application is the final program that is used by people.
10. The full form of IDE is Integrated Development Environment.
11. Project window displays all forms and modules of the required application?
12. In Code editor window we can write code.
13. A Visual Basic application works in Design mode, Run mode and Break/Suspended mode.
14. Generally, the following steps are required for building a VB application:
Step 1: Designing the interface
Step 2: Setting properties of the controls (objects)
Step 3: Writing the procedures of the events

NOTES

1.5 SUMMARY

- The Visual Basic (VB) programming environment provides all the features that are required to develop a graphical user interface.
- It provides a common programming platform across all MS-Office applications.
- It offers many tools that provide a quick and easy way to develop an application.
- It supports ActiveX Control, with which you can create your own ActiveX control and use it in your application.
- It has a package and deployment wizard to make distribution of applications easy.
- The original Visual Basic for Disk Operating System (DOS) and Visual Basic for Windows were introduced in 1991.
- Visual Basic 3.0 was a powerful language but it was reasonably small.
- The addition of classes to the language in Visual Basic 4.0 made it much more complex.
- Though more support for database programming and other topics like custom controls in Versions 4.0, 5.0 and 6.0 made it even more complex, Visual Basic remained an easy to understand language.
- Visual Basic .NET speeded up the expansion of Visual Basic immensely. The .NET framework added various powerful new tools.

NOTES

- Visual Basic 6.0 for Windows requires Microsoft Windows 95/Windows NT 3.51, a 486 processor and a minimum RAM of 16 MB.
- In a VB project, the occurring processes have to be associated with events.
- An event is something that occurs when button being clicked by the user or a form being opened. The operation is driven by event because every execution is a result of some event.
- The TextBox control supports several events, such as Change, Click, Mouse Move and others that will be listed in the drop-down list of Properties in the code window for the TextBox control.
- The code entered in the Change event fires in case of a change in the contents of the TextBox.
- Methods make an object do something. Events are what occur when something is done by an object.
- VB 6.0 was released in mid-1998. It is started either by clicking the VB icon or Programs → Microsoft - VB 6.0 → VB 6.0.
- The Project Explorer window provides forms, classes and modules collectively in a group as elements to the programmer.
- For opening a VB environment and working with it, select and click on Microsoft Visual Basic 6.0 in the start menu.
- A project in VB is a collection of several types of files that make up your program. An application is the final program that is used by people.
- The working environment in VB incorporates several different functions such as editing, designing, compiling and debugging within a common environment.
- TitleBar is the topmost bar displaying the title of the project.
- Form is the main feature of the VB application; it is the 'Window' or 'Screen' that users interact with. It can be considered as a 'Canvas' on which the user places the objects that form an application.
- ToolBox window has a set of controls used for customizing forms. These controls help the user in creating an interface between the user and the application.
- Properties window helps in changing the property settings or characteristics of the form itself and also the elements of visual interface on the form. There are two columns in the Properties window: the first is the property name that cannot be changed and the second is the setting of the property that can be changed.
- Project explorer or project window shows the list of forms and modules in a project. A VB Project Explorer consists of a number of forms, modules and controls that make up an application.

- Form layout window shows how big a form is in relation to the screen. It also displays the position of the window where it will be displayed when the project is run.
- Code editor window is the place for writing VB code for your application. By code we mean language statements, declarations and constants. For entering application code, the code editor window serves as an editor.
- While an application is being created or designed, it is in the design mode. When the application is executing, it is said to be in the run mode.
- An application is an assortment of objects that work together for accomplishing something useful. The application in VB is called Project. A project could be calculation of mortgages, management of a video store, the payroll for 1000 employees or a dating service.
- An object is a part of software with properties and functions that can be changed. A window is an object with properties like color, size, position on the screen, etc. The function of a window, also known as methods.
- The code was written by somebody and put together in a small package called window object.
- The new Help Topics dialog box is the 'Way In' to Help information in an application. To invoke the Help dialog box, choose the content command from the Help menu.
- Contents tab helps in displaying the Help topics by title in the categories set up by you. Jumps to topics as well as the capability of running macros can be created in other Help files.
- We can make the bunch of controls on the form more attractive by changing the Properties of the controls in the Properties window.
- The program window appears and looks similar to the form that was designed. The window controls are active. The program behaves like any other window on the desktop while it is running; you can minimize it, move it, etc.
- From the design mode, the indicator on the Visual Basic (VB) window title bar changes to the run mode. Several items in the VB window disappear during the run mode and several menu and toolbar operations are not enabled.
- From the run mode, the indicator on the VB window title bar changes to the design mode. Items in the VB window reappear, and menu and toolbar operations are enabled again.
- Surprisingly, there is no Printer control. Unlike most of the things in VB, sending output the printer can be a tiresome process. It requires sending a long list of instructions to the printer describing exactly the way the output should look like.

NOTES

NOTES

- Despite the problems sometimes faced with printing, you can easily learn to control all the aspects of printing that include the font of individual characters that is sent by the application to the printer.
- The control needed for printing provides accuracy and it lets you control all the printing details.

1.6 KEY WORDS

- **Application:** A collection of objects that work together to accomplish something useful. In VB, the application is called Project.
- **Object:** A piece of software that has properties and functions that can be manipulated.
- **Title bar:** The topmost bar displaying the title of the project.
- **Form:** The heart of a VB application; it is the 'Screen' or 'Window' that the users interact with.
- **Properties window:** The window which lets the user change the characteristics or the property settings of the form itself as well as of the visual interface elements on the form.
- **Project explorer or project window:** The window which shows the list of forms and modules in a project.
- **Form layout window:** The window which shows how big a form is in relation to the screen.
- **Code editor windows:** The window where the VB code is written for an application.

1.7 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. Explain about the Visual Basic IDE.
2. Describe any four features of Visual Basic.
3. Explain about the properties and methods.
4. Visual Basic was developed in which year?
5. State about the new features of Visual Basic 6.0.
6. What is Standard EXE?
7. Name the different types of projects that can be created in VB.
8. Why we use ActiveX Control?
9. Define the term Object.

Long-Answer Questions

1. Discuss about the Project in VB 6.0? How many types of project's are there in Visual Basic? Explain each of them briefly.
2. Analyse about the Visual Basic IDE?
3. Discuss about the Visual Basic development process in detail.
4. Explain how to opening, saving and running the Visual Basic Project.
5. Elaborate briefly on the opening an existing program in Visual Basic.
6. Create the simple Hello World application in Visual Basic working environment. Giving code and output.
7. Briefly describe each of the following IDE features:
 - a. Toolbox
 - b. Form
 - c. ToolBox Window
 - d. Explorer window

NOTES

1.8 FURTHER READINGS

- Cornell, Gary. 1998. *Visual Basic 6 from the Ground Up*. New Delhi: Tata McGraw-Hill.
- Warner, Scott L. 1998. *Teach Yourself Visual Basic 6*. New Delhi: Tata McGraw-Hill.
- Jerke, Noel. 1999. *Visual Basic 6 - The Complete Reference*. New York: McGraw-Hill.
- Smith, Eric A., Valor Whisler and Hank Marquis. 1998. *Visual Basic 6 Programming Bible*. New Jersey: John Wiley & Sons, Inc.
- Azam, M. 2001. *Programming with Visual Basic*. New Delhi: Vikas Publishing House Pvt. Ltd.
- Manchanda, Mahesh. 2009. *Visual Programming*. New Delhi: Vikas Publishing House Pvt. Ltd.
- Balena, Francesco. 1999. *Programming Microsoft Visual Basic 6.0*. Bangalore: WP Publishers and Distributors (P) Ltd.
- Petroutsos, Evangelos. 1998. *Mastering Visual Basic 6, 1st Edition*. New Delhi: BPB Publications.
- Deitel, Harvey M., Paul J. Deitel and T. Tem R. Nieto. 1999. *Visual Basic 6: How to Program*. New Jersey: Prentice-Hall.

NOTES

Norton, Peter. 1998. Peter Norton's Complete Guide to Visual Basic 6. New Delhi: Techmedia.

Reselman, Bob and Richard A. Peasley. 1998. Using Visual Basic 6. New Jersey: Pearson Education (Que Publishing).

Donald, Bob and Oancea Gabriel. 1999. Visual Basic 6 from Scratch. New Delhi: Prentice-Hall of India.

UNIT 2 USING THE TOOLBOX

- 2.0 Introduction
- 2.1 Objectives
- 2.2 Working with Toolbar
- 2.3 Use of the Toolbox
- 2.4 Project Programming Structure in Visual Basic Application
- 2.5 Event and Event Driven Procedures
- 2.6 Answers to Check Your Progress Questions
- 2.7 Summary
- 2.8 Key Words
- 2.9 Self-Assessment Questions and Exercises
- 2.10 Further Readings

NOTES

2.0 INTRODUCTION

The **Toolbox** window displays controls that you can add to Visual Studio projects. To customize the **Toolbox** by adding pages to it or by adding controls by using the **Additional Controls** command from the **Tools** menu.

In Visual Basic (VB) **Toolbox** is a palette of controls **CommandButtons**, **CheckBox**, **TextBox**, **ListBox**, and **ComboBox** that can be dragged-and dropped onto a user form. For VB, to make the **Toolbox** available, form must have open in design mode. **Toolbox** appears in conjunction with designer views, such as the designer view of a XAML file or a Windows Forms App project. **Toolbox** displays only those controls that can be used in the current design mode.

A toolbar is a bar that displays in the top section under the main menu. A toolbar is a classic control container. It can host text, buttons, etc. No Windows application would truly be complete without at least one toolbar. Toolbars provide a quick way for users to perform tasks without having to navigate the application's menu system. Toolbars are referred to as **toolStrips** in Visual Basic.

A Visual Basic program is built up from standard building blocks. A *project* in turn can contain one or more assemblies. Each *assembly* is compiled from one or more source files. A *source file* provides the definition and implementation of classes, structures, modules, and interfaces, which ultimately contain all the VB code.

An event is an action or occurrence recognized by software often originating asynchronously from the external environment, that may be handled by the software. Event driven programming is when a program is designed to respond to user engagement in various forms. It is known as a programming paradigm in which the flow of program execution is determined by 'Events'. Events are any user interaction, such as a click or key press, in response to prompt from the system. In Visual Basic event procedure is a procedure that executes on a specific event, such as click button event or action by a Visual Basic (VB) object.

In this unit, you will study about the toolbar, toolbox, project programming structure of Visual Basic application, event and event driven procedures in Visual Basic.

NOTES

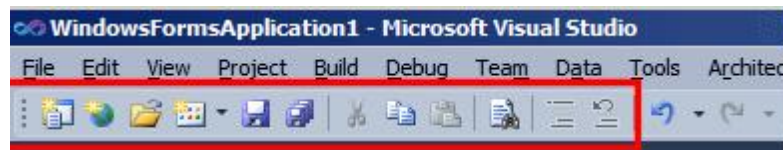
2.1 OBJECTIVES

After going through this unit, you will be able to:

- Learn about the basic structure of Visual Basic program
- Understand the Visual Basic Projects
- Able to work with toolbox
- Explain about the Visual Basic environment
- Define the term event and event driven procedure

2.2 WORKING WITH TOOLBAR

A 'Toolbar' is a bar that displays in the top section under the main menu. Following are the screenshot.



A toolbar is a classic control container. It can host text, buttons, etc. It is up to you to decide whether your application needs a toolbar and what you want to position on it. A Toolbar is primarily a container, which itself means nothing and doesn't even do anything. The controls you position on it give it meaning. Still, because of its position, it enjoys some visual characteristics but also imposes some restrictions to its objects. Although it is a container, a Toolbar must be hosted by another container. For this reason, the dimensions, namely the width, of a toolbar are restricted to those of its host. When you add a toolbar to a form, it automatically positions itself in the top section of the form and uses the same width as the form. This means that the default `Dock` value of a Toolbar is `displayed at the top`.

Toolbar is created visually or programmatically. To create toolbars, the .NET Framework provides a class named `ToolStrip`. The `ToolStrip` class is derived from the `Scrollable Control` class and implements both the `IComponent` and the `IDisposable` interfaces.

Therefore, to start a toolbar, declare a variable of this class. Because a Toolbar is hosted by a container, namely a form, if you want to display it, you should (must) add it to the `controls` collection of its host. Following is the code for creating toolbar:

```
Imports System.Windows.Forms
```

```
Inherits System.Windows.Forms.Form
```

```
Dim tbrStandard As ToolStrip
```

```
Public Sub New()
```

```
    tbrStandard = New ToolStrip
```

```
    Controls.Add(tbrStandard)
```

```
End Sub
```

```
<STAThread(>
```

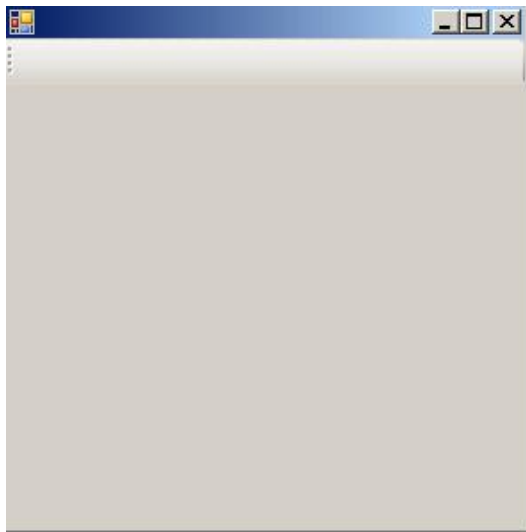
```
Public Shared Function Main() As Integer
```

```
    Application.Run(New Exercise)
```

```
    Return 0
```

```
End Function
```

```
End Class
```



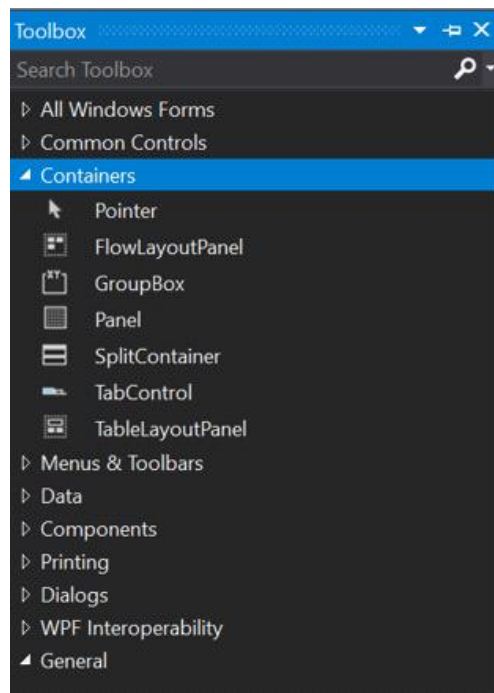
For creating a toolbar visually, select the Container section, the Toolbox provides a ToolStrip button that you can click and click your form.

2.3 USE OF THE TOOLBOX

The toolbox window displays controls that you can add to Visual Studio projects. To open toolbox, choose **View** → **Toolbox** from the menu bar, or press **Ctrl+Alt+X**.

NOTES

NOTES



You can drag and drop different controls onto the surface of the designer you are using, and to resize and position the controls.

Toolbox appears in conjunction with designer views, such as the designer view of a XAML file or a Windows Forms App project. Toolbox displays only those controls that can be used in the current designer. You can search within Toolbox to further filter the items that appear.

2.4 PROJECT PROGRAMMING STRUCTURE IN VISUAL BASIC APPLICATION

Without knowing the basic software languages like C and C++ it is easy to learn VB.NET why because its drag and drop feature , but while making an application in console it is mandatory to know about the flow of the code so following are the steps to be placed in the code.

A VB.NET program consists of the following:

- Namespace Declaration
- One or More Procedures
- A Class or Module
- Variables
- The Main Procedure
- Comments
- Statements and Expressions

Hello World Program

Using the Toolbox

Step 1 Create a new console application.

Step 2 Add the following code:

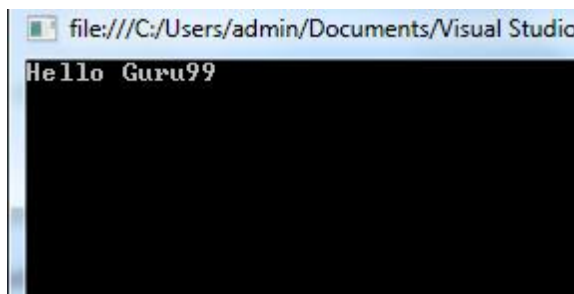
```
Imports System
Module Module1

    'Prints Hello Guru99
    Sub Main()

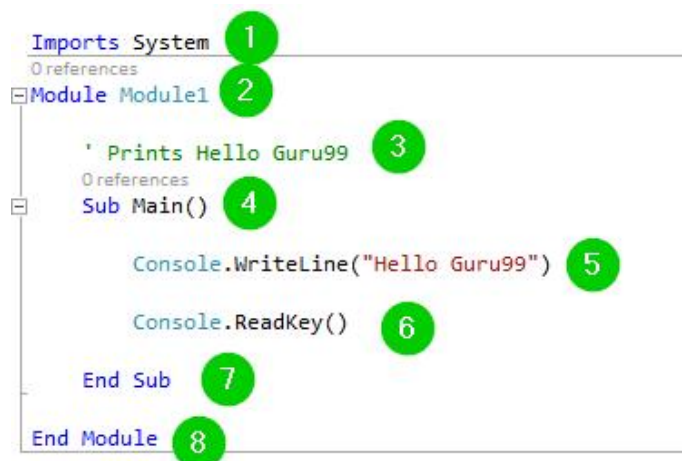
        Console.WriteLine("Hello Guru99")
        Console.ReadKey()

    End Sub
End Module
```

Step 3 Click the Start button from the toolbar to run it. It should print the following on the console: Beginner Tutorial



Following are the various parts of the above program:



NOTES

NOTES**Explanation of Code**

1. This is called the namespace declaration. What we are doing is that we are including a namespace with the name `System` into our programming structure. After that, we will be able to access all the methods that have been defined in that namespace without getting an error.
2. This is called a module declaration. Here, we have declared a module named `Module1`. Visual Basic (VB) is an object-oriented language. Hence we must have a class module in every program. It is inside this module that you will be able to define the data and methods to be used by your program.
3. This is a comment. To mark it as a comment, we added a single quote (‘) to the beginning of the sentence. The VB compiler will not process this part. The purpose of comments is to improve the readability of the code. Use them to explain the meaning of various statements in your code. Anyone reading through your code will find it easy to understand.
4. A VB module or class can have more than one procedures. It is inside procedures where you should define your executable code. This means that the procedure will define the class behaviour. A procedure can be a Function, Sub, Get, Set, AddHandler, Operator, RemoveHandler, or RaiseEvent. In this line, we defined the `Main` sub-procedure. This marks the entry point in all VB programs. It defines what the module will do when it is executed.
5. This is where we have specified the behaviour of the primary method. The `WriteLine` method belongs to the `Console` class, and it is defined inside the `System` namespace. Remember this was imported into the code. This statement makes the program print the text `Hello Guru99` on the console when executed.
6. This line will prevent the screen from closing or exiting soon after the program has been executed. The screen will pause and wait for the user to perform an action to close it.
7. Closing the main sub-procedure.
8. Ending the module.

Classes

In VB, we use classes to define a blueprint for a data type. It does not mean that a class definition is a data definition, but it describes what an object of that class will be made of and the operations that we can perform on such an object.

An object is an instance of a class. The class members are the methods and variables defined within the class.

To define a class, we use the `Class` keyword, which should be followed by the name of the class, the class body, and the `End Class` statement. This is described in the following syntax:


```
[ <attributelist> ] [ accessmodifier ] _
Class name
```

```
[ Inherits classname ]
```

```
[ statements ]
```

```
End Class
```

Here,

- The `attributeList` denotes a list of attributes that are to be applied to the class.
- The `accessModifier` is the access level of the defined class. It is an optional parameter and can take values like `Public`, `Protected`, `Protected Friend`, `Friend`, and `Private`.
- The `Inherits` denotes any parent class that it inherits.

Following is example code to create a class in VB -

Step 1 Create a new console application.

Step 2 Add the following code:

```
Imports System
Module Module1

    Class Figure
        Public length As Double

        Public breadth As Double
    End Class
    Sub Main()
        Dim Rectangle As Figure = New Figure()
        Dim area As Double = 0.0

        Rectangle.length = 8.0

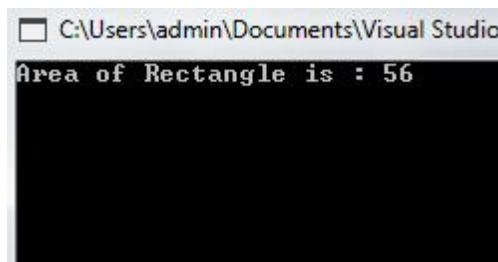
        Rectangle.breadth = 7.0
        area = Rectangle.length * Rectangle.breadth
        Console.WriteLine("Area of Rectangle is : {0}",
area)

        Console.ReadKey()
    End Sub
End Module
```

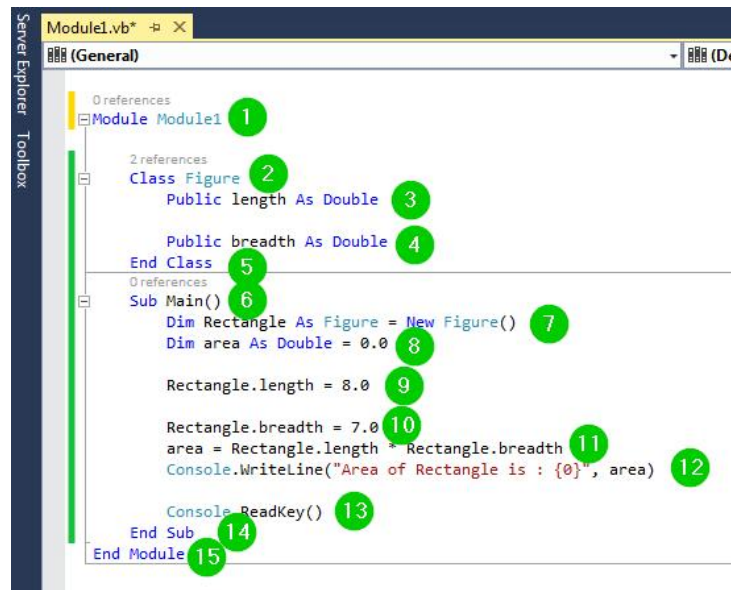
Step 3 Run the code by clicking the Start button from the Toolbar. You will get the following window:

NOTES

NOTES



We have used the following code:



Explanation of Code

1. Creating a module named Module1.
2. Creating a class named Figure.
3. Creating a class member named length of type Double. Its access level has been set to public meaning that it will be accessed publicly.
4. Creating a class member named breadth of type Double. Its access level has been set to public meaning that it will be accessed publicly.
5. Ending the class.
6. Creating the main sub procedure.
7. Creating an object named Rectangle. This object will be of type figure, meaning that it will be capable of accessing all the members defined inside the Figure class.
8. Defining a variable named area of type Double and initializing its value to 0.0.
9. Accessing the length property defined in the Figure class and initializing its value to 8.0.

10. Accessing the breadth property defined in the Figure class and initialize its value to 7.0.
11. Calculating the area of the rectangle by multiplying the values of length and breadth. The result of this calculation will be assigned to the area variable.
12. Printing some text and the area of the rectangle on the console.
13. Pausing the console waiting for a user to take action to close it.
14. Ending the sub-procedure.
15. Ending the class.

NOTES

Structure

A structure is a user-defined data type. The Structure provide us with a way of packaging data of different types together. A structure is declared using the Structure keyword. Example to create a structure in VB:

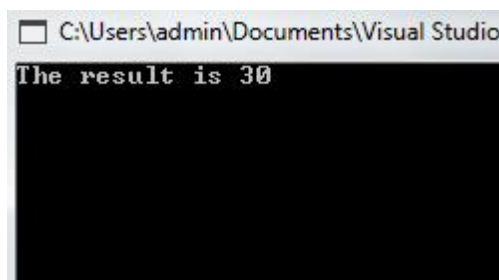
Step 1 Create a new console application.

Step 2 Add the following code:

```
Module Module1
    Structure Struct
        Public x As Integer
        Public y As Integer
    End Structure
    Sub Main()
        Dim st As New Struct
        st.x = 10
        st.y = 20
        Dim sum As Integer = st.x + st.y
        Console.WriteLine("The result is {0}", sum)
        Console.ReadKey()

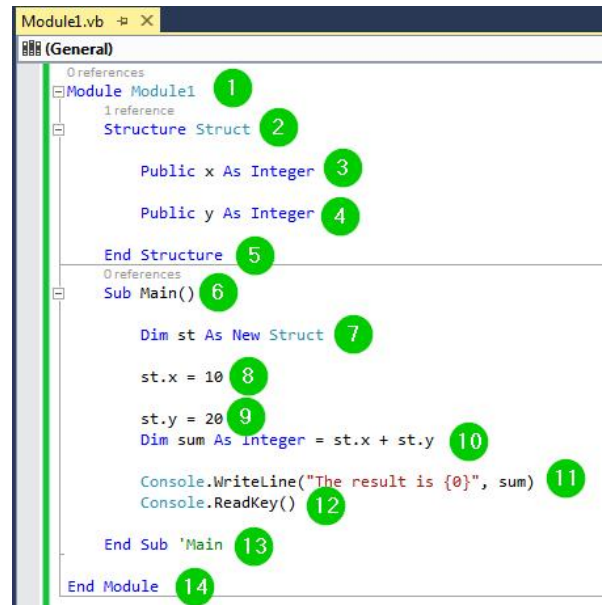
    End Sub
End Module
```

Step 3 Run the code by clicking the Start button from the Toolbar. You should get the following window:



We have used the following code:

NOTES



```

Module1.vb
General
0 references
Module Module1 1
1 reference
Structure Struct 2
    Public x As Integer 3
    Public y As Integer 4
End Structure 5
0 references
Sub Main() 6
    Dim st As New Struct 7
    st.x = 10 8
    st.y = 20 9
    Dim sum As Integer = st.x + st.y 10
    Console.WriteLine("The result is {0}", sum) 11
    Console.ReadKey() 12
End Sub 'Main 13
End Module 14
  
```

Explanation of Code

1. Creating a module named Module1.
2. Creating a structure named Struct.
3. Creating a variable x of type integer. Its access level has been set to Public to make it publicly accessible.
4. Creating a variable y of type integer. Its access level has been set to Public to make it publicly accessible.
5. End of the structure.
6. Creating the main sub procedure.
7. Creating an object named st of type Struct. This means that it will be capable of accessing all the properties defined within the structure named Struct.
8. Accessing the variable x defined within the structure Struct and initializing its value to 10.
9. Accessing the variable y defined within the structure Struct and initializing its value to 20.
10. Defining the variable sum and initializing its value to the sum of the values of the above two variables.
11. Printing some text and the result of the above operation on the console.
12. Pausing the console window waiting for a user to take action to close it.
13. End of the main sub procedure.
14. End of the module.

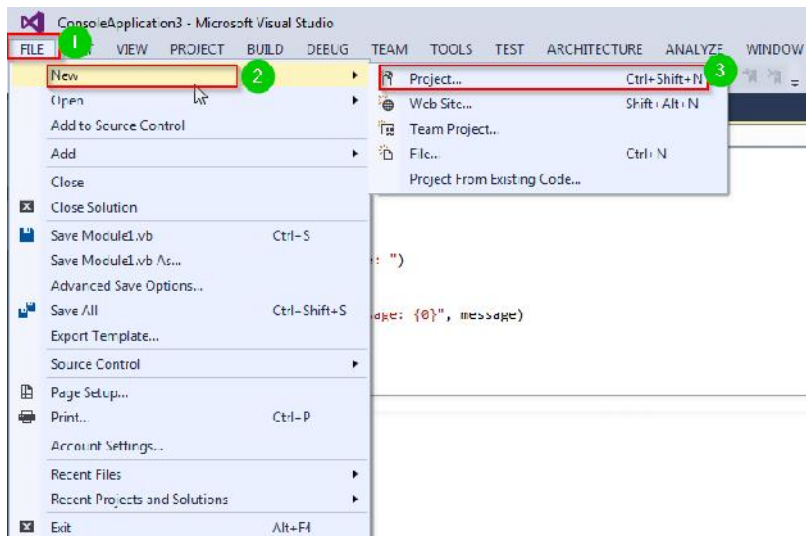
Using Microsoft Visual Studio IDE

Using the Toolbox

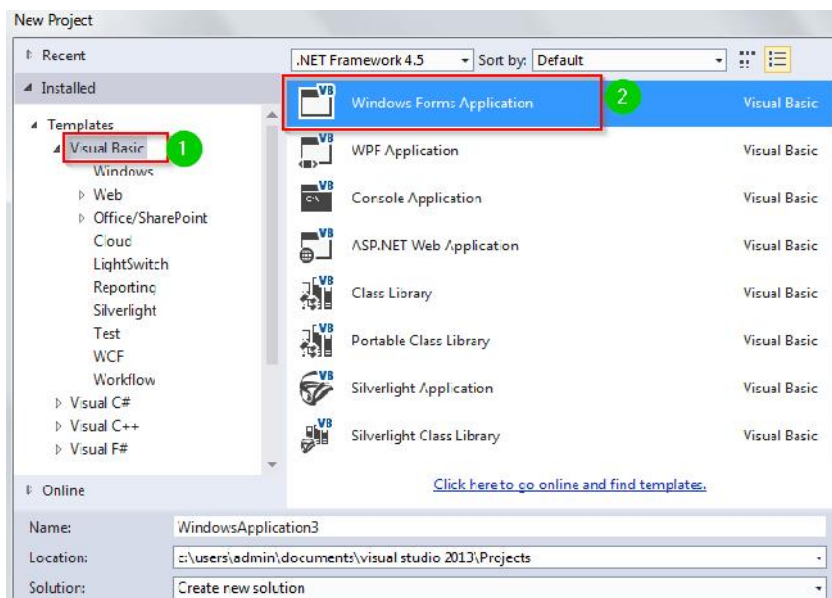
IDE stands for Integrated Development Environment. That is where the code is written. The most popular form of IDE for VB programming is Microsoft Visual Studio.

To write your code, you need to create a new project. The following steps can help you achieve this.

Step 1 Open Visual Studio, and click the File menu, Choose New then Project from the Toolbar.



Step 2 On the new window, click Visual Basic from the left vertical navigation pane. Choose Windows Forms Application.



NOTES

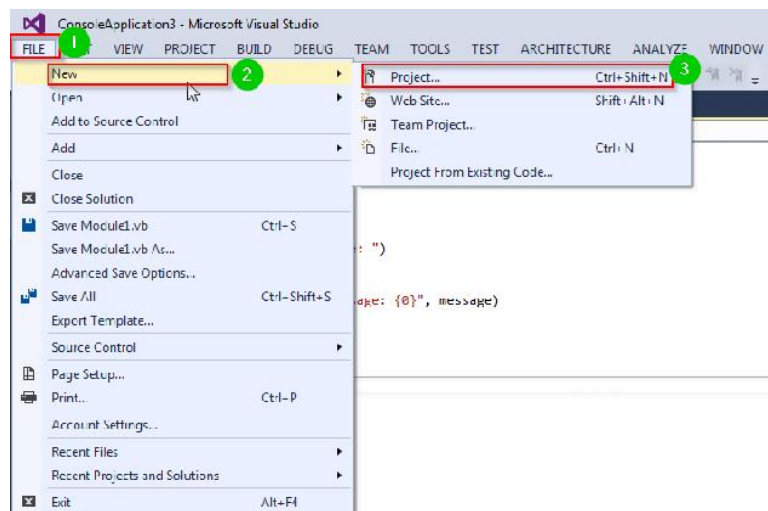
NOTES

Step 3 Give it a name and click the OK button. The project will be created.

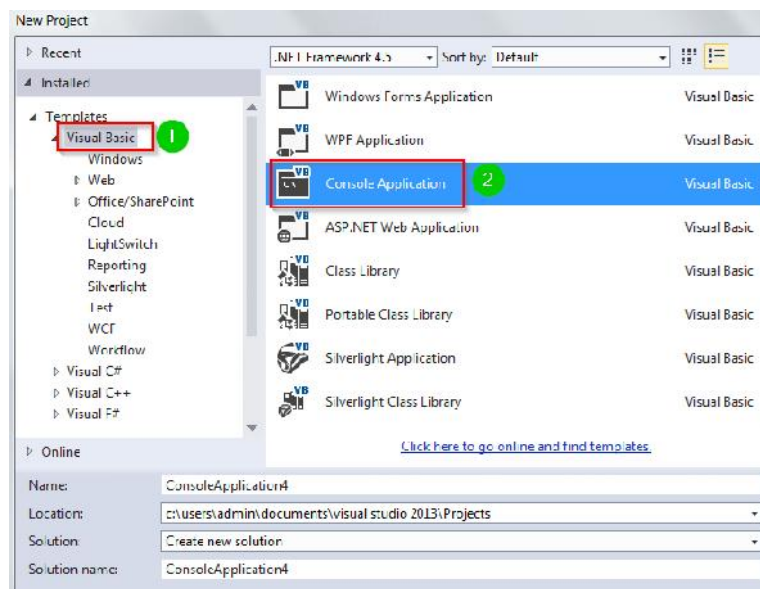
You will have a Windows type application project developed. By dragging and dropping elements, this form of project will allow you to create a Graphical User Interface.

An application that runs on the console would need to be built. This needs that you create a project for Console Program. The following steps will help you in achieving this.

Step 1 Open Visual Studio, and click the File menu, Choose New then Project from the Toolbar.



Step 2 On the new window, click Visual Basic from the left vertical navigation pane. Choose Console Application.



Step 3 Give it a name and click the OK button. The project will be created.

Check Your Progress

1. Explain the term Toolbar.
2. Why we use Toolbar in Visual Basic?
3. List the two types for creating Toolbar in Visual Basic application.
4. State about the Toolbox.
5. Give the definition of object.
6. Which keyword is used to define a class in Visual Basic?
7. Write down the syntax of class.
8. What is `attributeList` and `AccessModifier`?
9. Why we use keyword `Inherits`?
10. Elaborate on the structure of a program.

NOTES**2.5 EVENT AND EVENT DRIVEN PROCEDURES**

In a Visual Basic (VB) project, the occurring processes have to be associated with events. An event is something that occurs when a button is clicked by the user or a form is opened. The operation is driven by event because every execution is a result of some event. The programmer's role is to anticipate the events and to write the code for execution during the occurrence of the event.

A VB application is interactive because of the constant interaction of the user with the program. Visual Basic (VB) programs are built around events, which are different incidents that can occur in a program. This will become clearer when VB is compared to procedural programming. In procedural languages, an application is written and executed by a logical checking of the program through the program statements, one after another. The control can be transferred to some other point in a program on a temporary basis. In applications that are event driven, the execution of the program statements occurs only when a certain event calls a particular part of the code assigned to it.

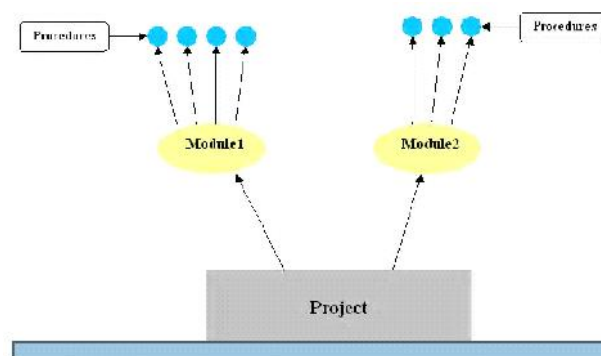
We can look at a `TextBox` control and a few of its related events to understand the concept of event driven programming. The `TextBox` control supports several events, such as `Change`, `Click`, `Mouse Move` and others that will be listed in the drop-down list of Properties in the code window for the `TextBox` control. Let us look at a few of them:

- The code entered in the `Change` event fires in case of a change in the contents of the `TextBox`.
- The `Click` event fires when the `TextBox` control is clicked.
- The `Mouse Move` event fires when the mouse is moved over the `Textbox`.

NOTES

While working on Visual Basic (VB), many times we need to create an application or a project that consists of thousands of lines. Now, it is very difficult and time consuming to control, to find out logical and syntax error, and perform error handling. Therefore, a project is divided into subparts and each subpart contains a portion of the complete code, and is a complete application in itself. These subparts are known as modules of the project. These modules can further be divided into subparts and these subparts are known as procedures.

Below figure shows the hierarchy of a project containing modules and procedures.



Event Procedures

Event procedure is a procedure that executes on a specific event, such as click button event or action by a Visual Basic (VB) object.

Now create an event procedure to display a message box with the following message

"You are in the Event procedure process".

Creating Form

1. Open a new project in the Visual Basic (VB).
2. Double-click the `CommandButton` in the Toolbox to create a `CommandButton` on the `Form1` window.

Adding Code to the Form

1. Double-click the `CommandButton` to open the Code window.
2. Enter the following lines of code in the Code window:

```
Private Sub Command1_Click ()
    Eventproc 'calling the procedure named Eventproc
End Sub
```

3. Now, to create an event procedure, select **Tools** → **Add Procedure** on the Menu bar to display the Add Procedure Dialog Box.

Below figure shows the Add Procedure Dialog Box.



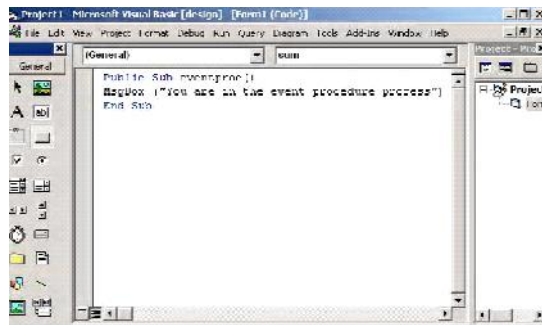
NOTES

4. Enter a name in the Name TextBox for the event procedure, for example eventproc.
5. Click **OK** to display the following lines of code in the Code window:

```
Public Sub eventproc()
End Sub
```
6. Enter the following lines of code in the Code window:

```
Public Sub eventproc()
Msgbox("You are in the event procedure process")
End Sub
```

Below figure shows the Code window for the event procedure.



Executing the Form

1. Press F5 to compile and execute the form. After execution, the output of the form is displayed on the computer screen.

Below figure shows the compiled form.



NOTES

- Now, click the `CommandButton` to display the output.
Below figure shows output message box for the event procedure.



In this program, `Command1` is working as a module, which calls the sub procedure `subproc`. The sub procedure is displaying a message on the message box.

Check Your Progress

- What is event in Visual Basic?
- Explain about the event procedure.
- Which key is used to execute the program in Visual Basic?

2.6 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

- A Toolbar is a bar that displays in the top section under the main menu.
- A Toolbar is a classic control container. It can host text, buttons, etc.
- Toolbar is created visually or programmatically.
- The `Toolbox` window displays controls that you can add to Visual Studio projects.
- An object is an instance of a class.
- `Class` keyword is used to define a class in Visual Basic.
- Following are the syntax for class :

```
[ <attributelist> ] [ accessmodifier ] _
Class name
    [ Inherits classname ]
    [ statements ]
End Class
```
- The `attributeList` denotes a list of attributes that are to be applied to the class. The `accessModifier` is the access level of the defined class. It is an optional parameter and can take values like `Public`, `Protected`, `Protected Friend`, `Friend`, and `Private`.
- The `Inherits` denotes any parent class that it inherits.
- A structure is a user-defined data type. Structures provide us with a way of packaging data of different types together. A structure is declared using the `Structure` keyword.

11. An event is something that occurs when a button is clicked by the user or a form is opened. The operation is driven by event because every execution is a result of some event.
12. Event procedure is a procedure that executes on a specific event, such as click button event or action by a VB object.
13. F5 is used to execute the program in Visual Basic.

NOTES

2.7 SUMMARY

- A Toolbar is a bar that displays in the top section under the main menu.
- A Toolbar is a classic control container. It can host text, buttons, etc. It is up to you to decide whether your application needs a Toolbar and what you want to position on it.
- A Toolbar is primarily a container, which itself means nothing and doesn't even do anything. The controls you position on it give it meaning. Still, because of its position, it enjoys some visual characteristics but also imposes some restrictions to its objects. Although it is a container, a Toolbar must be hosted by another container.
- When you add a Toolbar to a form, it automatically positions itself in the top section of the form and uses the same width as the form. This means that the default `Dock` value of a Toolbar is `Top`.
- Toolbar is created visually or programmatically.
- To create toolbars, the .NET Framework provides a class named `ToolStrip`. The `ToolStrip` class is derived from the `ScrollableControl` class and implements both the `IComponent` and the `IDisposable` interfaces.
- Therefore, to start a Toolbar, declare a variable of this class. Because a Toolbar is hosted by a container, namely a form.
- For creating a Toolbar visually, select the Container section, the Toolbox provides a `ToolStrip` button that you can click and click your form.
- Without knowing the basic software languages like C and C++ it is easy to learn VB.NET why because its drag and drop feature.
- The `Toolbox` window displays controls that you can add to Visual Studio projects.
- In VB, we use classes to define a blueprint for a data type. It does not mean that a class definition is a data definition, but it describes what an object of that class will be made of and the operations that we can perform on such an object.
- An object is an instance of a class. The class members are the methods and variables defined within the class.
- To define a class, we use the `Class` keyword, which should be followed by the name of the class, the class body, and the `End Class` statement.

NOTES

- The `attributeList` denotes a list of attributes that are to be applied to the class.
- The `accessModifier` is the access level of the defined class. It is an optional parameter and can take values like `Public`, `Protected`, `Protected Friend`, `Friend`, and `Private`.
- The `Inherits` denotes any parent class that it inherits.
- A structure is a user-defined data type.
- The `Structure` provide a way of packaging data of different types together.
- A structure is declared using the `Structure` keyword.
- IDE stands for Integrated Development Environment. That is where the code is written. The most popular form of IDE for VB programming is Microsoft Visual Studio.
- In a Visual Basic (VB) project, the occurring processes have to be associated with events.
- An event is something that occurs when a button is clicked by the user or a form is opened.
- The operation is driven by event because every execution is a result of some event. The programmer's role is to anticipate the events and to write the code for execution during the occurrence of the event.
- A VB application is interactive because of the constant interaction of the user with the program. Visual Basic (VB) programs are built around events, which are different incidents that can occur in a program.
- In procedural languages, an application is written and executed by a logical checking of the program through the program statements, one after another.
- The control can be transferred to some other point in a program on a temporary basis.
- In applications that are event driven, the execution of the program statements occurs only when a certain event calls a particular part of the code assigned to it.
- The code entered in the `Change` event fires in case of a change in the contents of the `TextBox`.
- The `Click` event fires when the `TextBox` control is clicked.
- The `Mouse Move` event fires when the mouse is moved over the `Textbox`.
- While working on Visual Basic (VB), many times we need to create an application or a project that consists of thousands of lines. Now, it is very difficult and time consuming to control, to find out logical and syntax error, and perform error handling.
- A project is divided into subparts and each subpart contains a portion of the complete code, and is a complete application in itself.
- These subparts are known as modules of the project.

- These modules can further be divided into subparts and these subparts are known as procedures.
- Event procedure is a procedure that executes on a specific event, such as click button event or action by a Visual Basic (VB) object.

NOTES

2.8 KEY WORDS

- **Toolbar:** A Toolbar is a bar that displays in the top section under the main menu. Toolbar is a classic control container. It can host text, buttons, etc.
- **Toolbox:** **Toolbox** window displays controls that you can add to Visual Studio projects. To customize the **Toolbox** by adding pages to it or by adding controls by using the Additional Controls command from the Tools menu.
- **Object:** An object is an instance of a class. The class members are the methods and variables defined within the class.
- **AttributeList:** The attributeList denotes a list of attributes that are to be applied to the class.
- **AccessModifier:** The accessModifier is the access level of the defined class. It is an optional parameter and can take values like Public, Protected, Protected Friend, Friend, and Private.
- **Inherits:** The Inherits denotes any parent class that it inherits.
- **Event:** An event is something that occurs when a button is clicked by the user or a form is opened.
- **Event procedure:** Event procedure is a procedure that executes on a specific event, such as click button event or action by a Visual Basic (VB) object.

2.9 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. Explain about the Toolbar.
2. What is Toolbox?
3. List the components of program structure in Visual Basic.
4. Why we use comment in a program?
5. Define about the WriteLine method.
6. Explain Visual Basic Integrated Development Environment (IDE).
7. Which keyword is used for creating structure in Visual Basic?
8. What is event driven programming?
9. Explain mouse press event and double-click event.

NOTES**Long-Answer Questions**

1. Briefly discuss about the Toolbar. Write a code for creating Toolbar in Visual Basic.
2. Write a Visual Basic program to print Hello World on a console with code explanation.
3. Create a Visual Basic program to find out the area of rectangle with code explanation.
4. Describe in detail about the programming structure with help of appropriate example.
5. Elaborate briefly on the create application using Integrated Development Environment (IDE) in Visual Basic with the help of code examples.
6. Write down the steps to create event procedure in Visual Basic.

2.10 FURTHER READINGS

- Cornell, Gary. 1998. *Visual Basic 6 from the Ground Up*. New Delhi: Tata McGraw-Hill.
- Warner, Scott L. 1998. *Teach Yourself Visual Basic 6*. New Delhi: Tata McGraw-Hill.
- Jerke, Noel. 1999. *Visual Basic 6 - The Complete Reference*. New York: McGraw-Hill.
- Smith, Eric A., Valor Whisler and Hank Marquis. 1998. *Visual Basic 6 Programming Bible*. New Jersey: John Wiley & Sons, Inc.
- Azam, M. 2001. *Programming with Visual Basic*. New Delhi: Vikas Publishing House Pvt. Ltd.
- Manchanda, Mahesh. 2009. *Visual Programming*. New Delhi: Vikas Publishing House Pvt. Ltd.
- Balena, Francesco. 1999. *Programming Microsoft Visual Basic 6.0*. Bangalore: WP Publishers and Distributors (P) Ltd.
- Petroutsos, Evangelos. 1998. *Mastering Visual Basic 6, 1st Edition*. New Delhi: BPB Publications.
- Deitel, Harvey M., Paul J. Deitel and T. Tem R. Nieto. 1999. *Visual Basic 6: How to Program*. New Jersey: Prentice-Hall.
- Norton, Peter. 1998. *Peter Norton's Complete Guide to Visual Basic 6*. New Delhi: Techmedia.
- Reselman, Bob and Richard A. Peasley. 1998. *Using Visual Basic 6*. New Jersey: Pearson Education (Que Publishing).
- Donald, Bob and Oancea Gabriel. 1999. *Visual Basic 6 from Scratch*. New Delhi: Prentice-Hall of India.

UNIT 3 PROGRAM DESIGN

- 3.0 Introduction
- 3.1 Objectives
- 3.2 Forms and Controls
 - 3.2.1 Creating and Saving a New Program
 - 3.2.2 Restoring/Opening an Existing Program
 - 3.2.3 Running the Program
 - 3.2.4 Stopping the Program
 - 3.2.6 Printing Visual Basic Project
 - 3.2.7 Exiting Visual Basic
- 3.3 Making Exe Files
- 3.4 Answers to Check Your Progress Questions
- 3.5 Summary
- 3.6 Key Words
- 3.7 Self-Assessment Questions and Exercises
- 3.8 Further Readings

NOTES

3.0 INTRODUCTION

Visual Basic (VB) is an event driven programming language. The 'Visual' part refers to the method used to create the Graphical User Interface (GUI). Rather than writing numerous lines for a code to describe the appearance and location of interface elements.

The 'Basic' part refers to the BASIC language that is used by most of the programmers in the history of computing. Visual Basic has evolved from the BASIC language and now contains several hundred statements, functions and keywords, many of which are directly related to the Windows GUI. Visual Basic allows professionals to implement anything that can be implemented using any other Windows programming language.

The Visual Basic environment is a platform provided by Visual Basic to develop and execute a project. The environment consists of different project templates, control tools and various other options. A project is a collection of files that you use to build an application.

A project file is simply a list of all the files and objects associated with a project, as well as information on the environment options that you set. This information is updated every time you save the project. All these files and objects can be shared by other projects as well. To you have completed all the files for a project, convert the project into an executable file. You can also create other type of executable files, such as .ocx, .dll, etc.

In this unit, you will study about the Forms and controls, writing the code, saving the project, running and testing the project, making EXE file and printouts.

NOTES

3.1 OBJECTIVES

After going through this unit, you will be able to:

- Know about how to opening a Visual Basic Project
 - Discuss about the opening an existing program
 - Understand the introduction of saving a Visual Basic Project
 - Explain about the running a Visual Basic Project
 - Able to making the EXE file of a Visual Basic Project
 - Elaborate on the printing of the project
-

3.2 FORMS AND CONTROLS

To open a Visual Basic project, invoke Visual Basic 6.0 (VB6) by either double-clicking on the shortcut path or by going through the pull-up menu from Start. To invoke VB by going through the pull-up menu, select Start ->Programs->Microsoft Visual Studio 6.0->Microsoft Visual Basic 6.0. After a while, it will show the New Project dialog box which contains three tabs. The current tab is 'New'. The other two tabs are 'Existing' and 'Recent'. Now double-click the project that you want to open and this will open the project in the Code window. The Existing and recent tab pages show the existing project and recently created project respectively. And the New tab page contains various icons. The commonly used icons are:

Standard: This project type must be chosen if you wish to develop a small or large standalone application.

ActiveX EXE: Choose this option if you wish to create an executable component. An ActiveX executable component can be executed from other applications also. This is a program that provides functionality to a number of other applications.

ActiveX Control: This helps create a custom ActiveX control that can be used in other applications. These are like the third party controls that you buy from other software vendors.

ActiveX DLL: Like the ActiveX EXE, it provides added functionality to your application, but will work 'In-process' with your application.

Data Project: Choose this option to create a project with the database components.

IIS: This helps create an Internet application.

ActiveX Document: Creates a component that can take over the application at runtime. It creates an Internet application that can be executed from a browser.

DHTMLApplication: Creates an application that can be executed from a web browser only.

After creating a project, you need to save it. For this, click the File menu in the VB environment and select the Save project option. You will then be asked to provide a name for the Form File. This is the name of the file. This should not be confused with the form caption. Give a name for the form. Make sure that you know in which directory you are going to save it. Then you will be asked to give a name for the project. Give an appropriate name.

Note: Each time you save a project, Visual Basic updates the project file (.vbp). A project file contains the same list of files that appears in the Project Explorer window as well as references to the ActiveX controls and objects that are used in the project.

Various options available to save a VB project are:

- **Save Project** Updates the project file of the current project and all of its form, standard and class modules.
- **Save Project as** Updates the project file of the current project by saving the project file under a file name that you specify. Visual Basic also prompts you to save any forms or modules that have changed.

The following point will discuss the life cycle of a Visual Basic project.

3.2.1 Creating and Saving a New Program

To create and some a new proram follow the given steps:

- (i) Start VB from the Windows start menu.

The large VB window appears with a New Project dialog box (as shown in Figure 3.1). (In case there is no dialog box, bring it up through **File Æ New Project**).

Select Standard EXE. The VB window will show Project1 with an empty Form1 (as shown in Figure 3.2).

- (ii) Name the project.

Select Project Æ Project1 Properties. In the Project Properties dialog box, change Project1 to your project name, say Sample.

The TitleBar on the VB window changes to match.

- (iii) Name the form and set the form caption.

The Properties window at the right side of the VB window shows the properties for the form. The first property at the top of the list is Name. Change the name to, say frmsample1.

NOTES

The title bar in the Form window changes to match as shown in Figure 3.1.

NOTES

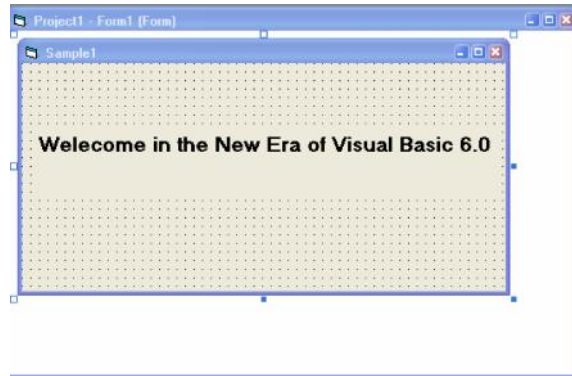


Fig. 3.1 Showing Change in the Title Bar

Scroll down to Caption still remaining in the Properties window. Rename the caption with the same name as the project, say Sample.

The title bar on the form will change automatically to match.

- (iv) Save the form (you should always name and save the form *before* you save the project).

Select File → Save frmSample1 As The **Save File As** and the **Save Project As** dialog boxes appear (as shown in Figures 3.2 and 3.3 respectively). The **Save in:** TextBox shows the folder's name where the form will be saved. The form file name `frmSample1.frm` should appear in the **File name:** box (if not, type in the correct name). Click on **Save**.

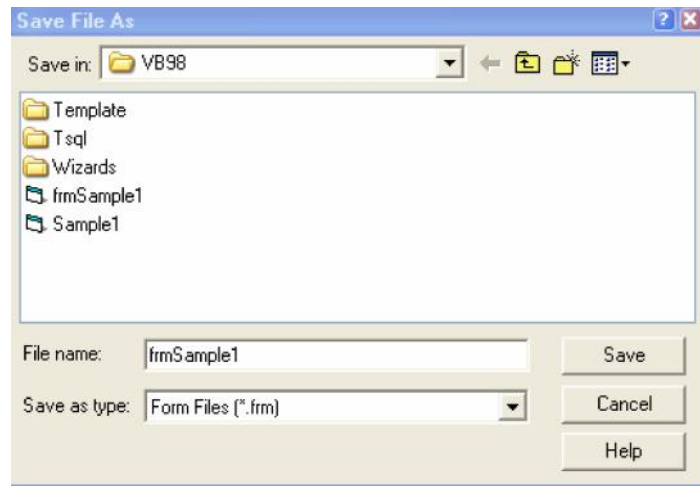


Fig. 3.2 Save File as Dialog Box (to save form file as `frmSample1.frm`)

The form is saved.

- (v) Save the project (be sure to name and save the form *before* you save the project).

Select File → Save Project As The **Save Project As** dialog box

appears. The Save in: Textbox should show the correct folder (the one you just created for the form). The File name: box should show the correct name (Sample.vbp in this example). Click on Save.

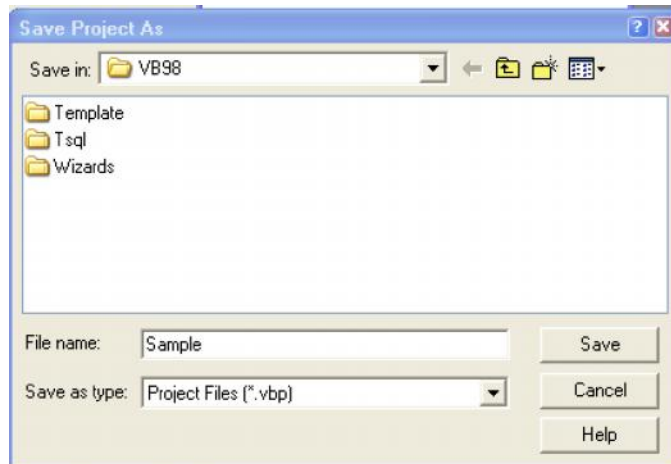


Fig. 3.3 Save Project as Dialog Box (to save project file as Sample.vbp)

The project is saved. After naming and saving the form and the project, it is better to exit and then restart Visual Basic (VB) for making sure that the project can be restored.

3.2.2 Restoring/Opening an Existing Program

To restore and open the existing program, follow the given steps:

- (i) Start VB.

Find Visual Basic on the Windows Start menu and start it.

The large VB window appears with a New Project dialog box. Choose the Existing tab (as shown in Figure 3.4). (If there is no dialog box, bring it up by File → Open Project).



Fig. 3.4 New Project Dialog Box (to choose the Existing Project)

NOTES

NOTES

Navigate to the folder that was created earlier. There will be a .vbp file for the project created earlier (Sample.vbp in this example). Select that file.

Alternatively, for navigating to the project folder, you can use My Computer or Windows Explorer instead of the Start menu. The VB starts and opens that project when the .vbp file or the .frm file is double clicked.

The restored project appears in the VB window. The project name appears in the title bar.

(ii) Open the form.

An icon for a folder of forms appears in the Project window on the right side of the VB window. Open this folder and select the form you saved. Click on the View Code and View Object icons for displaying the code or form layout windows.

Now you are ready to add controls and code to the form. At any time, you can use **File** → **Save frmSample1** and **File** → **Save Project** to save your changes to the form and the project without going through the Save As ... dialogs.

3.2.3 Running the Program

There are various ways of running your program:

- Press the F5 Key.
- On the VB Menu Bar Click Run → Start
- On the VB Toolbar, Click the VB Run Icon (the Arrow)

The program window appears and looks similar to the form that was designed. The window controls are active. The program behaves like any other window on the desktop while it is running — you can minimize it, move it, etc.

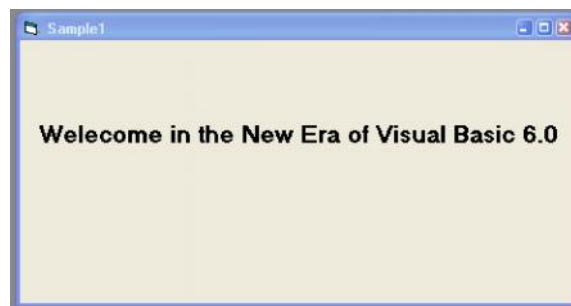


Fig. 3.5 Output of Sample Program

From the design mode the indicator on the VB window title bar changes to the run mode. Several items in the VB window disappear during the run mode and several menu and toolbar operations are not enabled.

3.2.4 Stopping the Program

There are various ways of stopping your program:

- In the window of your program, click the Exit button or menu (if you are provided one).
- In the title bar of your program window, click the Close button (VB always provides one).
- On the VB menu bar, Click Run → Stop.
- On the VB toolbar, click the VB Stop icon (the box).

From the `run` mode, the indicator on the VB window title bar changes to the `design` mode. Items in the VB window reappear, and menu and toolbar operations are enabled again.

3.2.5 Examining the Program in Break Mode

As the program runs, there are several ways of putting VB in the break mode:

- On the VB menu bar, click Run → Break
- On the VB toolbar, click the Break icon (the vertical bars)
- Your program reaches a breakpoint or experiences an error

3.2.6 Printing Visual Basic Project

There is no printer control for printing a Visual Basic project. Unlike most of the functions in VB, sending output to the printer can be a tiresome process. It requires sending a long list of instructions to the printer, describing exactly the way the output should look like.

Despite the problems sometimes faced with printing, you can easily learn to control all the aspects of printing that include the font of individual characters that is sent by the application to the printer. The control needed for printing provides accuracy and it lets you control all the printing details.

3.2.7 Exiting Visual Basic

Following are the ways to exit VB:

- On the menu bar, click File → Exit
- On the VB window title bar, click the Close button

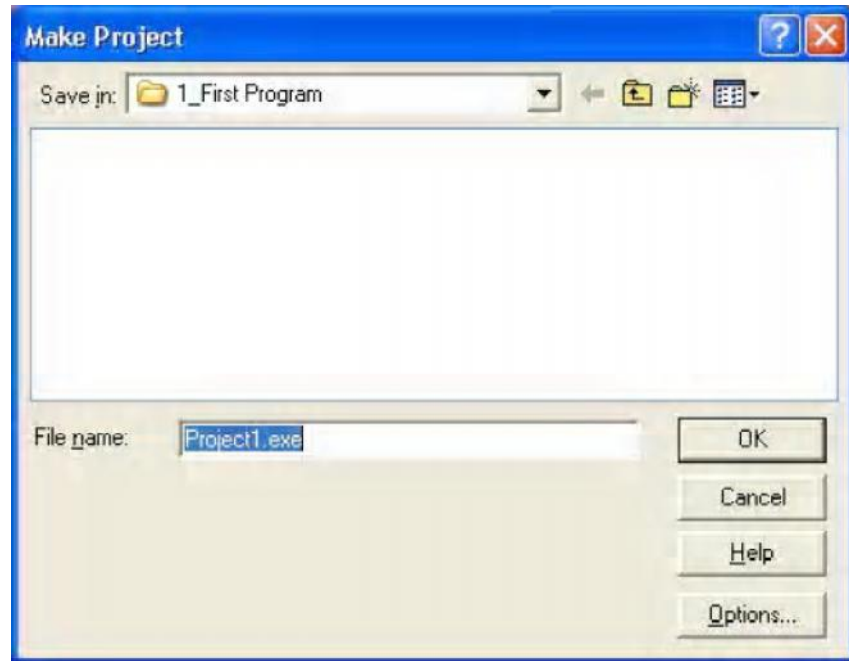
VB will give you a warning if you have forgotten to save your work.

3.3 MAKING EXE FILES

Once you prepare a Project 'Project1' in Visual Basic you can make an executable 'EXE' file to it. For this you need to select **File→Make Project1.exe** which is saved in '1_First Program' folder as shown in the following screen:

NOTES

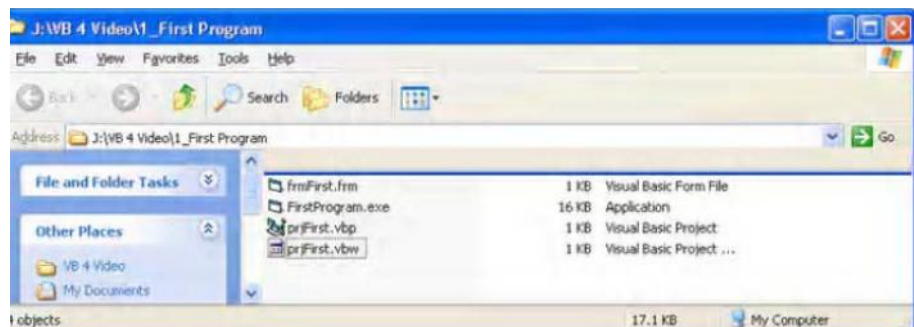
NOTES



Then, you need to save the executable in the same folder by the same name as 'Project1.exe'. The EXE version in Visual Basic is the version that should be tested for documentation. The Visual Basic development environment sometimes called the desktop prevents or protects from some errors and your user will be running the EXE version. Following steps are required to save all the files:

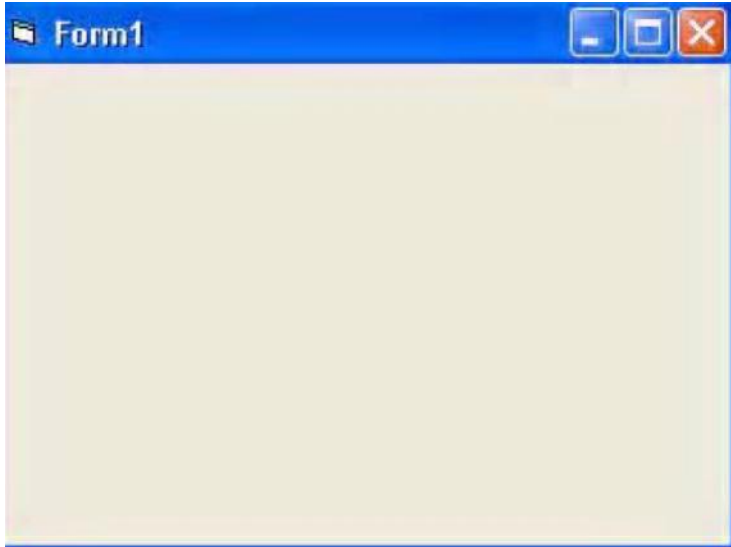
- The form name should reflect what it does in the project as frmFirst prefix the form name with frm (short for form).
- The project name should reflect the task of the project named as prjFirstProgram.
- The name for the executable should be meaningful to the user.

Run a Visual Basic program using the executable (EXE).



You can find frmFirst.frm (Visual Basic Form File), FirstProgram.exe (application), prjFirst.vbp (Visual Basic Project) and prjFirst.vbw (Visual Basic Project) as shown in the above screen containing file size (in KB). To run the executable version double click on FirstProgram.exe then check that the Visual Basic

development environment sometimes called the desktop is closed and yet the program works. Following screen shows that 'Form1' form is prepared and ready for testing method:



NOTES

Following characteristics appear in the 'Form1' form:

- The Window can be minimized.
- The Window can be maximized.
- The Window can be moved about the screen.
- The Window can be resized by pulling the borders.
- The Window can be closed.

Check Your Progress

1. Write down the various options to save a VB project.
2. State the various methods to execute a VB program.
3. Which short cut key is used for executing the Visual Basic project?
4. List down the ways to run VB program in break mode.
5. Explain about the printing of Visual Basic project.
6. What are the procedures for exiting from the Visual Basic (VB)?
7. Define the steps to save EXE file of VB project.

3.4 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

NOTES

1. Various options to save a VB project are:
 - Select the Start option from the Run menu on the Visual Basic design window.
 - Press the F5 key on the keyboard.
 - Click the Start button on the Visual Basic standard toolbar.
2. Various methods to execute a Visual Basic Program are:
 - A. Selecting the Start option from the Run menu.
 - B. Pressing the F5 key on the keyboard.
 - C. Selecting the Start icon from the Standard toolbar on the Visual Basic designer window.
3. F5 is used for executing the Visual Basic project.
4. As the program runs, there are several ways of putting VB in the break mode:
 - On the VB menu bar, click Run ® Break
 - On the VB toolbar, click the Break icon (the vertical bars)
 - Your program reaches a breakpoint or experiences an error
5. There is no printer control for printing a Visual Basic project. Unlike most of the functions in VB, sending output to the printer can be a tire some process. It requires sending a long list of instructions to the printer, describing exactly the way the output should look like.
6. Following are the ways to exit VB:
 - On the menu bar, click File ® Exit
 - On the VB window title bar, click the Close button
7. Following steps are required to save all the files of VB project:
 - The form name should reflect what it does in the project as frmFirst prefix the form name with frm (short for form).
 - The project name should reflect the task of the project named as prjFirstProgram.
 - The name for the executable should be meaningful to the user.

3.5 SUMMARY

- Standard project type must be chosen if you wish to develop a small or large standalone application.

- Choose ActiveX EXE option if you wish to create an executable component. An ActiveX executable component can be executed from other applications also. This is a program that provides functionality to a number of other applications.
- ActiveX Control helps create a custom ActiveX control that can be used in other applications. These are like the third party controls that you buy from other software vendors.
- Like the ActiveX EXE, ActiveX DLL provides added functionality to your application, but will work 'in-process' with your application.
- Choose data project option to create a project with the database components.
- IIS helps create an Internet application.
- ActiveX Document Creates a component that can take over the application at runtime. It creates an Internet application that can be executed from a browser.
- DHTML Application Creates an application that can be executed from a web browser only.
- The Properties window at the right side of the VB window shows the properties for the form. The first property at the top of the list is Name. Change the name to, say frmsample1.
- Scroll down to Caption still remaining in the Properties window. Rename the caption with the same name as the project, say Sample. The title bar on the form will change automatically to match.
- Save the form (you should always name and save the form before you save the project).
- The project is saved. After naming and saving the form and the project, it is better to exit and then restart VB for making sure that the project can be restored.
- Alternatively, for navigating to the project folder, you can use My Computer or Windows Explorer instead of the Start menu. The VB starts and opens that project when the .vbp file or the .frm file is double clicked. The restored project appears in the VB window. The project name appears in the title bar.
- The program window appears and looks similar to the form that was designed. The window controls are active. The program behaves like any other window on the desktop while it is running-you can minimize it, move it, etc.
- From the design mode the indicator on the VB window title bar changes to the run mode.

NOTES

NOTES

- Several items in the VB window disappear during the run mode and several menu and toolbar operations are not enabled.
- In the window of your program, click the Exit button or menu (if you are provided one).
- In the title bar of your program window, click the Close button (VB always provides one). On the VB menu bar, Click Run? Stop. On the VB toolbar, click the Stop icon (the box).
- From the run mode, the indicator on the VB window title bar changes to the design mode. Items in the VB window reappear, and menu and toolbar operations are enabled again.
- As the program runs, there are several ways of putting VB in the break mode: On the VB menu bar, click Run Break. On the VB toolbar, click the Break icon (the vertical bars). Your program reaches a breakpoint or experiences an error.
- There is no printer control for printing a Visual Basic project. Unlike most of the functions in VB, sending output to the printer can be a tiresome process.
- It requires sending a long list of instructions to the printer, describing exactly the way the output should look like.
- Despite the problems sometimes faced with printing, you can easily learn to control all the aspects of printing that include the font of individual characters that is sent by the application to the printer.
- The control needed for printing provides accuracy and it lets you control all the printing details.
- The ways to exit VB: On the menu bar, click File Exit. On the VB window title bar, click the Close button. VB will give you a warning if you have forgotten to save your work.
- The form name should reflect what it does in the project as frmFirst prefix the form name with frm (short for form).
- The project name should reflect the task of the project named as prjFirstProgram.
- The name for the executable should be meaningful to the user.

3.6 KEY WORDS

- **Project:** project is a collection of file which are used file, that one can use to build and application.
- **Program:** A computer program is a collection of data and instructions that can be executed by a computer to perform a specific task.

- **Visual Basic:** Visual Basic is a third-generation event-driven programming language from Microsoft known for its Component Object Model (COM) programming model first released in 1991 and declared legacy during 2008.
- **Break mode:** Break mode is the state of an application when the execution gets paused and allows the developer to edit the value in the current state
- **File:** A file is a collection of data stored in one unit, identified by a filename. It can be a document, picture, audio or video stream, data library, application, or other collection of data.
- **Standard:** This project type must be chosen if you wish to develop a small or large standalone application.
- **ActiveX EXE:** Choose this option if you wish to create an executable component. An ActiveX executable component can be executed from other applications also. This is a program that provides functionality to a number of other applications.
- **ActiveX Control:** This helps create a custom ActiveX control that can be used in other applications. These are like the third party controls that you buy from other software vendors.
- **ActiveX DLL:** Like the ActiveX EXE, it provides added functionality to your application, but will work 'in-process' with your application.
- **IIS:** This helps to create an Internet application.
- **DHTML application:** Creates an application that can be executed from a web browser only.

NOTES

3.7 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. Write the various procedures for running a program.
2. State the different ways for running the Visual Basic program.
3. What are the various ways of stopping VB program?
4. Explain about the printing Visual Basic project.
5. Elucidate on the different ways to save EXE file.

Long-Answer Questions

1. Briefly discuss the how to create and save the new program for Visual Basic?
2. Discuss about the restoring/opening and existing program for Visual Basic.
3. Describe the various procedures of running a program giving appropriate examples?

NOTES

4. Write the short notes on :
 - a) Stopping the Program
 - b) Examining the Program in Break Mode
 - c) Printing Visual Basic Project
 - d) Exiting Visual Basic

3.8 FURTHER READINGS

- Cornell, Gary. 1998. *Visual Basic 6 from the Ground Up*. New Delhi: Tata McGraw-Hill.
- Warner, Scott L. 1998. *Teach Yourself Visual Basic 6*. New Delhi: Tata McGraw-Hill.
- Jerke, Noel. 1999. *Visual Basic 6 - The Complete Reference*. New York: McGraw-Hill.
- Smith, Eric A., Valor Whisler and Hank Marquis. 1998. *Visual Basic 6 Programming Bible*. New Jersey: John Wiley & Sons, Inc.
- Azam, M. 2001. *Programming with Visual Basic*. New Delhi: Vikas Publishing House Pvt. Ltd.
- Manchanda, Mahesh. 2009. *Visual Programming*. New Delhi: Vikas Publishing House Pvt. Ltd.
- Balena, Francesco. 1999. *Programming Microsoft Visual Basic 6.0*. Bangalore: WP Publishers and Distributors (P) Ltd.
- Petroutsos, Evangelos. 1998. *Mastering Visual Basic 6, 1st Edition*. New Delhi: BPB Publications.
- Deitel, Harvey M., Paul J. Deitel and T. Tem R. Nieto. 1999. *Visual Basic 6: How to Program*. New Jersey: Prentice-Hall.
- Norton, Peter. 1998. *Peter Norton's Complete Guide to Visual Basic 6*. New Delhi: Techmedia.
- Reselman, Bob and Richard A. Peasley. 1998. *Using Visual Basic 6*. New Jersey: Pearson Education (Que Publishing).
- Donald, Bob and Oancea Gabriel. 1999. *Visual Basic 6 from Scratch*. New Delhi: Prentice-Hall of India.

BLOCK II

VISUAL BASIC CODE, EVENTS AND CONTROLS

*Adding Code
and Using Events*

UNIT 4 ADDING CODE AND USING EVENTS

NOTES

- 4.0 Introduction
- 4.1 Objectives
- 4.2 Data Types
- 4.3 Declaring and Using Variables
- 4.4 Introducing Operators
 - 4.4.1 Arithmetic Operators
 - 4.4.2 Relational Operators
 - 4.4.3 Concatenation Operators
 - 4.4.4 Logical Operators
- 4.5 Answers to Check Your Progress Questions
- 4.6 Summary
- 4.7 Key Words
- 4.8 Self-Assessment Questions and Exercises
- 4.9 Further Readings

4.0 INTRODUCTION

Data types, operators and functions are the tools to learn the basics of coding of any language. In Visual Basic (VB) code basics, the various data types are: Boolean, byte, double, integer, long, single and string. Constants or literals are values given to a variable. In fact, a constant is a value that does not change. A constant whose data type is not specified is considered to be of variant type. A variable is a named storage location whose contents can be varied, or it is simply a name you give for a memory area in which the value of data, which was utilized by your program, is contained. As the name suggests, it is a location whose contents can be varied. Like other programming languages, Visual Basic (VB) also supports variables. A variable has two things associated with it: a name and its data type.

The variable name is used to refer to the value stored in it and the data type tells what type of value can be stored in the variable.

Operators are special symbols or characters used to describe an operation or an action that is to take place between two or more values. Operators are available in Visual Basic (VB) for performing arithmetic, comparison and logical operations.

Programming languages generally support a set of operators that are similar to operators in mathematics. A language may contain a fixed number of built-in operators or it may allow the creation of programmer-defined operators. Operators

NOTES

perform operations on one or more operands. They are used to manipulate primitive data types.

An expression in a programming language refers to a combination of explicit values, constants, variables, operators and functions that are interpreted according to the rules of precedence and association followed by the language. When an expression is computed, it produces a value.

In this unit, you will study about the literals data types, declaration of variables, operator subroutine and functions.

4.1 OBJECTIVES

After going through this unit, you will be able to:

- Discuss about the constants, their types and scope
- Introducing and declaring variables to store values during computations
- Explain the data types, such as integer and strings
- Declaring data types as per programming syntax in an application
- Describe variables and data types
- Understand the method of declaring a data type
- Use various operators in Visual Basic

4.2 DATA TYPES

Every programming element, such as variable, literal, constant, procedure etc., has a data type. Visual Basic (VB) have a name and declared data type. The data type of a programming element determines how the bits/bytes representing these values are stored in the computer memory. When you declare a variable, you also declare a data type for it. All variables have a data type that determines the kind of data that they can store.

For example, in an invoice program, the variable, `ItemName` will probably hold values like bread, margarine, coke, toothpaste, etc., that are all purely textual in nature. Besides, you will not perform any arithmetic operations on these values. On the other hand, the `ItemRate` variable holds numeric values which will be used for arithmetic and other operations.

Therefore, you have at least two types of variables. Visual Basic has a number of variable types to deal with various programming requirements. One needs to use different types of variables for different requirements in order to optimize speed and memory requirements.

Data types can be applied to other elements also besides variables. When you assign a value to a property, that value has a data type. In fact, just about anything in Visual Basic that involves data also includes data types. Table 4.1 lists various data types and their description.

Table 4.1 Data Types and their Storage Capacity

Data Type	Range
Integer	A numeric variable holds numeric values from -32,768 to 32,767.
Long Integer	A numeric variable holds a wider range of integers than Integer -2,147,483,648 to 2,147,483,647.
Single	A numeric variable, which holds numbers with decimal places. -3.402823E38 to -1.401298E-45–For negative values, 1.401298E-45 to 3.402823E38–For positive values.
Decimal	A decimal number is a floating point value that consists of a sign, a numeric value (from 0 to 9), and a scaling factor that indicates the position of a floating decimal point that separates the integral and fractional parts of the numeric value. 0 through +/- (+/-7.9...E+28) with no decimal point; 0 through +/- 7.9228162514264337593543950335 with 28 places to the right of the decimal.
Double	A numeric variable with a wider range 1.79769313486232E308 to -4.940656458412 47E-324 for negative values; 4.94065645841247E-324 to 1.79769313486232E308.
Currency	For holding monetary values. -922,337,203,685,477.5808 to 922,337,203,685,477.5807.
String	For holding text or string values. 0 to approximately 2 billion for variable length. 1 to approximately 65,400 for fixed length.
Byte	A numeric variable, holding less than the value 255, 0 to 255.
Boolean	For holding True or False values.
Date	For holding Date values inclusive of and between January 1, 100 to December 31, 9999.
Object	For holding references to objects or pictures in Visual Basic and other applications.
User defined	Number required by elements. The range of each element is the same as the range of its data type.
Variant	A general purpose variable that can hold most other types of variables values (with number). Any numeric value up to the range of a Double. With Character values, it has the same range as for a variable length String.

During the data type declaration, you declare a variable name and its data type as discussed in the above section. You can declare a variable and its data type in the following two ways:

- Using explicit declaration.
- Using implicit declaration.

NOTES

NOTES

Using Explicit Declaration

In explicit declaration, you use the in-built Visual Basic statement, `Dim` statement, to declare a variable name and its data type. The `Dim` statement does not allow assigning values to the declared variable. Various data types that can be declared in Visual Basic using `Dim` statement are as follows:

- **Integer:** This data type is used to store whole numbers and cannot be used in calculations where decimals or fractions are involved. They can store reasonably large numbers. The `Integer` data type occupies only two bytes of memory and is quite fast when used in calculations. Declaration for `Integer` is:

```
Dim a As Integer
```

The above declaration defines `a` as variable of the `Integer` data type.

- **Long Integer:** It must be used only where the calculations involve very large numbers. Declaration for `Long Integer` is:

```
Dim x as Long
```

The above declaration defines `x` as variable of the `Long integer` data type.

- **Single:** This is the equivalent of the floating point number. It can store fractions and provide precision to a fairly high level. It occupies 4 bytes of memory space and should be used where very high precision is not required. Declaration for `Single` is:

```
Dim x As Single
```

The above declaration defines `x` as variable of the `Single` data type.

- **Double:** This solves the problem of precision that the `Single` data type lacks. It occupies 8 bytes of memory space and should be used in applications where the requirement of precision is very high. This is not advisable for regular commercial applications as it can be fairly slow compared to the `Integer` data type and should be used when you want accuracy in calculations involving figures beyond the fourth decimal point. Declaration for `Double` is:

```
Dim x as Double
```

The above declaration defines `x` as variable of the `Double` data type.

- **Currency:** This data type is used for holding values related to item rates, payroll details and other financial functions. However, this data type should not be used if you need extreme accuracy beyond the fourth decimal point. For example, if you are working on Foreign Exchange details or interest rates for very large values. Declaration for `Currency` is:

```
Dim salary As Currency
```


NOTES

The above declaration defines `salary` variable of the `Currency` data type.

- **Byte:** This data type can hold values from 0 to 255. It cannot hold negative numbers or numbers larger than 255. Assigning negative values or values beyond 255 will result in a runtime overflow error. The `Byte` data type occupies only one byte of memory. Declaration for `Byte` is:

```
Dim salary As Byte
```

The above declaration defines `salary` as variable of the `Byte` data type.

- **Boolean:** This data type accepts only `True` or `False` values. Since the default value for all numeric data types is zero, the default value for a `Boolean` data type is also zero. The zero value is interpreted as `False` and a non-zero value is interpreted as `True`. The VB keywords, `True` and `False` can be used to assign values to the `Boolean` data type. Declaration for `Boolean` is:

```
Dim b As Boolean
```

The above declaration defines `b` as variable of the `Boolean` data type.

- **Date:** This variable holds date and time data. It can hold time from January 1 100 to December 31 9999 and time from 00.00.00 (midnight) to 23.59.59 (one second before midnight) in one second increments. It occupies 8 bytes of memory. The data is displayed as per the settings in your computer. You can store it in British format, American format or any other format that is available in the Regional Settings in your control panel.

When other numeric data types are converted to `Date`, values to the left of the decimal represent date information, while values to the right of the decimal represent time. Midnight is 0 and midday is 0.5. Negative whole numbers represent dates before December 30, 1899. Declaration for `Date` is:

```
Dim Tomorrow As Date
```

The above declaration defines `Tomorrow` as variable of the `Date` data type.

- **String:** This is the most commonly used data type. Declare a variable with this data type if it will always contain a string value and not a numeric value. You can declare the variable as a 'variable length string or a fixed length string'. By default, a string variable or argument is a variable length string; the string grows or shrinks as you assign new data to it.

To declare a fixed length string, you need to specify a fixed length string with using the following syntax:

NOTES

Declaration for String is:

```
Dim ItemName As String * 30 ' a fixed length string
```

```
Dim ItemName As String ' a variable length string
```

The above declaration defines `ItemName` as variable of the `String` data type. The string `ItemName` will always be 30 characters long. If you assign a string of fewer than 30 characters, `ItemName` will be padded with enough trailing spaces to total 30 characters. If you assign a string that is too long for the fixed length string, Visual Basic simply truncates the characters.

In order to remove trailing spaces while working with fixed length strings, you can use the functions like `Trim` and `Rtrim`.

- **Object:** In Visual Basic, forms, controls, procedures and recordsets are all considered as objects. All programming activity revolves around these objects. Since VB is very much an object based programming language, it is very natural to use `Object` data types.

An `Object` variable refers to an object within the application or in some other application. This object can be a `TextBox`, a `Form` or a database.

A variable declared as an `Object` is one that can be subsequently assigned by using the set statement to refer to any actual object recognized by the application. Declaration for `Object` is:

```
Dim obj As Object
```

The above declaration defines `obj` as variable of the `Object` data type.

- **Variant:** A `Variant` data type is a variable that can freely change its type. It can accept text, number or byte data easily. If you do not supply a data type, the variable is given the `Variant` data type by default. Declaration for `Variant` data type is:

```
Dim VarValue
```

```
'Variant by default'
```

The above declaration defines `VarValue` as variable of the `Variant` data type.

The `Variant` data type comes in handy when you are not sure about the data value. For example, if you are reading a file and you are not sure if the data is numeric or text or if there is a `Null` value, then you can use this data type.

However, the ease with which the conversion takes place can make you careless. If you are performing calculations, make sure that the value contained in the `Variant` is a number.

NOTES

The program will throw an error if you attempt to perform a mathematical operation or function on a `Variant` that does not contain a number. For example, you cannot perform any arithmetic operations on the value `30+` even though it contains a numeric character, but the entire value is not a valid number. It is a good idea to determine if a `Variant` variable contains a value that can be used as a number.

If you are performing concatenation then you must make sure that the values in the variants are strings. It is better to use the “&” operator rather than the “+” operator.

If both the variants contain numbers, the `+` operator performs addition of these numbers. If both the variants contain strings, then the `+` operator performs string concatenation. However, if one of the values is numeric and the other a string, then you have a problem. Visual Basic first attempts to convert the string into a number. If the conversion is successful, the `+` operator adds two values; if unsuccessful, it generates a `Type mismatch` error.

The fact that `Variant` data type contains values that other variables cannot handle makes it special. These values are:

- `Null`
- `Empty`
- `Error`

The `Null` Value

`Null` is commonly used in database applications to indicate unknown or missing data. If you assign `Null` to a variable of any type other than `Variant`, an error occurs. Assigning `Null` to a `Variant` variable does not cause an error.

You can also assign `Null` with the `Null` keyword as follows:

```
Z = Null
```

You can use the `IsNull` function to test if a `Variant` variable contains `Null`:

```
If IsNull(X) Then  
    Debug.print .....  
End If
```

You can return `Null` from any function procedure with a `Variant` return value. Variables are not set to `Null` unless you explicitly assign `Null` value to them.

The `Error` Value

In a `Variant`, `Error` is a special value used to indicate that an error condition has occurred in a procedure. An `Error` value is created by converting a real number by using the `CVErr` function.

NOTES

The Empty Value

A `Variant` variable has the `Empty` value before it is assigned a value. The `Empty` value is a special value different from 0, a zero length string (“”) or the `Null` value.

When a `Variant` contains the `Empty` value, you can use it in expressions, where it is treated as either 0 or a zero length string, depending on the expression.

The `Empty` value disappears as soon as any value including 0, a zero length string or `Null` is assigned to a `Variant` variable.

A `Variant` always takes up 16 bytes, regardless of the type of data stored in it. Objects, strings and arrays are not physically stored in the `Variant`. Four bytes of the `Variant` are used to hold either an object reference or a pointer to the string or array. The actual data is stored elsewhere. Based on what has been discussed above, use the `Variant` data type only when required. It takes up space, it is slow, and may also cause errors.

Using Implicit Declaration

You can also use implicit declaration to declare a variable name and its data type. Visual Basic (VB) defines a special character for each data type. You need to use that special character at the end of a variable name to declare a data type for that variable. This type of declaration is known as implicit declaration. Table 4.2 lists the various data types and their respective special character.

Table 4.2 Data Types and their Respective Special Character

Data Type	Special Character
Integer	%
Long	&
Single	!
Double	#
Currency	@
String	\$
Byte	None
Boolean	None
Date	None
Object	None
Variant	None

For example, the following statements declare different variables and their data types using implicit declarations:

`Number% = 12` declares a `Number` variable of integer data type. The statement also assigns a value 12 to the `Number` variable. `Value$ = abc` declares a `Value` variable of `String` data type. The statement also assigns a string `abc` to the `Value` variable.

Most of the data types could hold either string or numeric data, and the numeric types could hold either integer or floating point values. The **Variant** type is different altogether. A type **Variant** variable can hold just about any kind of data string, integer or floating point. Even more amazing (at least to experienced programmers who are used to the standard fixed data types), the data in a type **Variant** is automatically treated in the appropriate way.

To create a variable or an array of type **Variant**, you can use the **Variant** keyword in your **Dim** statement or you can simply omit the **As** part of the **Dim** statement, because **Variant** is Visual Basic default data type. The following two statements are equivalent:

```
Dim X As Variant
Dim X
```

User Defined Data Types

One of the handiest features of VB is the ability to create user defined data types. A user defined type (also called a *structure*) is a compound data type containing two or more other data types. You can define exactly what goes into a structure, designing it around the exact needs of your program. You use the **Type...End Type** statement to define a structure.

The following rules apply to user defined data types:

- They can be declared only at the module level.
- They can have `Public` (project level) or `Private` (module level) scope. The default is `Public` if `Public` or `Private` is specified.
- User defined data types having `Public` scope can be defined only in standard modules and not in forms.

The following is the syntax for defining a user defined data types:

```
[Public | Private] Type TypeName
    Variable1 As datatype
    ...
    Variablen As datatype
End Type
```

For example, to define a user defined data types or `udt` for an employee record, you might code the following:

```
Public Type EmployeeRecord
    strEmpName As String
    dtmHireDate As Date
    sngHourlyRate As Single
End Type
```

NOTES

NOTES

However, the definition itself is not sufficient. The `Type` definition is basically a 'template' on which other variables are defined, the template itself does not store any data. For using a user defined data type or `udt`, you must define a variable 'As' the name, following the keyword 'Type' (in this case, 'EmployeeRecord'). For example:

```
Dim udtEmpRec
As EmployeeRecord
```

This defines a variable called 'udtEmpRec', which has the attributes defined by the structure 'EmployeeRecord'. Thus, you refer to 'udtEmpRec', in your procedural statements and not 'EmployeeRecord'. For referencing a single element of the structure, it should be qualified with the 'udt' variable which you have defined. For example, the following code puts data in the individual elements of `udtEmpRec`:

```
udtEmpRec.strEmpName = "JOE SMITH"
udtEmpRec.dtmHireDate = #1/15/2001#
udtEmpRec.sngHrlyRate = 25.50
```

Any number of variables can be declared 'As' the `udt` that you have defined. For example:

```
Dim udtEmpRec2 As EmployeeRecord
Dim audtEmpRec(1 To 10) As EmployeeRecord
```

Type Declaration Characters

An alternative way of specifying data types while declaring a variable is the use of some special characters in place of data types. These special characters which specify the data type are known as type declaration characters. The final character can be a '**type declaration character**'. Only some of the variable types can use them as shown in Table 4.3.

Table 4.3 Showing Type Declaration Characters

Data Type	Type Declaration Character
String	\$
Integer	%
Long	&
Single	!
Double	#
Currency	@

It is not a good idea to use type declaration characters in VB, these days the practice is to use the 'As' clause in a data declaration statement, which is described as follows.

VB is not case sensitive, i.e., **STUDENT**, **Student** and **student** all refer to the same variable. If you type a variable on a line in a case different from the case in which you have declared the variable, VB will change this case to match with that of the declared variable when you exit that line.

Standard Variable Prefixes

Table 4.4 shows the standard variable name prefixes for data types in variable names.

Table 4.4 Showing Standard Variable Prefixes

Variable Type	Recommended Prefix	Sample Variable Name
Boolean	bln	blnDataIsValid
Byte	byt	bytSmallNum
Currency	cur	curBigMoney
Date	dtm (for 'Date/Time')	dtmEmpBirthDate
Double	dbl	dblAnnSalary
Integer	int	intCounter
Long	lng	lngBigIntValue
Object	obj	objExcelApp
Single	sng	sngHrlyRate
String	str	strMyWord
Variant	vnt (or 'var')	vntWhatever

- It should be noted that the style of using a lower case, three character prefix is succeeded by a descriptive name in mixed case. This is the way of naming variables in all Visual Basic (VB) documentations. The scope of the variable should be indicated with an additional prefix, which is of one character, preceding the name of the variable.

Variable Default Values

When you create a variable, Visual Basic (VB) automatically assign a value to it. This value is known as **default value of the variable**. The default value of a variable depends on the data type variable. Default values for different data types are given in Table 4.5.

Table 4.5 Showing Variable Default Values

Data Type	Default Value
Single	0
Byte	0
Integer	0
Long	0
Double	0
String	"" blank
Boolean	False
Variant	Empty
Date	0
Currency	0

4.3 DECLARING AND USING VARIABLES

A variable is a named storage location whose contents can be varied or simply a name you give for a memory area in which the value of data which was utilized by

NOTES

your program is contained. As the name suggests, it is a location whose contents can be varied. Like other programming language, VB also supports variables. A variable has two associated things with it, i.e., a name and its **data type**.

NOTES

The variable name is used to refer to the value stored in it and the **data type** tells what type of value can be stored in the variable.

Table 4.6 lists the types of VB variables and the corresponding range of values that they can store.

Table 4.6 Different Data Types with their Description and Range

Type	Stores	Memory	Range of Values
Integer	Whole numbers	2 bytes	–32,768 to 32,767.
Long	Whole numbers	4 bytes	Approximately +/- 2.1E9, stores approx. +/- 2 billion.
Single	Decimal numbers	4 bytes	–3.402823E38 to –1.401298E–45 for negative values and 1.401298E–45 to 3.402823E38 for positive values.
Double	Decimal numbers (doubleprecision floating point)	8 bytes	–1.79769313486232E308 to –4.94065645841247E–324 for negative values and 4.94065645841247E–324 to 1.79769313486232E308 for positive values (–1.79E+308 to 1.79E+308).
Currency	Numbers with up to 15 digits left of the	8 bytes	922,337,203,685,477.5808 to 922,337,203,685,477.5807 Or stores approx. +/- 922 trillion.
String	Text information	1 byte per character	Up to 65,000 characters for fixed length strings and up to 2 billion characters for dynamic strings.
Byte	Whole numbers	1 byte	0 to 255.
Boolean	Logical values	2 bytes	True or False.
Date	Date and time information	8 bytes	1 January 100 to 31 December 9999. Additional Info: Dates are stored internally as eight-byte double precision floating point variables, which can represent dates from 1 January 100 A.D. up to 31 December 9999. The integer portion holds the date with zero being 31 December 1899. Dates prior to this are stored as negative values; those after as positive. 12 February 1997, for example, is stored as 35,473. The fraction portion holds time information. Midday, for example, is represented as 0.5.
Object	Pictures and any object reference	4 bytes	A 32-bit (4 bytes) address that refers to the location of an object.
Variant	Any of the preceding data types	16 bytes + 1 byte per character	Can hold any native data type, object reference, and the special values Error, Empty and Null. Numeric variants require 16 extra bytes, string variants require an extra 22.

Variable Naming Conventions

The rules for forming a valid VB variable name are as follows:

- Any of the English alphabets A to Z must function as the first character. Both upper case or lower case letters may be used. The remaining characters can be letters, digits or the underscore (_) character.
- As many as 255 characters can be present in a name.
- A space or a period (.) or a hyphen (-) cannot be present in a variable name.
- A variable name must begin with a letter.
- A reserved word (VB keyword) cannot constitute the name.

NOTES

Declaring a Variable

Declaring a variable means telling the program about it in advance. A variable can be declared according to the following syntax:

Syntax

```
Dim <Variable Name> [As <Datatype>]
```

Where,

- The keyword **Dim** indicates VB that a variable is being declared.
- <Variable Name> is the variable name provided by the user.
- **As** is another keyword that tells VB the data types of the variable.
- <Data type> is a legal data type in VB as given in Table 1.16.

For example,

```
Dim marks As Byte
```

```
Dim address As String
```

```
Dim rate As Double
```

If the 'As' data type clause is neglected during declaration of a variable, the variable type defaults to `Variant` unless a type declaration character is used. For example, the following two statements both declare an `Integer` called 'Counter' (the 'As' version is preferred):

```
Dim Counter As Integer
```

```
Dim Counter%
```

The following two statements declare a `Variant` variable called 'Whatever':

```
Dim Whatever As Variant
```

```
Dim Whatever
```

NOTES

Static Variables

You have learned that local or private variables have lifetimes equal to the runtime of their procedure. But there is one exception here. If you declare a local variable with keyword **Static** given as follows:

```
Static <Variable Name> As < Data Type>
```

The lifetime of the variable changes so that the variable lives on in the memory even after its parent procedure is over.

Suppose you have a Sub in a form called CountDemo that looks like this:

```
Private Sub CountDemo()  
Dim    intI As Integer  
Static intJ As integer  
    intI = intI + 1  
    intJ = intJ + 1  
    Print intI , intJ  
End Sub
```

Suppose you have some other Sub that calls CountDemo three times in a row:

```
Call CountThem  
Call CountThem  
Call CountThem
```

The following output would be displayed on the Form :

```
values for intI values for intJ  
1             1  
1             2  
1             3
```

Note that the 'regular' variable, intI, declared with 'Dim' does not retain its value between calls, whereas the Static variable, intJ does.

Note: The keyword 'Static' can also be used in the Sub procedure header, which causes all variables in that procedure to be static. For example:

```
Private Static Sub StaticDemo()  
    Dim Counter      As Integer  ' as if declared Static  
    Dim ErrMsg       As String   ' as if declared Static  
    . . . ' other statements  
End Sub
```

The following is the syntax for declaring a variable in VB:

```
[Dim | Private | Public | Static | Global] variablename  
[As data type]
```

Note that you can use any one of the five keywords for declaring a variable, which one you use depends on the scope you want the variable to have.

The following are the three levels of scope:

- **Project Level Scope:** It also called 'global' or 'public' or 'application' scope; the variable is accessible to all procedures in all modules of the project. The variables available to all the modules and procedures in an **Public** keyword declares public or project level variables. Since the **Public** variable are available *publicly* or *globally* to all module of the application, these are also known **global variable**, i.e., variable with **global scope**.
- **Module Level Scope:** It the variable is accessible to all procedures in the module in which it is declared. Module is a place where you can put your commonly used routines, functions, constants, etc. In other words, a module is a place to store commonly used things. These things may be used in many projects. In VB, there are three kinds of modules namely *form* module, *standard* module and *class* module. A **form module** stores everything related to a form. A **standard module** stores the commonly used variables, constant and procedures, etc. A **class module** stores code to create new objects, the basis of object oriented programming.
- **Private or Local Level:** The variable is accessible only to the procedure in which it is declared. A variable that can be used only in one procedure in which it is declared is said to have **Private or local scope**.

In addition to the keyword used for declaring the variable, the site of the variable declaration also affects its scope. A variable can be declared in any one of the following two locations:

- **The General Declarations Section:** This section of a module is unlabelled, i.e., it is always present at the start of a code module, after the '*Option Explicit*' statement but preceding the first Sub or Function procedure. Declaring a variable here using the **Public** or **Global** keyword makes it a project level variable, i.e., or a module level variable if the **Private** or **Dim** keyword is used.
- **Within a Sub or a Function Procedure:** Declaring a variable here makes it a local level variable. Here, the **Dim** or **Static** keyword alone can be used. The usual practice involves declaring all local variables within a procedure immediately succeeding the Sub or the Function header and preceding any executable statements.

Table 4.7 shows how the five different declarative keywords and the location of their declaration affect the scope.

NOTES

NOTES

Table 4.7 Showing How the Location of Declaration affect Scope

Keyword Used to Declare the Variable	Where Declared	General Declarations Section of a Form (.frm) Module	General Declarations Section of a Standard (.bas) Module	Sub or Function procedure of a Form or Standard Module
Dim (preferred keyword for local, but not module level variables)		Module level scope	Module level scope	Local level scope (value of the variable is NOT preserved between calls)
Static		Not allowed	not allowed	Local level scope (value of the variable is preserved between calls)
Private (preferred keyword for module level variables)		Module level scope	Module level scope	Not allowed
Public		Project level scope (but references to the variable must be qualified with the form name; also there are some minor restrictions on the types of variables that can be declared as Public in a form)	Project level scope	Not allowed
Global (the use of this keyword is discouraged; it remains only for compatibility with older versions of VB)		Not allowed	Project level scope	Not allowed

Check Your Progress

1. Give the definition of data types.
2. Write down the storage capacity of currency data type.
3. In how many ways you can declare a variable and its data type in Visual Basic?
4. Why we use Integer data type in Visual Basic?
5. Elaborate on the variant for explicit declaration.
6. What is Error value?
7. Define about the implicit declaration.
8. Explain the term user-defined data types.
9. What is the variable default values?
10. Explain the term variable.
11. State the three levels of scope for declaring variables.

4.4 INTRODUCING OPERATORS

In most of the Visual Basic (VB) programs, various operations such as numerical and logical operations are performed to carry out a specific task. For each operation VB specifies a specific symbol, known as operators. Depending on the functioning of the operations, the operators are categorised into four categories:

- Arithmetic operators
- Relational operators
- Concatenation operators
- Logical operators

4.4.1 Arithmetic Operators

Arithmetic operators are used to perform the numerical operations, such as adding and multiplying two or more data values. Table 4.7 lists various numerical operations, their corresponding arithmetic operators, examples and description.

Table 4.7 Arithmetic Operators

Numerical Operation	Operator	Example	Description
Addition	+	Dim x As Integer $x = 67 + 34$	Adds the two values 67 and 34 and assigns the value, which we get after the addition, to x. Therefore, the output is $x=101$
Subtraction	-	Dim x As Integer $x = 67 - 34$	Subtracts the value 34 from 67 and assigns the value, which we get after the subtraction, to x. As a result, the output is $x=33$
Multiplication	*	Dim x As Integer $x = 7 * 3$	Multiplies the two values 7 and 3 and assigns the value, which we get after the multiplication, to x. Therefore, the output is $x=21$
Division	/	Dim x As Integer $x = 9 / 3$	Divides the value 9 by 3 and assigns the quotient, which we get after the division, to x. As a result, the output is $x=3$
Modulus	Mod	Dim x As Integer $x = 7 \text{ mod } 3$	The mod operator divides the value 7 by the value 3 and returns the remainder. Therefore, the output is $x=1$
Exponentiation	^	Dim x As Integer $x = 2 ^ 3$	^ represents that 3 is the exponential power of 2. As a result, the output is $x=8$
Integer division	\	Dim x As Integer $x = 7 \backslash 3$	The integer division returns the integer quotient without considering the remainder, if any. As a result, the output for the corresponding example is 2.

NOTES

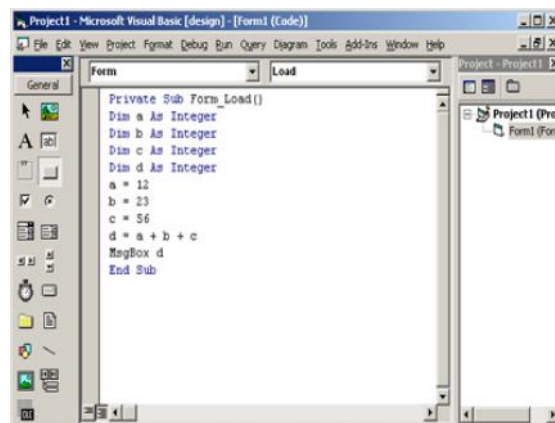
NOTES

The following steps show how to use + operator to add the three numbers, 12, 23 and 56:

1. Open the Code window.
2. Click the down arrow button next to the Object drop-down list box and select the Form control.
3. Click the down arrow button next to the Procedure drop-down list box and select the Load event.
4. Enter the following lines of code in the Code window:

```
Private Sub Form_Load()  
Dim a As Integer  
Dim b As Integer  
Dim c As Integer  
Dim d As Integer  
a = 12  
b = 23  
c = 56  
d = a+b+c  
MsgBox d  
End Sub
```

Fig. 4.1 shows the Code window for the + operator



5. Press F5 to compile and execute the code. Figure 4.2 shows the output after adding the three numbers.



Fig. 4.2 Displaying the Output of the + Operator

4.4.2 Relational Operators

Comparison operators are used to compare the two values of same data type and return a Boolean value, which represents the relation between the two values. A Boolean value has only two values, either True or False. These values may be numerical, strings and objects. Table 4.8 lists various relational operations, their corresponding comparison operators and description along with examples.

Note: In the following table, we are considering x and y as variables having values 33 and 22, respectively.

Table 4.8 Relational Operators

Compare Operation	Comparison Operator	Example	Description
Equality	=	$x = y$	Returns True if the two values are equal, else returns False. Here, the corresponding example returns False as 33 and 22 are not equal.
Inequality	\neq	$x \neq y$	Returns True if the two values are not equal, else returns False. Here, the corresponding example returns True as 33 and 22 are not equal.
Less than	<	$x < y$	Returns True if the first value (value on the left side of the <) is less than the second value (value on the right side of the <), else returns False. Here, the corresponding example returns False as 33 is not less than 22.
Greater than	>	$x > y$	Returns True if the first value is greater than the second value, else returns False. Here, the corresponding example returns True as 33 is greater than 22.
Less than or equal to	\leq	$x \leq y$	Returns True if the first value is less than or equal to the second value, else returns False. Here, the corresponding example returns False as 33 is neither equal to nor less than 22.
Greater than or equal to	\geq	$x \geq y$	Returns True if the first value is greater than or equal to the second value, else returns False. Here, the corresponding example returns True as 33 is greater than the 22.

NOTES

NOTES

For example, the steps for implementing the Greater than relational operator in VB are:

1. Open the Code window.
2. Click the down arrow button next to the Object drop-down list box and select the Form control.
3. Click the down arrow button next to the Procedure drop-down list box and select the Load event.
4. Enter the following lines of code in the Code window.

```
Private Sub Form_Load()  
Dim x As Boolean  
Dim a As Integer  
Dim b As Integer  
a = 3  
b = 6  
x = a > b 'Returns true if the value of a is greater  
than b else returns false  
MsgBox x  
End Sub
```

Figure 4.3 shows the Code window for the Greater than operator.

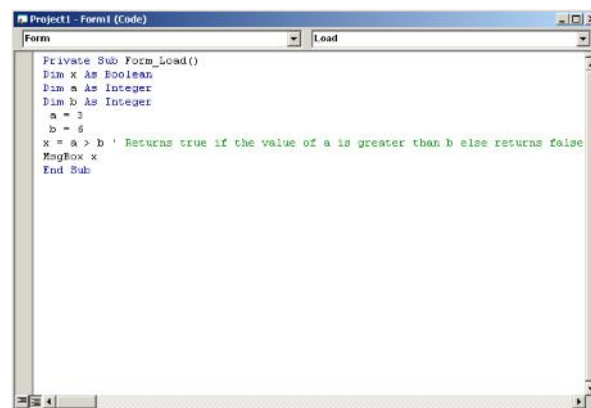


Fig. 4.3 Displaying the Code for Implementing the Greater than Operator

5. Press F5 to compile the above code. Figure 4.4 shows the output after compiling the Greater than operator.

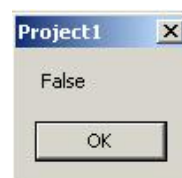


Fig. 4.4 Displaying the Output for Greater than Operator

4.4.3 Concatenation Operators

Concatenation operators are used to merge two or more strings into a single string. VB uses the & operator to merge two or more strings. For example, the steps for implementing the & operator in VB are:

1. Open the Code window.
2. Click the down arrow button next to the Object drop-down list box and select the Form control.
3. Click the down arrow button next to the Procedure drop-down list box and select the Load event.
4. Enter the following lines of code in the Code window.

```
Private Sub Form_Load()
    Dim x As String
    x = "John"
    Dim y As String
    y = "is"
    Dim z As String
    z = "boy"
    Dim a As String
    a = x & y & z
    MsgBox a
End Sub
```

Figure 4.5 shows the Code window for concatenation operator

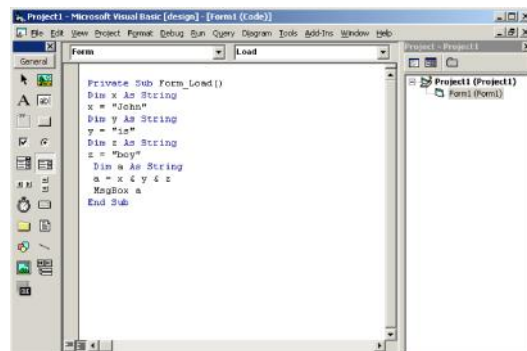


Fig. 4.5 Displaying the Code for the Concatenation Operator

Note: The concatenation operator combines John, is and boy and sets the variable a to "Johnisboy".

5. Press F5 to compile the above code. Figure 4.6 shows the output after compilation.



Fig. 4.6 Displaying the Output for the Concatenation Operator

NOTES

You can also use the + operator to concatenate two or more strings. It works same as the & operator; you just need to replace & with +.

4.4.4 Logical Operators

NOTES

Logical operators are used to compare Boolean expressions and return a Boolean result. Table 4.9 lists the various logical operations, their corresponding logical operators, examples and description.

Table 4.9 Logical Operators

Logical Operation	Logical Operator	Example	Description
Not operation	Not	Dim x As Boolean x = Not 3 > 5	The Not operand performs logical negation on the Boolean expression. If the expression evaluates True, then Not operand returns False and vice-versa. In the corresponding example, 3 is less than 5, therefore, the Boolean result for 3 > 5 is False. But the Not operand will return True and as a result, x is set to True.
And operation	And	Dim x As Boolean x = 3 > 5 And 4 < 5	The And operand returns logical conjunction of the two Boolean expressions. It returns True only if both the Boolean expressions return True, else it returns False. In the corresponding example, first Boolean expression (3 > 5) returns False and the second Boolean expression (4 < 5) returns True. As one of the expressions is returning False, the And operand returns False.
Or operation	Or	Dim x As Boolean x = 3 > 5 Or 4 < 5	The Or operand returns logical disjunction of the two Boolean expressions. It returns True if any one of the Boolean expressions returns True, else it returns False. In the corresponding example, first Boolean expression (3 > 5) returns False and the second Boolean expression (4 < 5) returns True. As one of the expressions is returning True, the Or operand returns True.
Xor operation	Xor	Dim x As Boolean x = 3 > 5 Xor 4 < 5	The Xor operand returns logical exclusion of the two Boolean expressions. It returns True if exactly only one of the Boolean expressions returns True, else it returns False. In the corresponding example, both the expressions are returning True. As more than one expression is returning True, the Xor operand returns False.

For example, the steps for implementing the And logical operator in VB are:

1. Open the Code window.
2. Click the down arrow button next to the Object drop-down list box and select the Form control.
3. Click the down arrow button next to the Procedure drop-down list box and select the Load event.

4. Enter the following lines of code in the Code window.

```
Private Sub Form_Load  
    Dim x As Boolean  
    Dim a As Integer  
    Dim b As Integer  
    Dim c As Integer  
    Dim d As Integer  
    a = 3  
    b = 6  
    c = 7  
    d = 1  
    x = a > b And c > d  
    MsgBox x  
End Sub
```

Figure 4.7 shows the Code window for the And operator.

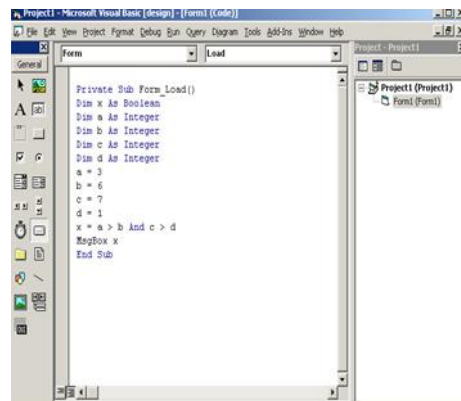


Fig. 4.7 Displaying the Code for Implementing the and Operator

5. Press F5 to compile the above code. Figure 3.8 shows the output after compiling the And operator.



Fig. 4.8 Displaying the Output

Check Your Progress

12. How many type of operators are used in Visual Basic.
13. MOD is which type of operator in VB?
14. Why Arithmetic operators are used in VB?
15. Name any two relational operators.

NOTES

4.5 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

NOTES

1. Data type is a programming element which is determine how the bits/bytes representing the values are stored in computer memory.
2. Storage capacity of currency data type is -922,337,203,685,477.5808 to 922,337,203,685,477.5807.
3. You can declare a variable and its data type in the following two ways:
 - Using explicit declaration.
 - Using implicit declaration.
4. Integer data type is used to store whole numbers and cannot be used in calculations where decimals or fractions are involved.
5. A `Variant` data type is a variable that can freely change its type. It can accept text, number or byte data easily. If you do not supply a data type, the variable is given the `Variant` data type by default.
6. The `Error Value` In a `Variant`, `Error` is a special value used to indicate that an error condition has occurred in a procedure. An `Error` value is created by converting a real number by using the `CVErr` function.
7. An alternative way of specifying data types while declaring a variable is the use of some special characters in place of data types. These special characters which specify the data type are known as type declaration characters. The final character can be a 'Type Declaration Character'. For Example
String \$
Integer %
Long &
Single !
Double #
Currency @
8. A user defined type (also called a *structure*) is a compound data type containing two or more other data types.
9. When you create a variable, VB automatically assign a value to it. This Values is known as default value of the variable.
10. A variable is a named storage location whose contents can be varied or simply a name you give for a memory area in which the value of data which was utilized by your program is contained.
11. The following are the tree level of scope for declaring variable:
 - Project level scope

- Module level scope
- Private or local level

12. Various operators used in Visual Basic are:

- a. Arithmetic operators
- b. Relational operators
- c. Concatenation operators
- d. Logical operators

13. MOD is an arithmetic operator.

14. Arithmetic operators are used to perform the numerical operations such as adding and multiplying two or more data values.

15. Following are the two relational operators:

- Equality
- Inequality

NOTES

4.6 SUMMARY

- Every programming element, such as variable, literal, constant, procedure, etc. has a data type.
- Visual Basic (VB) have a name and declared data type. The data type of a programming element determines how the bits/bytes representing these values are stored in the computer memory.
- Therefore, you have at least two types of variables. Visual Basic has a number of variable types to deal with various programming requirements.
- One needs to use different types of variables for different requirements in order to optimize speed and memory requirements.
- Data types can be applied to other elements also besides variables. When you assign a value to a property, that value has a data type.
- In fact, just about anything in Visual Basic that involves data also includes data types.
- In explicit declaration, you use the in-built Visual Basic statement, Dim statement, to declare a variable name and its data type. The Dim statement does not allow assigning values to the declared variable.
- Integer data type is used to store whole numbers and cannot be used in calculations where decimals or fractions are involved. They can store reasonably large numbers.
- Long Integer must be used only where the calculations involve very large numbers.

NOTES

- Single is the equivalent of the floating point number. It can store fractions and provide precision to a fairly high level.
- Double solves the problem of precision that the Single data type lacks. It occupies 8 bytes of memory space and should be used in applications where the requirement of precision is very high.
- Currency data type is used for holding values related to item rates, payroll details and other financial functions.
- Byte data type can hold values from 0 to 255. It cannot hold negative numbers or numbers larger than 255. Assigning negative values or values beyond 255 will result in a runtime overflow error.
- Boolean data type accepts only True or False values. Since the default value for all numeric data types is zero, the default value for a Boolean data type is also zero. The zero value is interpreted as False and a non-zero value is interpreted as True.
- Date variable holds date and time data. It can hold time from January 1 100 to December 31 9999 and time from 00.00.00 (midnight) to 23.59.59 (one second before midnight) in one second increments. It occupies 8 bytes of memory.
- The data is displayed as per the settings in your computer. You can store it in British format, American format or any other format that is available in the Regional Settings in your control panel.
- String is the most commonly used data type. Declare a variable with this data type if it will always contain a string value and not a numeric value.
- In Visual Basic, forms, controls, procedures and recordsets are all considered as objects.
- A Variant data type is a variable that can freely change its type. It can accept text, number or byte data easily. If you do not supply a data type, the variable is given the Variant data type by default.
- Null is commonly used in database applications to indicate unknown or missing data. If you assign Null to a variable of any type other than Variant, an error occurs. Assigning Null to a Variant variable does not cause an error.
- In a Variant, Error is a special value used to indicate that an error condition has occurred in a procedure. An Error value is created by converting a real number by using the CVer function.
- A Variant variable has the Empty value before it is assigned a value. The Empty value is a special value different from 0, a zero length string ("") or the Null value.
- You can also use implicit declaration to declare a variable name and its data type. VB defines a special character for each data type.

- You need to use that special character at the end of a variable name to declare a data type for that variable.
- Most of the data types could hold either string or numeric data, and the numeric types could hold either integer or floating point values.
- One of the handiest features of VB is the ability to create user defined data types.
- A user defined type (also called a *structure*) is a compound data type containing two or more other data types. You can define exactly what goes into a structure, designing it around the exact needs of your program.
- This defines a variable called 'udtEmpRec', which has the attributes defined by the structure 'EmployeeRecord'. Thus, you refer to 'udtEmpRec', in your procedural statements and not 'EmployeeRecord'.
- An alternative way of specifying data types while declaring a variable is the use of some special characters in place of data types. These special characters which specify the data type are known as type declaration characters. The final character can be a 'type declaration character'.
- It should be noted that the style of using a lower case, three character prefix is succeeded by a descriptive name in mixed case. This is the way of naming variables in all VB documentations.
- When you create a variable, VB automatically assign a value to it. This value is known as default value of the variable.
- A variable is a named storage location whose contents can be varied or simply a name you give for a memory area in which the value of data which was utilized by your program is contained.
- As the name suggests, it is a location whose contents can be varied. Like other programming language, VB also supports variables. A variable has two associated things with it, i.e., a name and its data type.
- The variable name is used to refer to the value stored in it and the data type tells what type of value can be stored in the variable.
- Any of the English alphabets A to Z must function as the first character. Both upper case and lower case letters may be used. The remaining characters can be letters, digits or the underscore (_) character.
- Declaring a variable means telling the program about it in advance.
- If the 'As' data type clause is neglected during declaration of a variable, the variable type defaults to Variant unless a type declaration character is used.
- Project Level Scope also called 'global' or 'public' or 'application' scope; the variable is accessible to all procedures in all modules of the project.
- The variables available to all the modules and procedures in an application are said to have *Project level scope*.

NOTES

NOTES

- Public keyword declares public or project level variables. Since the Public variable are available *publicly* or *globally* to all module of the application, these are also known global variable, i.e., variable with global scope.
- Module Level Scope is the variable is accessible to all procedures in the module in which it is declared.
- Module is a place where you can put your commonly used routines, functions, constants, etc. In other words, a module is a place to store commonly used things. These things may be used in many projects.
- In VB, there are three kinds of modules namely *form* module, *standard* module and *class* module.
- A form module stores everything related to a form. A standard module stores the commonly used variables, constant and procedures, etc.
- A class module stores code to create new objects, the basis of object oriented programming.
- The variable is accessible only to the procedure in which it is declared. A variable that can be used only in one procedure in which it is declared is said to have Private or local scope.
- The General Declarations section of a module is unlabelled, i.e., it is always present at the start of a code module, after the '*Option Explicit*' statement but preceding the first Sub or Function procedure.
- Declaring a variable here using the Public or Global keyword makes it a project level variable, i.e., or a module level variable if the Private or Dim keyword is used.
- Declaring a variable here makes it a local level variable. Here, the Dim or Static keyword alone can be used.
- The usual practice involves declaring all local variables within a procedure immediately succeeding the Sub or the Function header and preceding any executable statements.
- In most of the VB programs, various operations such as numerical and logical operations are performed to carry out a specific task.
- For each operation VB specifies a specific symbol, known as operators.
- Arithmetic operators are used to perform the numerical operations such as adding and multiplying two or more data values.
- Comparison operators are used to compare the two values of same data type and return a Boolean value, which represents the relation between the two values.
- A Boolean value has only two values, either True or False. These values may be numerical, strings and objects.
- Concatenation operators are used to merge two or more strings into a single string. VB uses the & operator to merge two or more strings.
- Logical operators are used to compare Boolean expressions and return a Boolean result.

10.7 KEY WORDS

- Ñ **Subroutines:** Subroutines can be thought of as miniature programs. A subroutine has a name attributed with it, much like a variable does.
- Ñ **Dim Statement:** An in-built VB statement to declare a variable name and its data type.
- Ñ **Constant:** It is a value given to a variable.
- Ñ **Data type:** It is a classification identifying one of the various types of data.
- Ñ **Variable:** It is a named storage location whose contents can be varied.
- Ñ **Function:** It refers to block of code that performs a specific task in a computer program.
- Ñ **Integer:** It refers to a data type in computer science that represents some finite subset of the mathematical integers.
- Ñ **Syntax:** It refers to the rules governing the formation of statements in a programming language.
- Ñ **Operators:** Various operations such as numerical and logical operations are performed to carry out a specific task. For each operation VB specifies a specific symbol, known as operators.

NOTES

4.8 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. Why we use Dim statement?
2. Give the definition of variable.
3. Explain about the data type.
4. How will you define a Boolean constant?
5. Write the rules for defining data types.
6. What are the two ways to declare a variable?
7. Explain the term Boolean for data type.
8. What is null value for explicit declaration?
9. Define the term user defined data types.
10. State about the private or local level.
11. How many types of arithmetic operators are used in Visual Basic (VB) for performing mathematical calculations?

NOTES

Short-Answer Questions

1. Briefly discuss about the data types and storage capacity table in Visual Basic giving examples.
2. Explain about the different ways to declare a variable and its data types giving syntax and example codes.
3. Discuss about the conventions used for naming a variable.
4. Differentiate between the explicit and implicit declarations giving example.
5. Elaborate on the defined data types and the alternate way of specifying data types while declaring a variable with the help of examples.
6. Briefly discuss the static variables and different scopes for static variables giving appropriate examples.
7. Analyse about the general declarations section and sub function procedure giving appropriate examples.
8. Expressions used for various logical operator.
9. Elaborate briefly on the arithmetic operators which are used in VB for performing mathematical calculations.

4.9 FURTHER READINGS

- Cornell, Gary. 1998. *Visual Basic 6 from the Ground Up*. New Delhi: Tata McGraw-Hill.
- Warner, Scott L. 1998. *Teach Yourself Visual Basic 6*. New Delhi: Tata McGraw-Hill.
- Jerke, Noel. 1999. *Visual Basic 6 - The Complete Reference*. New York: McGraw-Hill.
- Smith, Eric A., Valor Whisler and Hank Marquis. 1998. *Visual Basic 6 Programming Bible*. New Jersey: John Wiley & Sons, Inc.
- Azam, M. 2001. *Programming with Visual Basic*. New Delhi: Vikas Publishing House Pvt. Ltd.
- Manchanda, Mahesh. 2009. *Visual Programming*. New Delhi: Vikas Publishing House Pvt. Ltd.
- Balena, Francesco. 1999. *Programming Microsoft Visual Basic 6.0*. Bangalore: WP Publishers and Distributors (P) Ltd.
- Petroutsos, Evangelos. 1998. *Mastering Visual Basic 6, 1st Edition*. New Delhi: BPB Publications.
- Deitel, Harvey M., Paul J. Deitel and T. Tem R. Nieto. 1999. *Visual Basic 6: How to Program*. New Jersey: Prentice-Hall.
- Norton, Peter. 1998. *Peter Norton's Complete Guide to Visual Basic 6*. New Delhi: Techmedia.
- Reselman, Bob and Richard A. Peasley. 1998. *Using Visual Basic 6*. New Jersey: Pearson Education (Que Publishing).
- Donald, Bob and Oancea Gabriel. 1999. *Visual Basic 6 from Scratch*. New Delhi: Prentice-Hall of India.

UNIT 5 LOOPING AND DECISION CONTROL STRUCTURES

*Looping and Decision
Control Structures*

NOTES

- 5.0 Introduction
- 5.1 Objectives
- 5.2 Looping and Decision Control Structures
 - 5.2.1 Decision Structure
 - 5.2.2 Loop Structure
- 5.3 Answers to Check Your Progress Questions
- 5.4 Summary
- 5.5 Key Words
- 5.6 Self-Assessment Questions and Exercises
- 5.7 Further Readings

5.0 INTRODUCTION

Visual Basic is a graphical version of the old BASIC (Beginner's All-purpose Symbolic Instruction Code) language that was quite popular among programmers. It is a highly interactive and is a Graphical User Interface (GUI) programming language. This language increases the productivity of programmers by providing various features to create effective and robust.

The capacity to make decisions is needed for a useful machine.

The secret to programming the machine to make the right choices is to ensure that you understand how to accurately analyse and evaluate the expression that represents the decision.

Visual Basic offers more than one way for decisions to be made.

To make a program more flexible and efficient, the flow of execution can be altered using various control statements. Different types of control flow statements, such as selection statements and iteration statements.

Control Structures are just a way to specify flow of control in programmes. Any algorithm or software will be more transparent and understood if they use self-contained modules called as logic or control structures. Depending on such parameters or conditions, it essentially analyses and chooses the direction in which a programme flows.

There could be a case where you need to execute a code block many times. In general, statements are executed sequentially in a function, the first statement is executed first, followed by the second statement, and so on. Programming languages provide different structures of control that allow more complicated paths of execution. A loop statement allows one several times to execute a statement or set of statements.

In this unit, you will study about the looping and decision control structure, if then else, structure select structure, for next, do while and while...wend.

NOTES

5.1 OBJECTIVES

After going through this unit, you will be able to:

- Understand the different types of control statements in VB.NET
- Explain the various types of selection statements
- Discuss the different types of iteration statements
- Know about the for...next loop
- Elaborate on the select structure
- Define the basics of do...while and while...wend

5.2 LOOPING AND DECISION CONTROL STRUCTURES

Statements are used to control the flow of program execution control. Visual Basic supports various control statements given as follows:

5.2.1 Decision Structure

Structures for decision-making enable the programmer to define one or more conditions to be evaluated or checked by the program, along with a statement or statements to be executed if the condition is determined to be valid, and other statements to be executed optionally if the condition is determined to be false.

To be assessed by the program with statements, we can specify more than one condition. If the given condition is valid, the statement or block is executed according to the condition, and another statement is executed if the condition is false.

If...Then Structure

The If...Then selection structure performs the specified action only when the condition is True. If condition is false, no action is taken.

The **Syntax** of the If...Then selection :

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

The **Example** of the If...Then selection :

```
If gradepoint = 9 Then  
    txtGrade.Text = "A+"  
End If
```

If ... Then ...Else Structure

The If ...Then ... Else selection structure is useful when you want to execute the specified statements when the condition is True and you have different statements for different conditions.

The **Syntax** of the If ...Then selection :

```
If < condition 1 > Then
    statement
ElseIf < condition 2 > Then
    statement
Else
    statement
End If
```

The **Example** of the If...Then selection

```
If gradepoint = 9 Then
    txtGrade.Text = "A+"
ElseIf gradepoint = 8 Then
    txtGrade.Text = "A"
ElseIf gradepoint = 7 Then
    txtGrade.Text = "A-"
Else
    txtGrade.Text = "Invalid"
End If
```

Select ...Case Statement

The Select ...Case structure helps to get rid of the long series of If statements. This structure tests the condition just once. It executes one of the several groups of statements on the expression.

The **Syntax** of the Select ...Case statement :

```
Select Case testexpression
    Case expression1
        Statements
    Case expression2
        Statements
    Case Else
        Statements
End Select
```

The **Example** of the Select ...Case statement :

```
Select Case ascivariable
    Case 64
```

NOTES

NOTES

```
MsgBox "a"  
Case 65  
MsgBox "b"  
Case 66  
MsgBox "c"  
Case Else  
MsgBox "Invalid Number"  
End Select
```

Note that the term `Select` used in the `Select...Case` statement specifies a keyword that shows the starting point of the `Select...Case` statement and holds a test expression. `testexpression` specifies an integral expression defined by using `Integer` and `Char` data types. `Case` in Visual Basic specifies a keyword that holds an expression list which fulfils the `testexpression`. `Case Else` specifies a keyword that runs the code within the `Case Else` block when none of the cases within the

`Select...Case` statement executes.

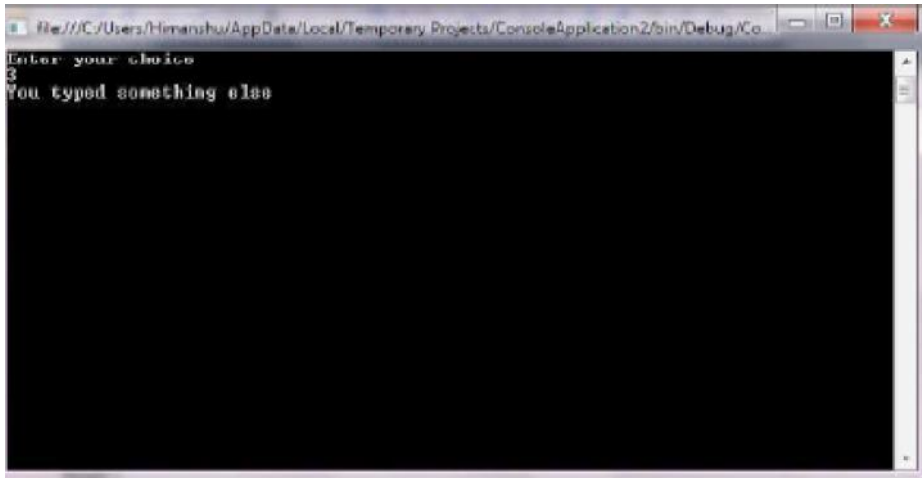
The `Select...Case` statement tests the value of the test expression in a sequence and compares it with the list of different cases. When a match is found, the control is transferred to that particular `Case` block and the statements contained in that particular `Case` block are executed.

Example 5.1: A program to demonstrate the use of nested `Select . . . Case` statement.

```
Imports System. Console  
Module Module1  
Sub Main ()  
Dim value1 As Integer  
WriteLine ("Enter your choice")  
value1 = Integer. Parse (ReadLine ())  
Select Case value1  
Case 1  
WriteLine ("You typed one")  
Case 2  
WriteLine ("You typed two")  
Case 5  
WriteLine ("You typed five")  
Case Else  
WriteLine ("You typed something else")  
End Select  
Read ()  
End Sub  
End Module
```

The output of the program is

Looping and Decision
Control Structures



NOTES

5.2.2 Loop Structure

There could be a case where you need to execute a code block many times. In general, statements are executed sequentially in a function, the first statement is executed first, followed by the second statement, and so on.

Programming languages provide different structures of control that allow more complicated paths of execution. A loop statement allows us to execute a statement or group of statements multiple times.

Do ... While Loop

The Do loop structure executes a series of statements as long as a given condition is true. It allows you to check a condition and execute a code block, if that condition is true. The Do...While version allows you to check for the true condition. You can use a Do...Until version until you check for a false.

The **Syntax** of the Do loop Structure :

```
Do [ { While | Until } condition ]  
Statements  
Loop  
Or  
Do  
Statements  
Loop [ { While | Until } condition ]
```

The **Example** of the Do loop Structure :

```
Dim count as Integer  
Dim sum as Integer  
Count = 1
```

NOTES

```
Sum = 0
Do While count < 10
    Sum = sum + count
    Count = count + 1
Loop
```

```
Txt_sum.Text = sum
```

While...Wend

The While ...Wend statement executes a series of statements as long as the given condition is true.

The **Syntax** of the While...Wend statement :

```
While Condition
Statements
Wend
```

The **Example** of the While ...Wend statement :

```
Dim count As Integer
    Dim sum As Integer
    count = 1
    sum = 0
    While count < 10
        sum = sum + count
        count = count + 1
    Wend
txt_sum.Text = sum
```

For ... Next

The For ... Next executes the block of code a fixed number of times.

The **Syntax** of the For ...Next Structure :

For variable initial_value to Final_value Step increment or decrement

Statements

Next

The **Example** of the For ...Next Structure :

```
Dim I as Integer
For I = 100 To 1 Step -10
    Print I
Next I
```


Check Your Progress

1. What are the decision structures available in Visual Basics?
2. Why we use if ... then ... else structure in Visual Basic?
3. What is a select...case statement in VB.NET?
4. Write down the syntax of select ... case statement.
5. Name the looping structures in Visual Basic.
6. Which statement executes at least once and continues executing until its loop-continuation condition becomes False.
7. Write down the syntax for the while ... wend statement.

NOTES

5.3 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Following are the decision structure available in Visual Basic:
 - If ...Then
 - If ... Then ... Else
 - Select ... Case
2. The If ...Then ... Else selection structure is useful when you want to execute the specified statements when the condition is True and you have different statements for different conditions.
3. The Select ...Case structure helps to get rid of the long series of If statements. This structure tests the condition just once. It executes one of the several groups of statements on the expression.
4. The Syntax of the Select ...Case statement :

```
Select Case testexpression
Case expression1
Statements
Case expression2
Statements
Case Else
Statements
End Select
```
5. Following are the looping structure in Visual Basic :
 - Do ... While Loop
 - While...Wend
 - For...Next
6. Do... statement executes at least once and continues executing until its loop-continuation condition becomes False.
7. The Syntax of the While...Wend statement is :

```
While Condition
Statements
Wend
```

NOTES

5.4 SUMMARY

- Statements are used to control the flow of program execution control.
- The If...Then selection structure performs the specified action only when the condition is True.
- The If ...Then ... Else selection structure is useful when you want to execute the specified statements when the condition is True and you have different statements for different conditions.
- Select...Case statement specifies a keyword that shows the starting point of the Select...Case statement and holds a test expression. testexpression specifies an integral expression defined by using Integer and Char data types.
- Case in Visual Basic specifies a keyword that holds an expression list which fulfils the testexpression. Case Else specifies a keyword that runs the code within the Case Else block when none of the cases within the Select...Case statement executes.
- The Select...Case statement tests the value of the test expression in a sequence and compares it with the list of different cases.
- When a match is found, the control is transferred to that particular Case block and the statements contained in that particular Case block are executed.
- The Select ...Case structure helps to get rid of the long series of If statements. This structure tests the condition just once. It executes one of the several groups of statements on the expression.
- The Do loop structure executes a series of statements as long as a given condition is true. It allows you to check a condition and execute a code block, if that condition is true.
- The Do-While version allows you to check for the true condition. You ceruse a Do-Until version until you check for a False.
- The While ...Wend statement executes a series of statements as long as the given condition is True.
- The For ... Next executes the block of code a fixed number of times.

5.5 KEY WORDS

- **Control statements:** Control the flow of a program during execution.
- **Declaration:** Declares the variables to store the property value.
- **Library functions:** The built-in functions which are defined in the Visual Basic (VB) library.
- **Decision structure:** Structures for decision-making enable the programmer to define one or more conditions to be evaluated or checked by the program,

along with a statement or statements to be executed if the condition is determined to be valid, and other statements to be executed optionally if the condition is determined to be false.

- **Loop structure:** There could be a case where you need to execute a code block many times. In general, statements are executed sequentially: in a function, the first statement is executed first, followed by the second statement, and so on.

NOTES

5.6 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. Explain control flow statements.
2. Elaborate on the purpose of decisions-making statements.
3. State the looping statements.
4. What are the ways to create an infinite loop?
5. Explain if ...then, select ... case, looping statements.
6. Write a program to find the area of a circle.
7. Write a code to demonstrate the use of nested Select...Case statement.

Long-Answer Questions

1. Briefly discuss the looping and decision control structures in VB giving appropriate examples.
2. Rewrite the following VB code using on Switch statement. Also specify how it is different than previous code.

```
If(ch='a' Or ch='A') then
```

```
    countA+=1
```

```
ElseIf(ch='e' Or ch='E') then
```

```
    countE+=1
```

```
ElseIf(ch='i' Or ch='I') then
```

```
    countI+=1
```

```
ElseIf(ch='o' Or ch='O') then
```

```
    countO+=1
```

```
ElseIf(ch='u' Or ch='U') then
```

```
    countU+=1
```

```
Else : Console.WriteLine("No vowel letter")
```

```
End If
```

3. Write a simple program to print three statement in Visual Basic.
4. Write a code to check whether the number is even or odd.

NOTES

5. Write a program to display the Days name using the select case statement in VB.NET.
6. Write a program to perform an arithmetic operation using the Select ... Case statement in VB.NET.
7. Write a program to compute the total number of students who have passed in the FIRST, SECOND and THIRD division. The condition is:
Total Marks > 60 – First Class
Total Marks > 50 – Second Class
Total Marks > 40 – Third Class
Total Marks < 40 – Fail
8. Discuss the following with the help of examples
 - Do ...loop
 - For...Next loop
 - While...Wend

5.7 FURTHER READINGS

- Cornell, Gary. 1998. *Visual Basic 6 from the Ground Up*. New Delhi: Tata McGraw-Hill.
- Warner, Scott L. 1998. *Teach Yourself Visual Basic 6*. New Delhi: Tata McGraw-Hill.
- Jerke, Noel. 1999. *Visual Basic 6 - The Complete Reference*. New York: McGraw-Hill.
- Smith, Eric A., Valor Whisler and Hank Marquis. 1998. *Visual Basic 6 Programming Bible*. New Jersey: John Wiley & Sons, Inc.
- Azam, M. 2001. *Programming with Visual Basic*. New Delhi: Vikas Publishing House Pvt. Ltd.
- Manchanda, Mahesh. 2009. *Visual Programming*. New Delhi: Vikas Publishing House Pvt. Ltd.
- Balena, Francesco. 1999. *Programming Microsoft Visual Basic 6.0*. Bangalore: WP Publishers and Distributors (P) Ltd.
- Petroutsos, Evangelos. 1998. *Mastering Visual Basic 6, 1st Edition*. New Delhi: BPB Publications.
- Deitel, Harvey M., Paul J. Deitel and T. Tem R. Nieto. 1999. *Visual Basic 6: How to Program*. New Jersey: Prentice-Hall.
- Norton, Peter. 1998. *Peter Norton's Complete Guide to Visual Basic 6*. New Delhi: Techmedia.
- Reselman, Bob and Richard A. Peasley. 1998. *Using Visual Basic 6*. New Jersey: Pearson Education (Que Publishing).
- Donald, Bob and Oancea Gabriel. 1999. *Visual Basic 6 from Scratch*. New Delhi: Prentice-Hall of India.

UNIT 6 USING INTRINSIC VISUAL BASIC CONTROLS WITH METHODS AND PROPERTIES

*Using Intrinsic Visual
Basic Controls with
Methods and Properties*

NOTES

- 6.0 Introduction
- 6.1 Objectives
- 6.2 Basic Controls
- 6.3 Control Array
- 6.4 Answers to Check Your Progress Questions
- 6.5 Summary
- 6.6 Key Words
- 6.7 Self-Assessment Questions and Exercises
- 6.8 Further Readings

6.0 INTRODUCTION

Visual Basic (VB) comes with many built-in controls. All controls are not equally useful. Some of them are very useful. They can be used in every application you write. Others you will use only when you have a special need for the features the controls offer.

Controls are objects that combine code with visual parts. They require a container i.e., as an object (like form or any other control) that carries the control. The developer part of the control includes an icon for the ToolBox. The part of the control has visual representation when it is placed on a container, such as a form. Control has properties, method and events, which are collectively known, as controls interface.

The creator of the control is responsible for the programming that enables the interface: for making sure that properties appear in the properties window, that events are fired when they are supposed to be fired, etc. The user of the control does not see the internal implementation of the control application. The users interaction with the control depends on the interface settings that the application developer has made and on code added by the developer.

In Visual Basic (VB), a control array is a group of related controls in a Visual Basic form that share the same event handlers. Control arrays are always single-dimensional arrays, and controls can be added or deleted from control arrays at runtime. One application of control arrays is to hold menu items, as the shared event handler can be used for code common to all of the menu items in the control array.

NOTES

Control arrays are a convenient way to handle groups of controls that perform a similar function. All of the events available to the single control are still available to the array of controls, the only difference being an argument indicating the index of the selected array element is passed to the event. Hence, instead of writing individual procedures for each control Visual Basic intrinsic controls, methods and properties, such as, you only have to write one procedure for each array.

In this unit, you will study about the basic Controls like Label, TextBox, Command Button, Frame, Checkbox, Option Button, ListBox, ComboBox, File ListBox, Directory List Box, Drive List Box and Tab Order.

6.1 OBJECTIVES

After going through this unit, you will be able to:

- Understand the significance of various controls
- Explain about the TextBox and Label Control
- Define the CommandButton and OptionButton
- Discuss about the CheckBox and ListBox Control
- Elaborate on the concept of methods and Events of various controls
- Know about the control arrays

6.2 BASIC CONTROLS

In Visual Basic, *forms* are the foundations you generally use to build programs. A form is where you put all the things that people interact with as they use your program. Those things you put on the form are *controls*, which enable the people who use your program to do things, such as enter text and click buttons.

All the controls in the ToolBox except the Pointer are referred as objects in VB. These objects have associated properties, methods and events. The programming objects are loaded with properties. A property is a named attribute of a programming object. Properties define the characteristics of an object, such as size, color, etc., or sometimes the way in which it behaves.

For the most part, you'll use a relatively small set of controls when you program in Visual Basic. However, these controls are very powerful. With them, you can add buttons, check boxes, labels, and text boxes to your programs. You can use them to see files on your hard drive right from your program. You can even read a database! These basic controls are *intrinsic controls*, and they're available in every edition of Visual Basic 6.

The intrinsic controls are available whenever you use Visual Basic. During design time, you can access them from the ToolBox . Table 6.1 lists the intrinsic controls.

Table 6.2 *Different Types of Intrinsic Controls and their Description*

Control	Description
Pointer	Used for interacting with the controls on the form.
PictureBox	Used for displaying images.
TextBox	Used for accepting user input that can display only editable text.
Frame	Used for grouping other controls.
CommandButton	Used for initiating an action by pressing on the button.
CheckBox	Used for making a choice for user (checked or unchecked).
OptionButton	Used in groups where one at a time can be true.
ListBox	Used for providing a list of items.
ComboBox	Used for providing a short list of items.
HScrollBar	A horizontal scrollbar.
VScrollBar	A vertical scrollbar.
Timer	Used for performing tasks at particular intervals.
DriveListBox	Used for accessing the system drives.
DirListBox	Used for accessing the directories on the system.
FileListBox	Used for accessing the files in the directory.
Shape	Used for drawing circles, squares, rectangles, ellipses, etc.
Line	Used for drawing lines.
Image	Used for displaying images but has less capability than the PictureBox.
Data	Used for connecting a database.
OLE	Used for interacting with other application of Windows.
Label	Used for displaying texts that cannot be edited.

NOTES

Syntax and Attributes of Controls

A control is an object that can be drawn on a Form object for enabling or enhancing user interaction with an application. Controls have properties for defining various aspects of their appearance like size, position and color, and the behavioral aspects like their response to input from a user. They can react to events set off by the system or initiated by the user. A code, for instance, could be written in a `CommandButton` control's click event procedure that would load a file or display a result.

Container Control

A container control can hold other controls within it, for example, a `Frame` (there can be multiple controls inside a frame) or a `Picture Box` (it holds a picture) or simply your `Form` (you can put so many controls on it). Controls inside containers are known as child controls. Child controls can exist completely inside their containers. So you cannot move them outside their container and if you try to drag them beyond the boundary of their container, part of the control gets hidden. When you delete a container control, all its child controls automatically get deleted.

NOTES

Form Control

In VB, the *Form* acts as the container for all the controls that form the interface. The *Form* is the top-level object in a VB application, and every application starts with the *Form*.

Appearance of Forms

The main characteristic of a Form is the title bar on which the Forms caption is displayed. On the left end of the title bar is the Control Menu icon. Clicking this icon opens the control menu. Maximize, Minimize and Close buttons can be found on the right side of the Form. Clicking on these buttons performs the associated function. Figure 6.1 illustrates the appearance of a Form.

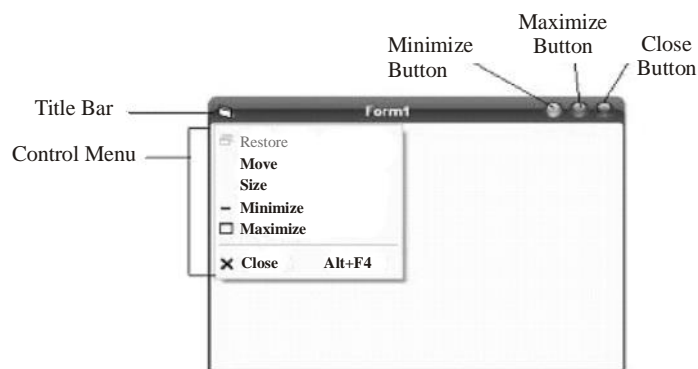


Fig. 6.1 General Appearance of Frequently Used Container Control Form

The control menu contains the following characteristics:

- **Restore:** Restores a maximized Form to its size before it was maximized; available only if the Form has been maximized
- **Move:** Lets the user move the Form around with the mouse
- **Size:** Lets the user resize the control with the mouse
- **Minimize:** Minimizes the Form
- **Maximize:** Maximizes the Form
- **Close:** Closes the Form

Some common properties used to customize a Form appearance are as follows:

Table 6.2 Common Properties of Form Control

Property	Description														
MinButton, MaxButton	These two properties, if True display the Minimize and Maximize buttons on the title bar. You can set them to False to hide the corresponding button on the title bar. By default True .														
ControlMenu	This property displays the control menu, if it is set to True . By default, it is True . You can set it to False to hide the Control menu icon.														
BorderStyle	<p>This property determines the border's style for a Form and also its appearance. It can take any of the following values:</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>0 None</td><td>No border for Form, cannot be resized.</td></tr> <tr> <td>1 Fixed Single</td><td>Visible border, but Form cannot be resized.</td></tr> <tr> <td>2 Sizable</td><td>Visible border. Form can be moved and resized also.</td></tr> <tr> <td>3 Fixed Dialog</td><td>For fixed dialog boxes</td></tr> <tr> <td>4 Fixed ToolWindow</td><td>Form has a close button only, cannot be resized</td></tr> <tr> <td>5 Sizeable ToolWindow</td><td>Same as no. 4 but the Form can be resized.</td></tr> </table>	Value	Description	0 None	No border for Form, cannot be resized.	1 Fixed Single	Visible border, but Form cannot be resized.	2 Sizable	Visible border. Form can be moved and resized also.	3 Fixed Dialog	For fixed dialog boxes	4 Fixed ToolWindow	Form has a close button only, cannot be resized	5 Sizeable ToolWindow	Same as no. 4 but the Form can be resized.
Value	Description														
0 None	No border for Form, cannot be resized.														
1 Fixed Single	Visible border, but Form cannot be resized.														
2 Sizable	Visible border. Form can be moved and resized also.														
3 Fixed Dialog	For fixed dialog boxes														
4 Fixed ToolWindow	Form has a close button only, cannot be resized														
5 Sizeable ToolWindow	Same as no. 4 but the Form can be resized.														
BackColor	Specifies the Form's background color.														
BorderStyle	Determines how the Form window appears.														
Enabled Picture	Determines whether the Form is active or not Determines a graphic image that appears on the background of the Form at runtime.														

NOTES

Frame Control

A *Frame* control is used to group various controls. A *Frame* control is a container control, i.e., it contains other controls in it; it does not carry out any job/action by itself, i.e., it does not respond to any event by itself.

Some common properties of *Frame* control are as follows:

Table 6.3 Common Properties of Frame Control

Property	Description
Caption	The property used to display the text in the Frame control.
Font	The property used to specify the display font, style and size of the caption of the Frame.
(Name)	The property used to name the Frame control. Frame is represented in coding by using this name.
BorderStyle	The property used to specify the style of border. It can be either 0—None or 1—Fixed Single. It appears under the Appearance category.
Appearance	The property used to set the look of the frame. It can either be 0—Flat or 1—3D.
Visible	The property used to set a value indicating whether the frame should be visible or not.

NOTES

TextBox Control

A TextBox control is an edit field or edit control, displaying information entered at design time by the user, or assigned to the control using code at runtime. Basically, the TextBox control is a small text editor that provides all the basic text-editing facilities; you can insert and select a text, scroll the text if it does not fit in its visible area and you can even exchange text with other application through the clipboard.

Some common properties and methods of TextBox control are as follows:

Table 6.4 Common Properties of TextBox Control

Property/ Method	Description
<i>Properties</i>	
Enabled	Specifies whether the user can interact with this control or not.
Index	Specifies the control array index.
Locked	If this control is set to True, the user can use it; else, if this control is set to false, the user cannot use the control.
MaxLength	Specifies the maximum number of characters to be input. Default value is set to 0 meaning the user can input any number of characters.
MousePointer	Using this, the shape of the mouse pointer can be set when over a TextBox.
Multiline	By setting this property to True, there can be more than one line in the TextBox.
PasswordChar	This specifies mask character to be displayed in the TextBox.
ScrollBars	This is for setting either the vertical or horizontal scrollbars for appearing in the TextBox. The user can also set it to both the horizontal and vertical scrollbars. This property is used with the Multiline property.
Text	Specifies the text to be displayed in the TextBox at runtime.
ToolTipIndex	This is used for displaying the text that is displayed in the control.
Visible	By setting this, the Textbox control can be made visible or invisible at runtime.
<i>Methods</i>	
SetFocus	Transfers focus to the TextBox.
Event procedures	
Change	Action happens when the TextBox changes.
Click	Action happens when the TextBox is clicked.
GotFocus	Action happens when the TextBox receives the active focus.
LostFocus	Action happens when the TextBox loses it focus.
KeyDown	Called when a key is pressed while the TextBox has the focus.
KeyUp	Called when a key is released while the TextBox has the focus.

For working with a part of a text in a TextBox, the text can be selected using the following three properties:

- **SelLength:** Returns or sets the number of selected characters.
- **SelStart:** Returns or sets the starting point of the text selected. *SelStart* indicates the position of the inserted point when no text is selected.
- **SelText:** Returns or sets the string that contains the currently selected text. *SelText* will have a zero-length string if no text is selected.

NOTES

Label Control

It is a graphical control which the user can use for displaying text that cannot be edited. A Label control displays text that cannot be changed. Basically, Labels are used for identifying controls (such as scrollbars and TextBoxes) that have no Caption property of their own.

Some common properties of Label control are as follows:

Table 6.5 Common Properties of Label Control

Property	
Caption	The property used to display the text in the Label control.
Font	The property used to specify the display font, style and size of the caption of the Label.
(Name)	The property used to name the label control. Label is represented in coding by using this name.
Alignment	The property used to specify the alignment of Label's Caption. 0—Left Justify, 1—Right justify and 2—Centralized. By default text is left justified.
WordWrap	The property used to set the word wrapping option for the caption text. If we set this property to true then the text is wrapped and expanded vertically.
Autosize	This property causes the control to expand horizontally and adjust to the size of its contents.
BorderStyle	This property, if set to 1, allows the label to appear with a border that gives it a similar look to that of a text box.

CommandButton Control

The CommandButton control allows the user to click on it to perform specific actions. When the user chooses the button, it carries out the appropriate action. It also looks as if it is being pushed in and released. The *Click event procedure* is invoked whenever the user clicks a button. You can put code in the Click event procedure for performing any action.

Some common properties of `CommandButton` control are as follows:

Table 6.6 Common Properties of `CommandButton` Control

Property	Description
BackColor	This property specifies the background colour of the <code>CommandButton</code> and selects a colour in the <code>BackColor</code> property.
Cancel	This property determines whether the command gets a click event if the user presses <code>Esc</code> .
Caption	To display text on a <code>CommandButton</code> control and set its caption property.
Default	This property determines if the command button responds to an <code>Enter</code> key press even if another control has the focus..
Enabled	This property enables or disables the buttons set the <code>Enabled</code> property to <code>True</code> or <code>False</code> .
Font	This property opens a font dialog box in which you can set the font name, style and size.
Height	This property holds the height of the command button's in twips.
MousePointer	This property determines the shape of the mouse pointer.
Picture	This property holds the name of an icon graphic image that appears on the command button as long as the style property is set to 1— <i>Graphical</i> .
Style	This property determines whether the command button appears as a standard windows command button (set to 0— <i>Standard</i>) or with a colour and possible picture (set to 1— <i>Graphical</i>).
ToolTip Text	This property holds the text that appears as a <code>ToolTip</code> at runtime.
Visible	To make visible or invisible the buttons at run-time, set the <code>Visible</code> property to <code>True</code> or <code>False</code> .

NOTES

OptionButton Control

The `OptionButton` control provides a set of choices from which a user can select only one button by:

- Clicking it at runtime
- Assigning the value property of the `Option Button` to *True*. The code to assign it to `True` like `Option1.Value = True`
- Using the shortcut keys specified in the `Caption` of a `Label`.

When one button is selected, all the other buttons in the group are cleared.

Some common properties of **OptionButton** control are as follows:

Table 6.7 Common Properties of **OptionButton** Control

Property	Description
Caption	This property is used to display the text for the option button.
(Name)	This property is used to name the option button that can be represented in coding.
Alignment	This property is used to set the alignment of the option button. You can set either left or right justify.
Style	This property determines whether the option button appears as a standard windows command button (set to 0— <i>Standard</i>) or with a colour and possible picture (set to 1— <i>Graphical</i>).
Value	This property is used to set the state of option button. If set to <i>True</i> , the option button appears selected.

NOTES

CheckBox Control

A **CheckBox** control is similar to an option button. **CheckBox** controls are used for offering a small set of choices from which one can choose one or more options. The **Value** property of both the controls is tested to check the current state. **Check Boxes** are valid as a single control and are not mutually exclusive.

Some common properties of the **CheckBox** control are as follows:

Table 6.8 Common Properties of the **CheckBox** Control

Property	Description								
Caption	It is used to display text for the option button.								
(Name)	It is used to name the option button that can be represented in coding.								
Alignment	It is used to set the alignment of the option button. You can set either left or right justify.								
Enabled	It specifies whether the option button or check box is enabled or disabled.								
Value	It specifies whether the option button or the check box has been checked or not. The value is either <i>True</i> or <i>False</i> for the option button and value for check boxes are as follows: <table border="1"> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>0</td><td>Unchecked</td></tr> <tr> <td>1</td><td>Checked</td></tr> <tr> <td>2</td><td>Grayed</td></tr> </table>	Value	Description	0	Unchecked	1	Checked	2	Grayed
Value	Description								
0	Unchecked								
1	Checked								
2	Grayed								

ListBox Control

A **ListBox** control displays a list of items from which one or more items can be displayed. A user gets a list of choices from the list box. The choices are displayed in a single column by default, though multiple columns can also be set up. Scroll bars will automatically appear on the control if the number of items is more than what can be displayed in the **ListBox**. The user can then scroll up and down, or left to right through the list.

Some common properties and methods of the ListBox control are as follows:

Table 6.9 Common Properties and Methods of the ListBox Control

Property/Methods	Description								
Columns	<p>It is used for specifying the number of columns in a list box. This property can have the following values:</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>0</td><td>Single column list box with vertical scrolling.</td></tr> <tr> <td>1</td><td>Single column list box with horizontal scrolling.</td></tr> <tr> <td>>1</td><td>Multiple column list box with horizontal scrolling.</td></tr> </table>	Value	Description	0	Single column list box with vertical scrolling.	1	Single column list box with horizontal scrolling.	>1	Multiple column list box with horizontal scrolling.
Value	Description								
0	Single column list box with vertical scrolling.								
1	Single column list box with horizontal scrolling.								
>1	Multiple column list box with horizontal scrolling.								
MultiSelect	<p>It is used for determining whether the user can select the List multiple items or not. This property must be set at design time. At runtime, you can change this property. It can take the following values:</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>0</td><td>Multiple selection not allowed; default setting.</td></tr> <tr> <td>1</td><td>Simple multiple selection, i.e., an item can be selected or deselected either through a mouse click or by pressing the spacebar on it.</td></tr> <tr> <td>2</td><td>Extended multiple selection, i.e., items can be selected either by pressing Shift and clicking the mouse button and Ctrl key simultaneously.</td></tr> </table>	Value	Description	0	Multiple selection not allowed; default setting.	1	Simple multiple selection, i.e., an item can be selected or deselected either through a mouse click or by pressing the spacebar on it.	2	Extended multiple selection, i.e., items can be selected either by pressing Shift and clicking the mouse button and Ctrl key simultaneously.
Value	Description								
0	Multiple selection not allowed; default setting.								
1	Simple multiple selection, i.e., an item can be selected or deselected either through a mouse click or by pressing the spacebar on it.								
2	Extended multiple selection, i.e., items can be selected either by pressing Shift and clicking the mouse button and Ctrl key simultaneously.								
Sorted	This property determines whether the items in the list will be sorted or not.								
Style	It is used to set the alignment of the option button. You can set either left or right justify.								
ListIndex	<p>The property gives the index of the selected item in the list; if multiple items are selected, then the ListIndex property stores the index of the most recently selected item. Consider the following statement that removes the selected item from List1 control:</p> <p>List1.Removeitem List1.Listindex</p> <p>In case no item is selected, the ListIndex property has a negative value.</p>								
NewIndex	This property returns the item's index most recently added to a ListBox control.								
Methods									
AddItem	Adds an item to the ListBox								
Clear	Removes all items from the ListBox								
RemoveItem	Removes the specified item from the ListBox								
SetFocus	Transfers focus to the ListBox								
Event Procedures									
Change	Called when text in ListBox is changed								
DropDown	Called when the ListBox drop-down list is displayed								
GotFocus	Called when ListBox receives the focus								
LostFocus	Called when ListBox loses its focus								

Some Common Methods of ListBox Control

In order to properly work with the ListBox control within your application, you should be able to do the following tasks:

- Add items to the list.
- Remove items from the list.
- Access individual items in the list.

NOTES

Adding Items to a List

To add items to a ListBox, use the Additem method. It is possible to populate the list at design time or runtime.

Design Time: To add items to a list at design time, click on the List property in the property box and then add the items. Press Ctrl+Enter after adding each item as shown in Figure 6.2.



Fig. 6.2 Design Time View of Adding Item in a ListBox

Runtime: The AddItem method is used to add items to a list at runtime. The AddItem method uses the following syntax:

```
<ListBox>.AddItem item [, Index]
```

Where,

- The `ListBox` argument is a name of the ListBox, and
- The `item` argument is a string that represents the text to add to the list.
- The `index` argument specifies where the new item in the list is to be inserted. The first position is represented by an index of 0. The item is inserted at the end (or in the proper sorted order) if index is omitted.

While list items are generally added in the `Form_Load()` event procedure, the `Additem` method provides the ability of adding items to the list dynamically.

The following is an example to add item to a list box named List1:

```
Private Sub Form_Load( )
    List1. Additem 'New Delhi'
    List1.Additem 'Mumbai'
    List1.Additem. 'Kolkotta'
    List1.Additem 'Chennai'
End Sub
```

For adding an item to a list at a particular position, an index value for the new item should be specified. The next line of code, for example, inserts '*Chandigarh*' into the first position, adjusting the position of other items downward:

```
List1.Additem 'Chandigarh' , 0
```

NOTES

NOTES

Removing Items from a List

The `RemoveItem` method is used to remove an item from a list. The syntax for this is given as follows:

```
<Listbox>.RemoveItem <Index>
```

The `<Listbox>` and `<Index>` arguments are the same as for `Additem`.

For example, for removing the first entry in a list, add the following line of code:

```
List1.Removeitem 0
```

The following code verifies that an item is selected in the list and then removes the selected item from the list:

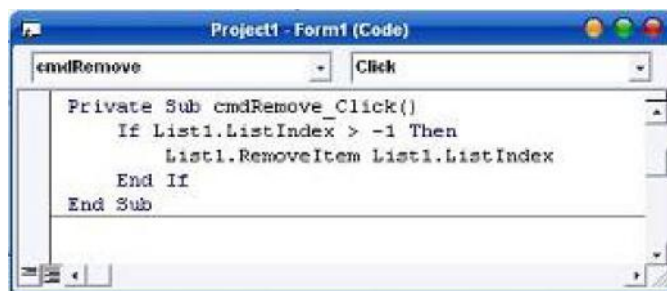


Fig. 6.3 Runtime Deletion of a Selected Item in a ListBox

To remove all entries in a list, use the `Clear` method.

```
List1.Clear
```

Sorting a List

You can specify the items to be added to a list alphabetically by setting the `Sorted` property to `True` and scraping the index. The sort is not case-sensitive; thus, the words 'Chandigarh' and 'chandigarh' are treated as same.

Accessing List Items in the List

The `List` property provides access to all items in the list. This property has an array in which each item in the list is an element of the array and each item is represented in a string form. To refer to an item in the list, use the following syntax:

```
<Listbox>.List(<index>)
```

The top item has an index of 0, the next has an index of 1, and so on. For example, the following statement displays the third item (index = 2) in a list in a text box.

```
txtCity.Text = List1.List(2)
```

To display only the selected item from the list in a text box, the following statement is used:

```
txtCity.text = List1.List(List1.Listindex)
```


Returning the Number of Items with the ListCount Property

For returning the number of items in a list box, use the `ListCount` property. The following statement, for example, uses the `ListCount` property for determining the number of entries in a list box:

```
Text1.Text = List1.ListCount
```

File System Controls

Three of the controls on the ToolBox let you access the computer's file system. They are `DriveListBox`, `DirListBox` and `FileListBox` controls (Refer Figure 1.7), which are the basic blocks for building dialog boxes that display the host computer's file system. Using these controls, a user can traverse the host computer's file system, locate any folder or files on any hard disk, even on network drives. The files controls are independent of each other, and each can exist on its own, but they are rarely used separately.

Figure 6.4 shows that three files controls are used in the design of forms that let the users explore the entire structure of their hard disks.

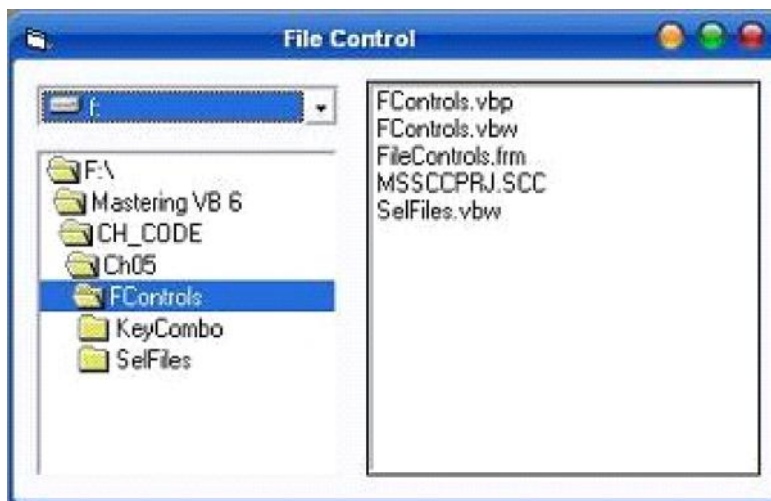


Fig. 6.4 Three Controls of File System

- **DriveListBox:** This displays the names of the drives within and connected to the PC. 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.
- **DirListBox:** This displays the folders of current Drive. The basic property of this control is the `Path` property, which is the name of the folder with subfolders displayed in the control.
- **FileListBox:** This displays the files of the current folder. The basic property of this control is also called `Path` and it is the path name of the folder whose files are displayed.

NOTES

NOTES

The three File controls are not tied to one another. If you place all three of them on a Form, you will see the names of all the folders under the current folder and so on. Each time you select a folder in the DirListBox by double-clicking its name, its subfolders are displayed. Similarly, the FileListBox control will display the names of all the files in the current folder. Selecting a drive in the DriveListBox control, however, does not affect the contents of the DirListBox.

To connect to the File controls, you must assign the appropriate values to the properties. To compel the DirListBox to display the folders of the selected drive in the DriveListBox, you must make sure that each time you select another drive, the Path property of the DirListBox control matches the Drive property of the DriveListBox.

The following is the minimum code you must place in the change event of the DriveListBox control:

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

Similarly, every time the current selection in the DirListBox control changes, you must set the FileListBox control's path property to point the new path of the DirListBox control:

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

This is all it takes to connect the three file controls.

Tab Order

TabIndex Property of Controls

Visual Basic makes use of the TabIndex property for determining the control that will be next in focus when a tab key is pressed. VB looks at the value of the TabIndex for the control that has focus, every time a tab key is pressed, and then it scans through the controls looking for the next highest TabIndex number. VB starts all over again with 0 when there are no more controls with higher TabIndex value. It looks for the first control with TabIndex of 0 or higher that can accept keyboard input.

VB, by default, assigns a tab order for controlling as we draw the controls on the Form, except for Menu, Data, Timer, Line, Image and Shape controls, which are not a part of tab order. Even though a TabIndex value is given, invisible or disabled controls also cannot receive the focus at runtime. In the development environment, setting the TabIndex property of controls is compulsory.

Visual Basic IDE Modes

A VB application works in either of the following three modes:

- Design Mode.
- Run Mode.
- Break/Suspended Mode.

While an application is being created or designed, it is in the *design mode*. When the application is executing, it is said to be in the *run mode* and while an application is in a state of *suspension*, it is said to be in the *break mode*.

Definition of Basic Terms

The following are the definitions of various VB terms which are used frequently.

Application

An application is an assortment of objects that work together for accomplishing something useful. The application in VB is called **Project**. A project could be calculation of mortgages, management of a video store, the payroll for 1000 employees or any required service.

Object

An object is a part of software with properties and functions that can be changed. A window is an **object** with **properties** like color, size, position on the screen, etc. The function of a window, also known as **methods**, can be manipulated to move it around, change the size, open it and close it. There is no need for writing the code for resizing a window. It can be done by clicking and dragging. The code can be written by anybody and put together in a small package called **window object**. The window object can be copied and pasted wherever it is needed by changing its properties for color or size. Its built-in methods can be used for opening and closing it or for resizing it whenever required. When an application is created using objects and are combined for producing results, it means the user is working in an **object oriented** environment.

Visual Basic Program Development Process

Generally, the following steps are required for building a VB application:

- Step 1:** Designing the Interface.
- Step 2:** Setting Properties of the Controls (Objects).
- Step 3:** Writing the Procedures of the Events.

Step 1: Designing the Interface

Visual Basic has its own IDE which can be used to design the Interface. The term Interface refers to GUI objects. These objects that are put on a Form are called **controls**. For getting a control, go to the **Toolbox**, then click on the control you

NOTES

NOTES

need, return to the Form and click and drag the control to the size and position you want. Position the controls as shown in the following screenshot:

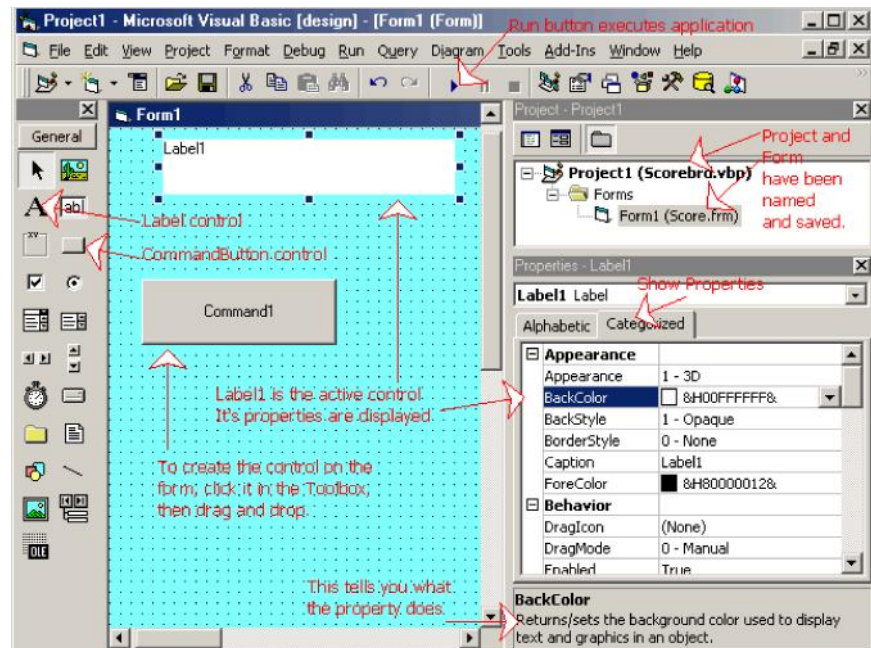


Fig. 6.5 Visual Basic IDE

We can make the bunch of controls on the form more attractive by changing the **Properties** of the controls in the **Properties window**. Each control has a whole series of properties. But right now, we only need the following ones:

- Alignment = How text aligns in the control.
- BackColor = The color of the background.
- Caption = The text that will appear in the control.
- Font = The font type and size.
- ForeColor = The color of the text (foreground).

Just as in all Windows applications, you can select multiple controls with Ctrl+Click for changing a property for all of them at once. For example, if there are all white backgrounds, select all controls, change ForeColor to white and all of them will be modified. Change the form to look like the one shown in the Figure 6.6. Note that there is no need to change the captions for Label4, Label5 and Label6 and the color of the buttons cannot be changed. The color of the buttons is what was earlier called 'IBM grey'. Remember to save your project as soon as you can.

NOTES

Step 2: Setting Properties of the Controls (Objects)

At this point, if you **Run** the application, your **Form** will appear just the way it was created. However, absolutely nothing happens if you click on any of the controls. Some **events** happen, i.e., the form opens, a button is clicked, etc. But, i.e., nothing tells the form what should be done when it sees an event. This is the reason that we have to write code, which is also known as **script**. Figure 6.6 illustrates the Design Mode.

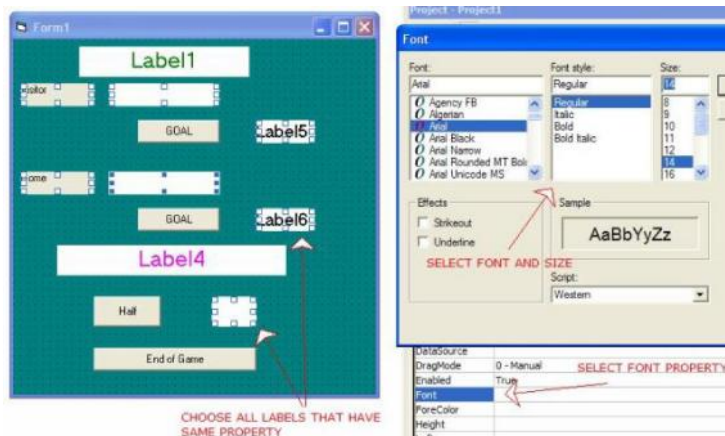


Fig. 6.6 Design Mode

Figure 6.7 illustrates the final display of the designed form.

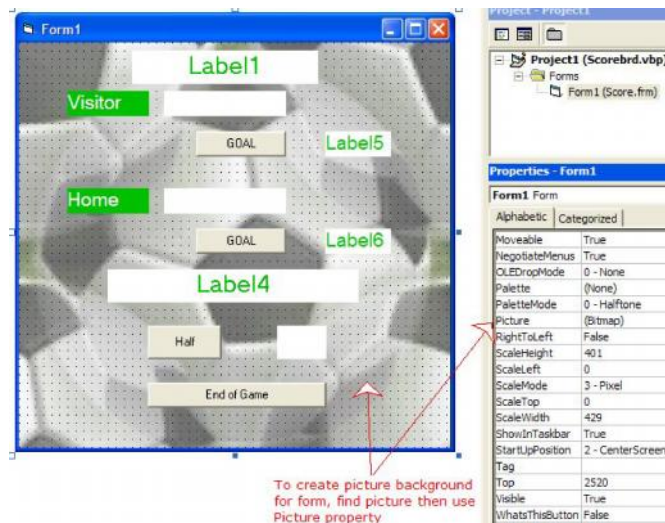


Fig. 6.7 Final Display

For switching between the Code window and the Form window, use the buttons just over the Project Explorer window (refer Figure 6.8 (a)).

Once you are in the Code window, you can see all the codes for the project or the code for one event at a time. Use the buttons in the lower left hand corner (refer Figure 6.8(b)).

NOTES

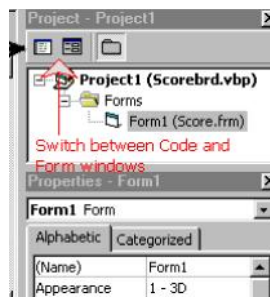


Fig. 6.8(a) Project Explorer Window

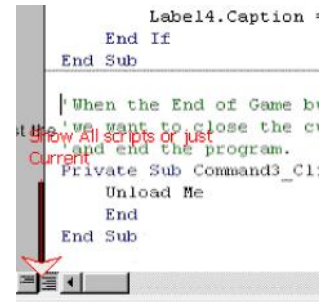


Fig. 6.8(b) Code Editor Window

For selecting the object and the event you want to code, use the two List boxes given at the top of the Code window where the left button is for the object and the right button is for the event. Start with **General ... Declarations** and then **Form ... Load**, etc.

Step 3: Write the Procedures of the Events

The user can write down the procedures of the event as shown in the following Figures 6.9 and 6.10.

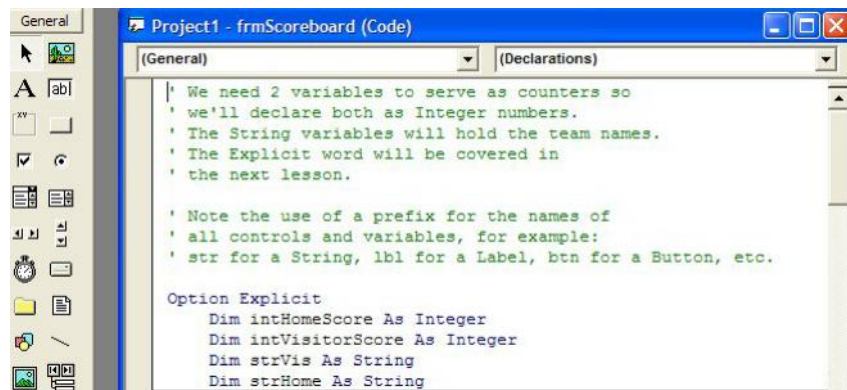


Fig. 6.9 Showing the Code

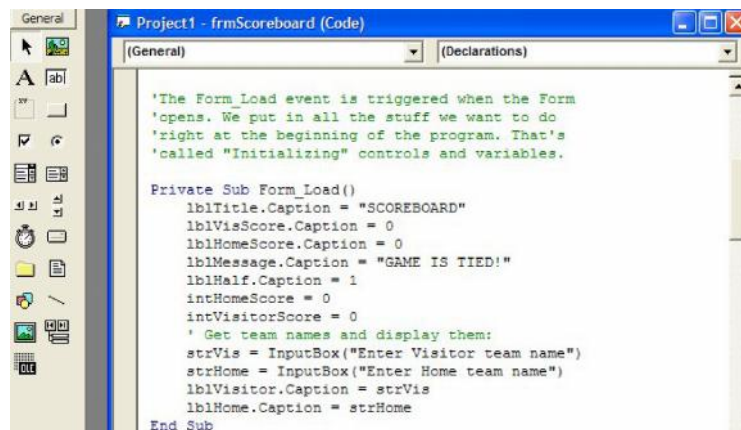


Fig. 6.10 Showing the Code of Form_Load Event

Now we can **Run** it and find something happening. When the Form loads, it will initialize the fields that were specified in the code.

ComboBox Control

A ComboBox control combines the features of a TextBox and a ListBox. This control allows the user to select an item either by typing text into the ComboBox or by selecting it from the list. The ComboBox control is similar to the ListBox control in the sense that it contains multiple item of which the user may select one, but it takes less space on the screen. The ComboBox control is practically an expandable ListBox control, which can grow when the user wants to make a selection and retracts after the selection is made. The real difference, however, between the ComboBox and ListBox controls is that the ComboBox control allows the user to specify items that do not exist in the list. Some common properties and methods of ComboBox control are shown in Table 6.10.

Table 6.10 Common Properties and Methods of ComboBox Control

Property/Method	Description
Properties	
Enabled	By setting this property to True or False, the user can decide whether the user can interact with this control or not.
Index	Specifies the control array index.
List	String array. Contains the strings displayed in the drop-down list. Starting array index is 0.
ListCount	Integer. Contains the number of drop-down list items.
ListIndex	Integer. Contains the index of the selected ComboBox item. If an item is not selected, ListIndex is -1.
Locked	Boolean. Specifies whether user can type or not in the ComboBox.
MousePointer	Integer. Specifies the shape of the mouse pointer when over the area of the ComboBox.
NewIndex	Integer. Index of the last item added to the ComboBox. If the ComboBox does not contain any items, NewIndex is -1.
Sorted	Boolean. Specifies whether the ComboBox's items are sorted or not.
Style	Integer. Specifies the style of the ComboBox's appearance.
TabStop	Boolean. Specifies whether ComboBox receives the focus or not.
Text	String. Specifies the selected item in the ComboBox.
ToolTipIndex	String. Specifies what text is displayed as the ComboBox's tool tip.
Visible	Boolean. Specifies whether the ComboBox is visible or not at run-time.
Methods	
AddItem	Adds an item to the ComboBox.
Clear	Removes all items from the ComboBox.
RemoveItem	Removes the specified item from the ComboBox.
SetFocus	Transfers focus to the ComboBox.
Event Procedures	
Change	Called when text in the ComboBox is changed.
DropDown	Called when the ComboBox drop-down list is displayed.
GotFocus	Called when the ComboBox receives the focus.
LostFocus	Called when the ComboBox loses it focus.

NOTES

NOTES

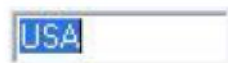
Note: As we have already discussed that the ComboBox control is similar to the ListBox control to some extent. From Table 6.10, you can observe that the property methods and events of both the controls (ListBox, ComboBox) are almost common. As adding, removing and accessing items in a ComboBox control is similar to the ListBox control. There is no need to explain each method again for the ComboBox control.

There are three ComboBox styles (refer Figure 6.11). Each style can be set at either design time or runtime and uses values or equivalent VB constant to set the style of the ComboBox.

Style	Value	Constant
Drop-down ComboBox	0	vbComboDrop-Down
Simple ComboBox	1	vbComboSimple
Drop-down ListBox	2	vbComboDrop-DownList



Drop-Down Combo



Simple Combo



Drop-Down List

Fig. 6.11 Three Different Styles of ComboBox Control

The simple ComboBox displays an edit area with an attached ListBox that 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 edit box 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 drop-down ComboBox first appears having only an edit area with a down arrow button at the right. The list portion stays hidden until the user dropdowns the list portion by clicking the down arrow button. The user can either select a value from the list or type a value in the edit area.

The drop-down list ComboBox changes the ComboBox into a dropdown ComboBox. At runtime, the control looks similar to the drop-down ComboBox. The user could click the down arrow to view the list. The difference between the drop-down combo and drop-down list combo is that the edit area in the drop-down list combo is disabled. The user can only select an item and cannot type anything in the edit area. Anyway, this area displays the selected item(s).

NOTES

Check Your Progress

1. What is TextBox?
2. For selecting the text which properties are used?
3. Explain about the Label control.
4. Which property is used to display the text in the Label control?
5. What do you understand by CommandButton control?
6. Which control is used for selecting one of several options?
7. For indicating whether a particular condition is on or off which control is used?
8. Define about the CheckBox control.
9. State about the different tasks of List Box control.

6.3 CONTROL ARRAY

A control array is a collection of more than one VB control that shares the same name, which is the name of the array. All the controls in an array must be same, such as an array of Labels. In a control array, each control has a unique index number and you can access the control by using these unique numbers only. By increasing the index number, you can access all the subsequent controls in an array.

You can create a control array in the following two ways:

- Set the Index property of a control during the design time. For example, if you set the index of a Label to 0, then it will be the first Label in the control array of labels.
- Create a control component on the design window. Now, click the control component and press Ctrl+C to copy that control. After copying the component, press Ctrl+V to paste that copied control. When you paste the component, the VB compiler asks to create a control array.

NOTES

Figure 6.12 shows Microsoft Visual Basic warning dialog box.

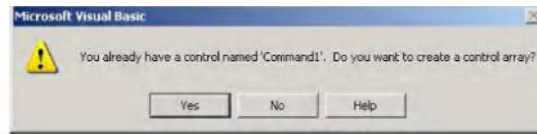


Fig. 6.12 Displaying Warning for Creating a Control Array

Now, click Yes to create a control array, else click No.

You can use the following index to access each element of the control array:

<Control array name> (array index)

You can also change the properties of the control array elements by using the following syntax:

<Control array name> (array index). <Property name> =
<property value>

Check Your Progress

10. Define the term Control array.
11. State about the Ways to create Control array.

6.4 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. A TextBox control is an edit field or edit control displaying information entered as design time by the user.
2. The text can be selected using the following three properties:
 - Ñ **SelLength:** Returns or sets the number of selected characters.
 - Ñ **SelStart:** Returns or sets the starting point of the text selected. *SelStart* indicates the position of the inserted point when no text is selected.
 - Ñ **SelText:** Returns or sets the string that contains the currently selected text. *SelText* will have a zero-length string if no text is selected.
3. Label is a graphical control which the user can use for displaying text that cannot be edited. A Label control displays text that cannot be changed. Basically, Labels are used for identifying controls (such as scrollbars and TextBox) that have no Caption property of their own.
4. Caption property is used to display the text in the Label control.
5. The CommandButton control allows the user to click on it to perform specific actions. When the user chooses the button, it carries out the appropriate action. It also looks as if it is being pushed in and released. The *Click event*

procedure is invoked whenever the user clicks a button. You can put code in the Click event procedure for performing any action.

6. OptionButton control is used for selecting one of several options.
7. For indicating whether a particular condition is on or off CheckBox control is used.
8. A CheckBox control is similar to an option button. CheckBox controls are used for offering a small set of choices from which one can choose one or more options. The Value property of both the controls is tested to check the current.
9. In order to properly work with the ListBox control within your application, you should be able to do the following tasks:
 - Add items to the list.
 - Remove items from the list.
 - Access individual items in the list.
10. A control array is a collection of more than one VB control that shares the same name, which is the name of the array.
11. You can create a control array in the following two ways:
 - Set the Index property of a control during the design time. For example, if you set the index of a Label to 0, then it will be the first Label in the control array of labels.
 - Create a control component on the design window. Now, click the control component and press Ctrl+C to copy that control. After copying the component, press Ctrl+V to paste that copied control. When you paste the component, the VB compiler asks to create a control array.

NOTES

6.5 SUMMARY

- The working environment in VB incorporates several different functions like editing, designing, compiling and debugging, within a common environment.
- Title Bar is the topmost bar displaying the title of the project. The window titled Project 1 is the name of project containing the project files.
- Form is the main feature of the VB application. It is the 'window' or 'screen' that users interact with. It can be considered as a 'canvas' on which the user places the objects that forms an application.
- Toolbox Window has a set of controls used for customizing forms.
- Properties Window helps in changing the property settings or characteristics of the form itself and also the elements of visual interface on the form.
- Project Explorer or Project Window shows the list of forms and modules in a project. A VB Project Explorer consists of a number of forms, modules and controls that make up an application.

NOTES

- Form Layout Window shows how big a form is in relation to the screen. It also displays the position of the window where it will be displayed when the project is run.
- Code Editor Window is the place for writing VB code for your application. By code we mean language statements, declarations and constants. For entering application code, the code editor window is used which serves as an editor.
- A control is an object that can be drawn on a Form object for enabling or enhancing user interaction with an application.
- A container control can hold other controls within it, for example, a *Frame* (there can be multiple controls inside a frame) or a *Picture Box* (it holds a picture) or simply your *Form* (you can put so many controls on it). Controls inside containers are known as child controls.
- In VB, the *Form* acts as the container for all the controls that form the interface. The *Form* is the top-level object in a VB application, and every application starts with the *Form*.
- The main characteristic of a Form is the title bar on which the Form's caption is displayed.
- Restores a maximized Form to its size before it was maximized; available only if the Form has been maximized.
- A *Frame* control is used to group various controls. A Frame control is a container control, i.e., it contains other controls in it; it does not carry out any job/action by itself, i.e., it does not respond to any event by itself.
- A *TextBox* control is an edit field or edit control, displaying information entered at design time by the user, or assigned to the control using code at runtime.
- It is a graphical control which the user can use for displaying text that cannot be edited. A *Label* control displays text that cannot be changed.
- The *CommandButton* control allows the user to click on it to perform specific actions.
- *CheckBox* controls are used for offering a small set of choices from which one can choose one or more options.
- A *ListBox* control displays a list of items from which one or more items can be displayed. A user gets a list of choices from the list box.
- The choices are displayed in a single column by default, though multiple columns can also be set up.
- Scroll bars will automatically appear on the control if the number of items is more than what can be displayed in the *ListBox*.
- To add items to a *ListBox*, use the *Additem* method. It is possible to populate the list at design time or runtime.

- To add items to a list at design time, click on the List property in the property box and then add the items.
- The AddItem method is used to add items to a list at runtime.
- The *ListBox* argument is a name of the ListBox, and the *item* argument is a string that represents the text to add to the list.
- The *index* argument specifies where the new item in the list is to be inserted. The first position is represented by an index of 0. The item is inserted at the end (or in the proper sorted order) if index is omitted.
- While list items are generally added in the Form_Load () event procedure, the AddItem method provides the ability of adding items to the list dynamically.
- For adding an item to a list at a particular position, an index value for the new item should be specified.
- The RemoveItem method is used to remove an item from a list.
- You can specify the items to be added to a list alphabetically by setting the Sorted property to true and scraping the index.
- The sort is not case-sensitive; thus, the words 'Chandigarh' and 'chandigarh' are treated as same.
- The List property provides access to all items in the list. This property has an array in which each item in the list is an element of the array and each item is represented in a string form.
- For returning the number of items in a list box, use the ListCount property.
- Three of the controls on the toolbox let you access the computer's file system.
- They are DriveListBox, DirListBox and FileListBox controls, which are the basic blocks for building dialog boxes that display the host computer's file system.
- Using these controls, a user can traverse the host computer's file system, locate any folder or files on any hard disk, even on network drives. The files controls are independent of each other, and each can exist on its own, but they are rarely used separately.
- This displays the names of the drives within and connected to the PC. 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.
- DirListBox displays the folders of current Drive. The basic property of this control is the Path property, which is the name of the folder with subfolders displayed in the control.
- FileListBox displays the files of the current folder. The basic property of this control is also called Path and it is the path name of the folder whose files are displayed.
- The three File controls are not tied to one another. If you place all three of them on a Form, you will see the names of all the folders under the current folder and so on.

NOTES

NOTES

- Each time you select a folder in the DirListBox by double-clicking its name, its subfolders are displayed. Similarly, the FileListBox control will display the names of all the files in the current folder. Selecting a drive in the DriveListBox control, however, does not affect the contents of the DirListBox.
- To connect to the File controls, you must assign the appropriate values to the properties.
- To compel the DirListBox to display the folders of the selected drive in the DriveListBox, you must make sure that each time you select another drive, the Path property of the DirListBox control matches the Drive property of the DriveListBox.
- Visual Basic makes use of the TabIndex property for determining the control that will be next in focus when a tab key is pressed.
- VB looks at the value of the TabIndex for the control that has focus, every time a tab key is pressed, and then it scans through the controls looking for the next highest TabIndex number.
- VB starts all over again with 0 when there are no more controls with higher TabIndex value. It looks for the first control with TabIndex of 0 or higher that can accept keyboard input.
- VB, by default, assigns a tab order for controlling as we draw the controls on the Form, except for Menu, Data, Timer, Line, Image and Shape controls, which are not a part of tab order. Even though a TabIndex value is given, invisible or disabled controls also cannot receive the focus at runtime.
- In the development environment, setting the TabIndex property of controls is compulsory.

6.6 KEY WORDS

- **Project:** A collection of several different types of files that make up a program.
- **Form:** The main feature of the VB application; it is the 'window' or 'screen' that users interact with.
- **Methods:** The public functions or procedures that are defined in a class.
- **Events:** Generate and invoke events for an object in VB.
- **DriveListBox:** This displays the names of the drives within and connected to the PC. 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.
- **DirListBox:** This displays the folders of current Drive. The basic property of this control is the Path property, which is the name of the folder with subfolders displayed in the control.

- **FileListBox:** This displays the files of the current folder. The basic property of this control is also called Path and it is the path name of the folder whose files are displayed.
- **Control array:** A collection of more than one VB control that shares the same name which is the name of the array.

NOTES

6.7 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What is the difference between Enabled property and visible property of a TextBox control?
2. Visual Basic a tool allows you to develop which type of applications.
3. How the ComboBox control is similar to the ListBox control?
4. Name at least two ways in which the user can trigger a command button.
5. Which property setting of TextBox would you use for entering a secret message?
6. Which control is used to provide an identifiable grouping for other controls?
7. What is the property that indicates whether or not a ListBox item is selected?
8. Which is a group of controls that share the same name and type.

Long-Answer Questions

1. Discuss ComboBox controls and its various methods and properties.
2. Explain control arrays. How array elements can be processed in VB program?
3. List out controls which does not have events.
4. Elaborate briefly on the ListBox Control. Explain all the methods of ListBox Control in detail.
5. Briefly discuss about the File system control, and its type.
6. Design an application to display traffic lights. The red and green lights should display for 12 seconds and the yellow light should be on for 5 seconds.
7. Indian Railways has introduced some more trains. Write an event procedure in VB to add the names of the new trains to the list as well as delete the trains that will be no longer running.
8. Design an application in VB to allow the user to set an alarm at scheduled time. Use the Beep statement to ring the alarm.
9. Design an application that accepts an item name from the user and adds it to a list box. Its screenshot being shown below:

NOTES



10. Design a VB application with the following interface:

First Name	<input type="text" value="Ramesh"/>
Last Name	<input type="text" value="Kumar"/>
Street Address	<input type="text" value="123"/>
City	<input type="text" value="dehradun"/>
State	<input type="text" value="UK"/>

Name :	<input type="text" value="Ramesh Kumar"/>
Address:	<input type="text" value="123, Dehradun, UK"/>

6.8 FURTHER READINGS

- Cornell, Gary. 1998. *Visual Basic 6 from the Ground Up*. New Delhi: Tata McGraw-Hill.
- Warner, Scott L. 1998. *Teach Yourself Visual Basic 6*. New Delhi: Tata McGraw-Hill.
- Jerke, Noel. 1999. *Visual Basic 6 - The Complete Reference*. New York: McGraw-Hill.
- Smith, Eric A., Valor Whisler and Hank Marquis. 1998. *Visual Basic 6 Programming Bible*. New Jersey: John Wiley & Sons, Inc.
- Azam, M. 2001. *Programming with Visual Basic*. New Delhi: Vikas Publishing House Pvt. Ltd.
- Manchanda, Mahesh. 2009. *Visual Programming*. New Delhi: Vikas Publishing House Pvt. Ltd.
- Balena, Francesco. 1999. *Programming Microsoft Visual Basic 6.0*. Bangalore: WP Publishers and Distributors (P) Ltd.
- Petroutsos, Evangelos. 1998. *Mastering Visual Basic 6, 1st Edition*. New Delhi: BPB Publications.
- Deitel, Harvey M., Paul J. Deitel and T. Tem R. Nieto. 1999. *Visual Basic 6: How to Program*. New Jersey: Prentice-Hall.
- Norton, Peter. 1998. *Peter Norton's Complete Guide to Visual Basic 6*. New Delhi: Techmedia.
- Reselman, Bob and Richard A. Peasley. 1998. *Using Visual Basic 6*. New Jersey: Pearson Education (Que Publishing).
- Donald, Bob and Oancea Gabriel. 1999. *Visual Basic 6 from Scratch*. New Delhi: Prentice-Hall of India.

BLOCK III
VISUAL BASIC PROCEDURES,
FUNCTIONS AND ARRAYS

*Creating Procedures
and Functions*

NOTES

**UNIT 7 CREATING PROCEDURES
AND FUNCTIONS**

- 7.0 Introduction
- 7.1 Objectives
- 7.2 Creating Procedures and Functions
- 7.3 Answers to Check Your Progress Questions
- 7.4 Summary
- 7.5 Key Words
- 7.6 Self-Assessment Questions and Exercises
- 7.7 Further Readings

7.0 INTRODUCTION

A function procedure is a series of Visual Basic (VB) statements enclosed by the function and End Function statements. The function procedure performs a task and then returns control to the calling code. You can define a Function procedure in a module, class, or structure. It is Public by default, which means you can call it from anywhere in your application that has access to the module, class, or structure in which you defined it. A Function procedure can take arguments, such as constants, variables, or expressions, which are passed to it by the calling code.

When function procedure it returns control, it also returns a value to the calling code. Each time the procedure is called, its statements run, starting with the first executable statement after the Function statement and ending with the first End Function, Exit Function, or Return statement encountered.

Numeric functions are used to calculate numerical expressions.

String functions are used to manipulate one or more string arguments and after manipulation it returns a string value.

Date and Time functions are used to manipulate the date and time values.

In computer science, recursion is a method of solving a problem where the solution depends on solutions to smaller instances of the same problem.

In this unit, you will study about the procedures, functions, string function, numeric function, date and time function and recursive function.

NOTES

7.1 OBJECTIVES

After going through this unit, you will be able to:

- Understand the significance of functions in VB
- Elaborate on the library function
- Discuss about the Numeric function, String function, Date and Time function
- Able to handle common events of Visual Basic controls
- Differentiate between procedure and function
- Explain about the recursive function

7.2 CREATING PROCEDURES AND FUNCTIONS

Library functions are the built-in functions, which are defined in the VB (Visual Basic) library. Each library function performs a specific operation. You can use these functions directly, without implementing in a programming to perform various functions, such as manipulating strings and numbers. Depending on the functioning, the library functions can be categorized into the following categories:

- Numeric Functions.
- String Functions.
- Date and Time Functions.

Numeric Functions

Numeric functions are used to calculate numerical expressions. Table 7.1 lists the commonly used built-in numeric functions.

Table 7.1 Numeric Library Functions

Numeric Functions	Description
Abs (argument)	Returns the absolute value of the argument. The absolute value is the positive equivalent of the argument. For example, the absolute value for both Abs (78) and Abs (-78) will be 78.
Sqr (argument)	Returns the square root of the argument. For example, the value of the sqr (6) will be 36.
Cos (argument)	Returns the cosine value of the argument and this cosine value is expressed in radians.

For example, the steps for implementing the Sqr numerical function in VB are as follows:

1. Open the Code window.
2. Click the down arrow button next to the Object drop-down list box and select the Form control.

- Click the down arrow button next to the Procedure drop-down list box and select the Load event.

- Enter the following lines of code in the Code window.

```
Private Sub Form_Load()  
Dim x As Integer  
x = 34  
MsgBox (Sqr(x)) ' displays square root of the value 34  
End Sub
```

Figure 7.1 shows the Code window for the Sqr numerical function.

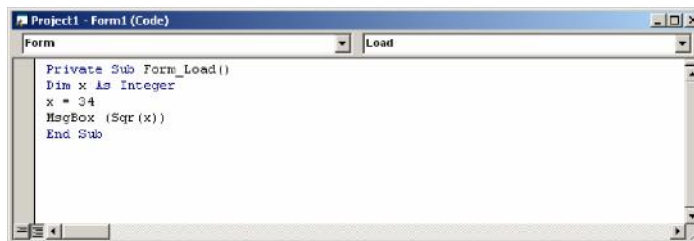


Fig. 7.1 Displaying the Code for Implementing the Sqr Numerical Function

- Press F5 to compile the above code. Figure 7.2 shows the output after compiling the Sqr numerical function.

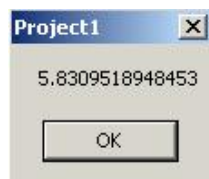


Fig. 7.2 Output for the Sqr Numerical Function

String Functions

String functions are used to manipulate one or more string arguments and after manipulation it returns a string value. Table 7.2 lists the commonly used built-in string functions.

Table 7.2 String Library Functions

String Functions	Description
UCase(argument)	Returns all characters of the argument in upper case. In the UCase(argument) function, the argument represents a string value.
LCase(argument)	Returns all characters of the argument in lower case. In the LCase(argument) function, the argument represents a string value.

NOTES

NOTES

Left(argument, int)	<p>Returns a specific number of characters from the leftmost portion of the string. In the Left (argument, no of characters), argument represents a string and the int parameter specifies the number of characters you want to extract from the string. Consider the following listing:</p> <pre>Dim value As String value = "abcdefgh" MsgBox (Left(Pname, 2))</pre> <p>The above listing will print the first two characters from the leftmost portion of the string value. As a result, the output will be ab.</p>
Right(argument, int)	<p>Returns a specific number of characters from the rightmost portion of the string. In the Right (argument, no of characters), argument represents a string and the int parameter specifies the number of characters you want to extract from the string. Consider the following listing:</p> <pre>Dim value As String value = "abcdefgh" MsgBox (Right(Pname, 2))</pre> <p>The above listing will print the first two characters from the rightmost portion of the string value. As a result, the output will be gh.</p>
Len (argument)	<p>Returns the total number of characters in a string. In Len (argument), argument represents the string. Consider the following listing:</p> <pre>Dim value As String value = "abcdefgh" MsgBox (Len(value))</pre> <p>The above listing will print 8, which is the length of the string.</p>
StrReverse (argument)	<p>Returns the characters of a string in reverse order. In StrReverse (argument), argument represents the string. Consider the following listing:</p> <pre>Dim value As String value = "abcdefgh" MsgBox (StrReverse (value))</pre>

The above listing will print all the characters of the string in reverse order. As a result, the output will be hgfedcba.

InStr (argument, search argument)	<p>Returns the location of a specific alphabet or word in a string. In InStr (argument, search argument), argument specifies the string, search argument specifies the alphabet or word that you want to search and startpos specifies the starting position for the search. Consider the following listing:</p> <pre>MsgBox (InStr ("I am a good boy" , "m"))</pre> <p>For the above listing, the m alphabet is at position 4 in the string "I am a good boy". Therefore, the output of the above listing will be 4.</p>
-----------------------------------	---

NOTES

For example, the following steps show how to use the Len function in VB:

1. Open the Code window.
2. Click the down arrow button next to the Object drop-down list box and select the Form control.
3. Click the down arrow button next to the Procedure drop-down list box and select the Load event.
4. Enter the following lines of code in the Code window:

```
Private Sub Form_Load()
Dim yourname As String
yourname = "John Kumar"
MsgBox Len(yourname)
End Sub
```

Figure 7.3 shows the Code window for the Len function.

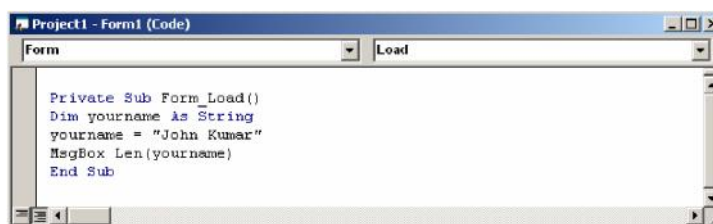


Fig. 7.3 Displaying the Code for Implementing the Len Function

5. Press F5 to compile the above code. Figure 7.4 shows the output of the Len function after compilation.

NOTES



Fig. 7.4 Output for Length of the yourname String

Consider another example that uses the UCase function. The following steps show how to use the UCase function in VB:

1. Open the Code window.
2. Click the down arrow button next to the Object drop-down list box and select the Form control.
3. Click the down arrow button next to the Procedure drop-down list box and select the Load event.
4. Enter the following lines of code in the Code window:

```
Private Sub Form_Load()  
Dim value As String  
value = "abcdefgh"  
MsgBox UCase(value)  
End sub
```

Figure 7.5 shows the Code window for the UCase function.

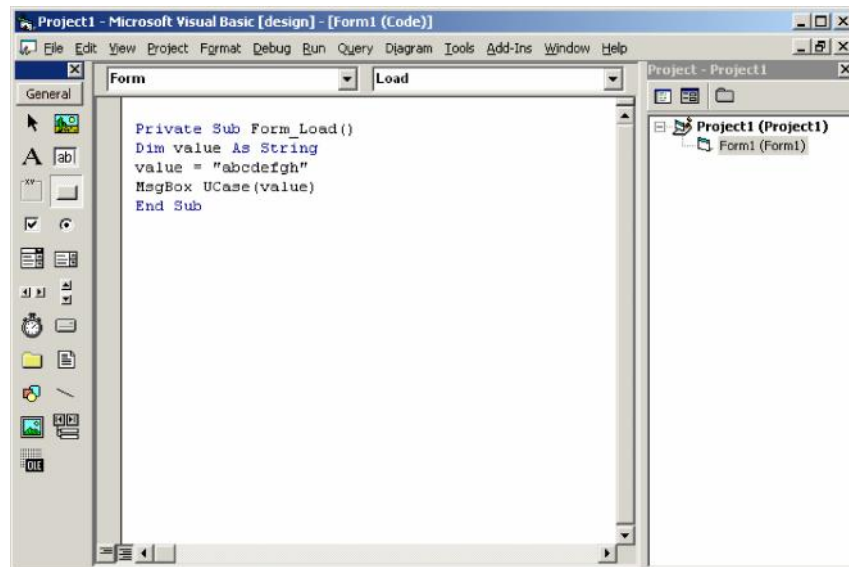


Fig. 7.5 Displaying the Code for Implementing the UCase Function

5. Press F5 to compile the above code. Figure 7.6 shows the output of the UCase function after compilation.



Fig. 7.6 Output for the UCase Functions

NOTES

Date and Time Functions

Date and Time functions are used to manipulate the date and time values. Table 7.3 lists the commonly used built-in Date and Time functions.

Table 7.3 String library functions

Date and Time Functions	Description
Date	Returns the current date.
Time	Returns the current time.
TimeSerial(hour, min, sec)	Returns the current date and time in the internal date format.
Now	Returns the current date and time.

For example, the steps for implementing the Now function in VB are:

1. Open the Code window.
2. Click the down arrow button next to the Object drop-down list box and select the Form control.
3. Click the down arrow button next to the Procedure drop-down list box and select the Load event.
4. Enter the following lines of code in the Code window.

```
Private Sub Form_Load()  
MsgBox (Now)  
End Sub
```

Figure 7.7 shows the Code window for the Now function.

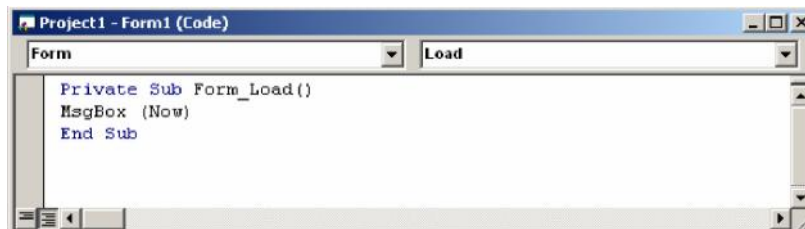


Fig. 7.7 Displaying the Code for Implementing the Function

5. Press F5 to compile the above code. Figure 7.8 shows the output after compiling the `Now` function.

NOTES



Fig. 7.8 Output for the `Now` Function

Common Methods of Visual Basic Controls

Methods are blocks of code that are designed into a control and let the control know how to do things like move to another location on a Form. Just like properties, all controls do not have the same methods, although there are some common methods, as shown in Table 7.4.

Table 7.4 Common Methods of VB Control

<i>Method</i>	<i>Description</i>
Move	Changes the position of an object in response to a code request.
Drag	Handles the execution of a drag and drop operation by the user.
SetFocus	Gives focus to the specified object in the method call.
ZOrder	Determines the order in which multiple objects appear onscreen.

Common Events of Visual Basic Controls

Events are what occur in and around a program. When, for example, a user clicks a button, several events happen. The mouse button is pressed, the `CommandButton` in the program is clicked and then the mouse button is released. These three things correspond to the `MouseDown` event, the `Click` event and the `MouseUp` event. During this process, the `GotFocus` event for the `CommandButton` and the `LostFocus` event for the object previously held also happen.

Again, all controls do not have the same events, but there are some events that are shared by several controls. These events are a result of some particular user action like clicking a `TextBox`, moving the mouse or pressing a key on the keyboard. These types of events are initiated by the user. Events common to most VB controls are described in Table 7.5.

Table 7.5 Common Events of VB Controls

<i>Event</i>	<i>Occurs When ...</i>
Change	The user modifies the text in a text box or a combo box.
Click	The user clicks the main mouse button on an object.
DbClick	The user double-clicks the main mouse button on an object.
DragDrop	The user drags an object to a different location.
DragOver	The user drags an object over another control.
GotFocus	An object receives focus.
KeyDown	The user presses a key on the keyboard while an object has focus. It reports the exact physical state of the keyboard itself.
KeyPress	The user presses and releases a key on the keyboard while an object has focus. It simply returns the character that the key represents.
KeyUp	The user releases a key on the keyboard while an object has focus. It reports the exact physical state of the keyboard itself.
LostFocus	An object loses focus.
MouseDown	The user presses a mouse button while the mouse pointer is over an object.
MouseMove	The user moves the mouse pointer over an object.
MouseUp	The user releases a mouse button while the mouse pointer is over an object.

NOTES**Distinction Between Procedures and Functions**

Procedures and functions are used to create modular programs. Visual Basic statements are grouped in a block enclosed by Sub, Function and matching End statements. The difference between the procedures and functions is that functions return values whereas procedures do not return values.

A procedure is a block of Visual Basic statements inside Sub, End Sub statements. Procedures do not return values.

```
Sub Main()
    SimpleProcedure()
End Sub
Sub SimpleProcedure()
    Console.WriteLine("Simple procedure")
End Sub
```

This example shows basic usage of procedures. In the above programs, we have two procedures. The Main() procedure and the user defined SimpleProcedure(). The Main() procedure is the entry point of a Visual Basic program.

```
SimpleProcedure()
```

Each procedure has a name. Inside the Main() procedure, we **call** our user defined SimpleProcedure() procedure.

```
Sub SimpleProcedure()
    Console.WriteLine("Simple procedure")
End Sub
```

NOTES

Procedures are defined outside the `Main()` procedure. Procedures name follows the `Sub` statement. When we call a procedure inside the Visual Basic program, the control is given to that procedure. Statements inside the block of the procedure are executed.

Procedures can take optional parameters.

```
Sub Main()  
Dim x As Integer = 55  
Dim y As Integer = 32  
Addition(x, y)  
End Sub  
Sub Addition(ByVal k As Integer, ByVal l As Integer)  
Console.WriteLine(k+l)  
End Sub
```

In the above example, we pass some values to the `Addition()` procedure.

```
Addition(x, y)
```

Here we call the `Addition()` procedure and pass two parameters to it. These parameters are two integer values.

```
Sub Addition(ByVal k As Integer, ByVal l As Integer)  
Console.WriteLine(k+l)  
End Sub
```

We define a **procedure signature**. A procedure signature is a way of describing the parameters and parameter types with which a legal call to the function can be made. It contains the name of the procedure, its parameters and their type and in case of functions also the return value. The `ByVal` keyword specifies how we pass the values to the procedure. In our case, the procedure obtains two numerical values, 55 and 32. These numbers are added and the result is printed to the console.

A function is a block of Visual Basic statements inside `Function`, `End Function` statements. Functions return values.

There are two basic types of functions. Built-in functions and user defined ones. The built-in functions are part of the Visual Basic language. There are various mathematical, string or conversion functions.

```
Sub Main()  
Console.WriteLine(Math.Abs(-23))  
Console.WriteLine(Math.Round(34.56))  
Console.WriteLine("ZetCode has {0} characters", _  
("ZetCode"))  
End Sub
```

In the preceding example, we use two math functions and one string function. Built-in functions help programmers do some common tasks.

In the following example, we have a user defined function.

```
Dim x As Integer = 55
Dim y As Integer = 32
Dim result As Integer
Sub Main()
    result = Addition(x, y)
    Console.WriteLine(Addition(x, y))
End Sub
Function Addition(ByVal k As Integer, _
    ByVal l As Integer) As Integer
    Return k+l
End Function
```

Two values are passed to the function. We add these two values and return the result to the Main() function.

```
result = Addition(x, y)
```

Addition function is called. The function returns a result and this result are assigned to the result variable.

```
Function Addition (ByVal k As Integer,ByVal l As Integer)
As
Integer
Return k+l
End Function
```

Recursive Function

Recursion occurs when a method calls itself. Recursive functions need special stop conditions. Otherwise they will infinitely continue calling themselves. In general, this is not the most effective way to write Visual Basic code.

The following procedure uses recursion to calculate the factorial of its original argument.

```
Function Factorial (n as Integer) As Integer
    If n <= 1 Then
        Return 1
    End If
    Return Factorial (n - 1) * n
End Function
```

NOTES

NOTES

In the recursive function, every time that a recursive call is made, the program branches to the new function call. Since the program was in the middle of executing another call to the same function, it needs to store the necessary information to allow it to return to the point where it branched. This information is stored in a data structure called a **stack** and is placed on the top of the stack. When the program has finished executing the new function, it returns to the stack, finds the values it was working with before and completes the unfinished function. In this function, there are lots of branches to place in the stack before returning and this slows the program down more than you might think.

The following program code contains VB implementations of the two functions described above. It uses the **Stopwatch** class to time the execution of each algorithm.

```
Sub Main()  
    Dim numToFind As Integer  
    Console.Write("Enter the term you want to find ")  
    numToFind = Console.ReadLine()  
    Dim stopW As Stopwatch = New Stopwatch()  
    Dim term As Long  
    stopW.Reset()  
    stopW.Start()  
    term = IterativeFibonacci(numToFind)  
    stopW.Stop()  
    Console.WriteLine("Term no {0} = {1} Iterative: {2}  
Milliseconds", numToFind, term, stopW.ElapsedMilliseconds)  
    stopW.Reset()  
    stopW.Start()  
    term = RecursiveFibonacci(numToFind)  
    stopW.Stop()  
    Console.WriteLine("Term no {0} = {1} Recursive: {2}  
Milliseconds", numToFind, term, stopW.ElapsedMilliseconds)  
    Console.ReadLine()  
End Sub  
  
Function RecursiveFibonacci(ByVal a As Integer) As Long  
    If a = 0 Then Return 0  
    If a = 1 Then Return 1  
    Return RecursiveFibonacci(a - 1) + RecursiveFibonacci(a  
- 2)  
End Function  
  
Function IterativeFibonacci(ByVal a As Integer) As Long  
    If a = 0 Then Return 0
```

```

If a = 1 Then Return 1
Dim fibNums(a + 1) As Long
fibNums(0) = 0
fibNums(1) = 1
For count = 2 To a
    fibNums(count) = fibNums(count - 1) + fibNums(count
- 2)
Next
Return fibNums(a)
End Function

```

NOTES

Considerations with Recursive Procedures

Limiting Conditions. You must design a recursive procedure to test for at least one condition that can terminate the recursion, and you must also handle the case where no such condition is satisfied within a reasonable number of recursive calls. Without at least one condition that can be met without fail, your procedure runs a high risk of executing in an infinite loop.

Memory Usage. Your application has a limited amount of space for local variables. Each time a procedure calls itself, it uses more of that space for additional copies of its local variables. If this process continues indefinitely, it eventually causes a `StackOverflowException` error.

Efficiency. You can almost always substitute a loop for recursion. A loop does not have the overhead of passing arguments, initializing additional storage, and returning values. Your performance can be much better without recursive calls.

Mutual Recursion. You might observe very poor performance, or even an infinite loop, if two procedures call each other. Such a design presents the same problems as a single recursive procedure, but can be harder to detect and debug.

Calling with Parentheses. When a Function procedure calls itself recursively, you must follow the procedure name with parentheses, even if there is no argument list. Otherwise, the function name is taken as representing the return value of the function.

Testing. If you write a recursive procedure, you should test it very carefully to make sure it always meets some limiting condition. You should also ensure that you cannot run out of memory due to having too many recursive calls.

NOTES

Check Your Progress

1. Explain the term library functions.
2. Why we use numeric functions in Visual Basic?
3. What is the use of string functions?
4. Explain about the Date and time functions.
5. Give the definition of method in Visual Basic.
6. What is the use of KeyPress event of Visual Basic Control?
7. Differentiate between procedures and functions.
8. Define the term procedure signature.
9. Elaborate on the term recursive function.
10. What is mutual recursive?

7.3 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Library functions are the built-in functions, which are defined in the VB (Visual Basic) library. Each library function performs a specific operation. You can use these functions directly, without implementing in a programming to perform various functions, such as manipulating strings and numbers.
2. Numeric functions are used to calculate numerical expressions.
3. String functions are used to manipulate one or more string arguments and after manipulation it returns a string value.
4. Date and Time functions are used to manipulate the date and time values.
5. Methods are blocks of code that are designed into a control and let the control know how to do things like move to another location on a Form. Just like properties, all controls do not have the same methods, although there are some common methods.
6. The user presses and releases a key on the keyboard while an object has focus. It simply returns the character that the key represents.
7. Procedures and functions are used to create modular programs. Visual Basic statements are grouped in a block enclosed by Sub, Function and matching End statements. The difference between the procedures and functions is that functions return values whereas procedures do not return values.
8. We define a procedure signature. A procedure signature is a way of describing the parameters and parameter types with which a legal call to the function can be made. It contains the name of the procedure, its parameters and their type and in case of functions also the return value.

9. Recursion occurs when a method calls itself. Recursive functions need special stop conditions. Otherwise they will infinitely continue calling themselves.
10. In computer science, mutual recursion is a form of recursion where two mathematical or computational objects, such as functions or data types, are defined in terms of each other.

NOTES

7.4 SUMMARY

- Library functions are the built-in functions, which are defined in the VB library.
- Library functions can be used directly, without implementing in a programming to perform various functions, such as manipulating strings and numbers.
- Numeric functions are used to calculate numerical expressions.
- String functions are used to manipulate one or more string arguments and after manipulation it returns a string value.
- Date and Time functions are used to manipulate the date and time values.
- Methods are blocks of code that are designed into a control and let the control know how to do things like move to another location on a Form.
- Just like properties, all controls do not have the same methods, although there are some common methods.
- Events are what occur in and around a program. When, for example, a user clicks a button, several events happen.
- The mouse button is pressed, the `CommandButton` in the program is clicked and then the mouse button is released. These three things correspond to the `MouseDown` event, the `Click` event and the `MouseUp` event.
- During this process, the `GotFocus` event for the `CommandButton` and the `LostFocus` event for the object previously held also happen.
- These events are a result of some particular user action like clicking a `TextBox`, moving the mouse or pressing a key on the keyboard.
- Procedures and functions are used to create modular programs.
- Visual Basic statements are grouped in a block enclosed by `Sub`, `Function` and matching `End` statements.
- The difference between the procedures and functions is that functions return values whereas procedures do not return values.
- A procedure is a block of Visual Basic statements inside `Sub`, `End Sub` statements.
- Procedures are defined outside the `Main ()` procedure. Procedures name follows the `Sub` statement.

NOTES

- When we call a procedure inside the Visual Basic program, the control is given to that procedure. Statements inside the block of the procedure are executed.
- We define a procedure signature. A procedure signature is a way of describing the parameters and parameter types with which a legal call to the function can be made.
- Procedure signature contains the name of the procedure, its parameters and their type and in case of functions also the return value. The ByVal keyword specifies how we pass the values to the procedure. A function is a block of Visual Basic statements inside `Function`, `End Function` statements. Functions return values.
- There are two basic types of functions. Built-in functions and user defined ones. The built-in functions are part of the Visual Basic language. There are various mathematical, string or conversion functions.
- Recursion occurs when a method calls itself. Recursive functions need special stop conditions. Otherwise they will infinitely continue calling themselves.
- In the recursive function, every time that a recursive call is made, the program branches to the new function call.
- Since the program was in the middle of executing another call to the same function, it needs to store the necessary information to allow it to return to the point where it branched.
- This information is stored in a data structure called a **stack** and is placed on the top of the stack.
- You must design a recursive procedure to test for at least one condition that can terminate the recursion, and you must also handle the case where no such condition is satisfied within a reasonable number of recursive calls.
- Memory usage application has a limited amount of space for local variables.
- Each time a procedure calls itself, it uses more of that space for additional copies of its local variables. If this process continues indefinitely, it eventually causes a `StackOverflowException` error.
- Efficiency can almost always substitute a loop for recursion. A loop does not have the overhead of passing arguments, initializing additional storage, and returning values. Your performance can be much better without recursive calls.
- Mutual recursion might observe very poor performance, or even an infinite loop, if two procedures call each other.
- Such a design presents the same problems as a single recursive procedure, but can be harder to detect and debug.
- Calling with parentheses `function procedure` calls itself recursively, you must follow the procedure name with parentheses, even if there is no argument

list. Otherwise, the function name is taken as representing the return value of the function.

- Testing write a recursive procedure, you should test it very carefully to make sure it always meets some limiting condition.
- You should also ensure that you cannot run out of memory due to having too many recursive calls.

NOTES

7.5 KEY WORDS

- **Library functions:** The built-in functions which are defined in the Visual Basic (VB) library.
- **Numeric functions:** Numeric functions are used to calculate numerical expressions.
- **String function:** String functions are used to manipulate one or more string arguments and after manipulation it returns a string value.
- **Date and time functions:** Date and Time functions are used to manipulate the date and time values.
- **Recursion:** In computer science, recursion is a method of solving a problem where the solution depends on solutions to smaller instances of the same problem.

7.6 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What is a function? How is it declared and called?
2. Give the definition of library function.
3. Explain the term numeric functions.
4. Define the term string functions.
5. Explain the date and time functions.
6. What are events? Give an example.
7. Explain the term procedure signature.
8. What are limiting conditions for recursive procedures?

Long-Answer Questions

1. Discuss about library function and its types with the help of examples.
2. Briefly discuss the difference between numeric and string functions giving appropriate examples.

NOTES

3. Briefly explain about the different methods available for string manipulation in Visual Basic giving syntax and example codes.
4. Elaborate briefly to add, remove and replace a string using in-built function.
5. Briefly discuss the use of procedures with the help of examples.
6. Differentiate between common method and common events giving examples.
7. Distinction between procedures and functions with the help of examples.
8. Describe the recursive function and considerations of recursive procedures. Write the program for recursive function.

7.7 FURTHER READINGS

- Cornell, Gary. 1998. *Visual Basic 6 from the Ground Up*. New Delhi: Tata McGraw-Hill.
- Warner, Scott L. 1998. *Teach Yourself Visual Basic 6*. New Delhi: Tata McGraw-Hill.
- Jerke, Noel. 1999. *Visual Basic 6 - The Complete Reference*. New York: McGraw-Hill.
- Smith, Eric A., Valor Whisler and Hank Marquis. 1998. *Visual Basic 6 Programming Bible*. New Jersey: John Wiley & Sons, Inc.
- Azam, M. 2001. *Programming with Visual Basic*. New Delhi: Vikas Publishing House Pvt. Ltd.
- Manchanda, Mahesh. 2009. *Visual Programming*. New Delhi: Vikas Publishing House Pvt. Ltd.
- Balena, Francesco. 1999. *Programming Microsoft Visual Basic 6.0*. Bangalore: WP Publishers and Distributors (P) Ltd.
- Petroutsos, Evangelos. 1998. *Mastering Visual Basic 6, 1st Edition*. New Delhi: BPB Publications.
- Deitel, Harvey M., Paul J. Deitel and T. Tem R. Nieto. 1999. *Visual Basic 6: How to Program*. New Jersey: Prentice-Hall.
- Norton, Peter. 1998. *Peter Norton's Complete Guide to Visual Basic 6*. New Delhi: Techmedia.
- Reselman, Bob and Richard A. Peasley. 1998. *Using Visual Basic 6*. New Jersey: Pearson Education (Que Publishing).
- Donald, Bob and Oancea Gabriel. 1999. *Visual Basic 6 from Scratch*. New Delhi: Prentice-Hall of India.

UNIT 8 MULTIPLE FORMS

- 8.0 Introduction
- 8.1 Objectives
- 8.2 Startup Forms
- 8.3 Submain Procedure
- 8.4 Answers to Check Your Progress Questions
- 8.5 Summary
- 8.6 Key Words
- 8.7 Self-Assessment Questions and Exercises
- 8.8 Further Readings

NOTES

8.0 INTRODUCTION

The Windows Form is a vital component in the development of any Windows based application. Forms essentially provide the windows that make up a Windows application. In fact, the terms window and form are often used interchangeably. Forms allow the Visual Basic developer to create windows and layout controls (such as, buttons, labels, etc.) in those forms to provide the application's user interface.

The main characteristic of a Form is the title bar on which the Form's caption is displayed. On the left end of the title bar is the Control Menu icon. Clicking this icon opens the Control Menu. Maximize, Minimize and Close buttons can be found on the right side of the Form. Clicking on these buttons performs the associated function. 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.

A procedure is a block of Visual Basic statements enclosed by a declaration statement (Function, Sub, Operator, Get, Set) and a matching End declaration. All executable statements in Visual Basic must be within some procedure. A Sub procedure is a series of Visual Basic statements enclosed by the Sub and End Sub statements. The Sub procedure performs a task and then returns control to the calling code, but it does not return a value to the calling code.

Each time the procedure is called, its statements are executed, starting with the first executable statement after the Sub statement and ending with the first End Sub, Exit Sub, or Return statement encountered.

Sub procedure can be define in modules, classes, and structures. By default, it is Public, which means you can call it from anywhere in your application that has access to the module, class, or structure in which you defined it. The term *method* describes a Sub or Function procedure that is accessed from outside its defining module, class, or structure.

In this unit, you will study about the multiple form, startup forms and submain procedure.

NOTES

8.1 OBJECTIVES

After going through this unit, you will be able to:

- Describe the basic concept of form
 - Explain the appearance of forms
 - Understand the common properties of form control
 - Elaborate on the form related methods
 - Explain about the procedure and submain procedure
-

8.2 STARTUP FORMS

Windows Form is a key element of any Windows-based application development. The windows which make up a Windows application are essentially provided by Forms. The terms window and shape are actually also used interchangeably. Forms allow the developer of Visual Basic to build windows and layout controls in those forms (such as, buttons, labels, etc.) to provide the user interface of the application.

The container for all the controls that make up the user interface is Visual Basic Form. Each window you see in a simple visual application is a shape, so the form and window terms represent the same entity. When you create a Windows Forms Program, Visual Studio creates a default form for you. 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.

A `control` is an object that can be drawn on a Form object to enable or enhance the user interaction with an application. Controls have properties that define the aspects of their appearance, such as position, size and color, and the aspects of their behaviour, such as their response to the user input. They can respond to events initiated by the user or set off by the system. For instance, a code could be written in a `CommandButton` control's click event procedure that would load a file or display a result.

The following are some important points about setting up the properties:

- 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 calculator program, all the captions of the command buttons, such as `+`, `-`, `MC` and `MR` are commonly found in an ordinary calculator and a user should have no problem in manipulating the buttons.
- A lot of programmers like to use a meaningful name for the Name

property; may be, because it is easier for them to write and read the event procedure, and debug or modify the programs later. However, it is not a must to do as long as you label your objects clearly and use comments in the program whenever you feel necessary.

- One more important property is whether the control is enabled or not.
- Finally, you must also consider making the control visible or invisible at run-time or decide when it should become visible or invisible.

Creating a Form

A container control can hold other controls with in it, e.g., a Frame (there can be multiple controls inside a frame) or a Picture Box (it holds a picture) or simply your form (you can put so many controls on it). Controls inside containers are known as **child controls**. Child controls can exist completely inside their containers. It means that you cannot move them outside their container and if you try to drag them beyond the boundary of their container, a part of the control gets hidden. When you delete a container control, all its child controls automatically get deleted.

Form Control

In VB (Visual Basic), the Form acts as the container for all the controls that form the interface. The Form is the top-level object in a VB application, and every application starts with the Form.

Appearance of Forms

The main characteristic of a Form is the title bar on which the Form's caption is displayed. On the left end of the title bar is the Control Menu icon. Clicking this icon opens the control menu. Maximize, Minimize and Close buttons can be found on the right side of the Form. Clicking on these buttons performs the associated function.

Figure 8.1 illustrates the appearance of a container control form.

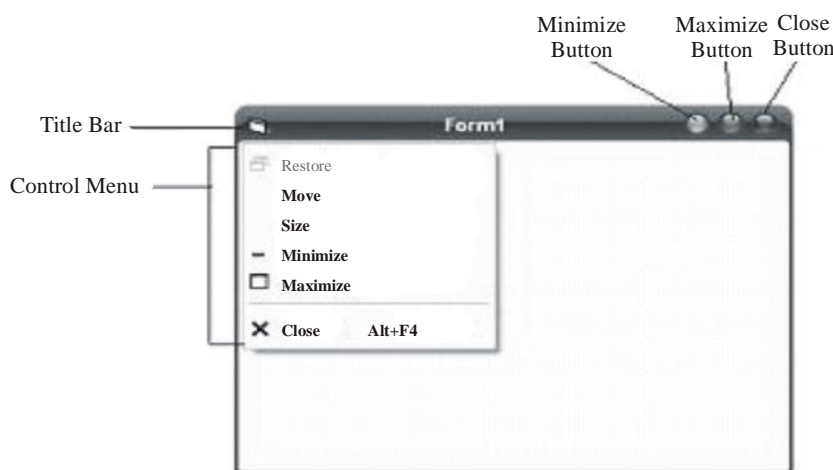


Fig. 8.1 General Appearance of Container Control Form

NOTES

NOTES

The control menu contains the following characteristics:

- **Restore:** Restores a maximized form to its size before it was maximized; available only if the Form has been maximized.
- **Move:** Lets the user move the form around with the mouse.
- **Size:** Lets the user resize the control with the mouse.
- **Minimize:** Minimizes the form.
- **Maximize:** Maximizes the form.
- **Close:** Closes the form.

Some common properties used to customize a **Form** appearance are listed in Table 8.1.

Table 8.1 Common Properties of Form Control

Property	Description														
MinButton, MaxButton	These two properties, if True display the Minimize and Maximize buttons on the title bar. You can set them to False to hide the corresponding button on the title bar. By default True.														
ControlMenu	This property displays the Control menu, if it is set to True . By default, it is True . You can set it to False to hide the Control menu icon														
BorderStyle	<p>This property determines the border's style for a Form and also its appearance. It can take any of the following values:</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>0 None</td><td>No border for form, cannot be resized.</td></tr> <tr> <td>1 Fixed Single</td><td>Visible border, but form cannot be resized.</td></tr> <tr> <td>2 Sizable</td><td>Visible border. Form can be moved and resized also.</td></tr> <tr> <td>3 Fixed Dialog</td><td>For fixed dialog boxes</td></tr> <tr> <td>4 Fixed ToolWindow</td><td>Form has a close button only, cannot be resized</td></tr> <tr> <td>5 Sizeable ToolWindow</td><td>Same as no. 4 but the form can be resized.</td></tr> </table>	Value	Description	0 None	No border for form, cannot be resized.	1 Fixed Single	Visible border, but form cannot be resized.	2 Sizable	Visible border. Form can be moved and resized also.	3 Fixed Dialog	For fixed dialog boxes	4 Fixed ToolWindow	Form has a close button only, cannot be resized	5 Sizeable ToolWindow	Same as no. 4 but the form can be resized.
Value	Description														
0 None	No border for form, cannot be resized.														
1 Fixed Single	Visible border, but form cannot be resized.														
2 Sizable	Visible border. Form can be moved and resized also.														
3 Fixed Dialog	For fixed dialog boxes														
4 Fixed ToolWindow	Form has a close button only, cannot be resized														
5 Sizeable ToolWindow	Same as no. 4 but the form can be resized.														
BackColor	Specifies the form's background colour.														
BorderStyle	Determines how the form window appears.														
Enabled	Determines whether the form is active or not														
Picture	Determines a graphic image that appears on the background of the form at run-time.														

Following code is required to print 'Welcome to Visual Basic' on Form1:

```
Private Sub Form_Load ( )
    Form1.show
    Print "Welcome to Visual Basic"
End Sub
```

After executing the above program following snapshot appears:



NOTES

Form Related Methods and Statements

Loading and Unloading Forms: A Form can generally be in one of the following three possible states:

- ¶ **Not Loaded:** In this state, the *form* lives on the disk file and does not take up any resources, such as memory, etc.
- ¶ **Loaded and Hidden:** In this state, the Form is loaded into memory and is ready to be displayed.
- ¶ **Loaded and Shown:** In this state, the Form is shown, and the user can interact with it.

The state of a Form can be changed by using Load, Unload, Show and Hide.

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

```
Load FormName
```

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 for both loading and displaying the Form, the load statement does not show the Form. You have to call the Form's Show method to display it on the desktop.

Showing Forms: The Show method is used to display a Form. If the Form is loaded but invisible, the Show method is used to bring the Form on top of the window. If the Form is not loaded, the Show method loads it and then displays it.

Syntax of the Show method of the Form is follows:

```
FormName.Show mode
```

NOTES

The `FormName` variable is the Form's name, and the optional argument mode determines whether the Form will be Modal or not. It can have one of the following syntax:

0-Modeless (default)

1-Modal

Modeless Forms are the normal Forms. Modeless Forms interact with the user and the user is allowed to switch to any other Form of the application. If you do not specify the optional mode argument, by default the mode is set to modeless.

Modal Forms take total control of the application where the user cannot switch to any other Form unless the Form is closed. A modal Form, thus, must have a Close button or some means to close the Form in order to return to the Form where the Modal Form was loaded.

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
```

In the click event of the Hide button, the following code is entered:

```
Me.Hide
```

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

```
Unload Me
```

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

To know the difference between the unload and hide methods, we open a new project and save the project. Draw two buttons on the Form and name those as shown in Figure 8.2.



Fig. 8.2 Demonstration of Unload and Hide

Form_Load event: Many events attached to a **Form**. But here we discuss only the most important one, i.e., the **Form_Load** event. This event occurs when the form is loaded in the memory. The **Form_Load()** event procedure can be used in preparing the form before showing it on the screen when the program is run.

You can open the Code Window for the `Form_Load()` event procedure by double-clicking on the Form at an empty area. You can use this event procedure to set various properties of controls on the Form. For example:

- Ñ You may set the caption text for Labels, Command Buttons, etc.
- Ñ You may even set the **Form Caption**, i.e. the text for the title bar of the Form.

You can specify many more things in the `Form_Load()` event. The following code shows a sample code written in a `Form_Load()` event procedure.

```
cmdSave      = 'Save a Documents'
cmdCancel    = 'Cancel the Action'
lblCity.Caption = 'Choose Name'
```

Form_Activate event: In the `Form_Load` event, you would perform tasks that are of the initialization type. However, some tasks cannot be performed in the Load event because the Form is fully loaded only after the completion of the Load event. For one thing, printing to the Form will not work when done in the Load event. In addition, if you want to set focus on a particular control on the Form during the Load event, you will receive the message Run-time error '5': Invalid procedure call or argument. Assume, for example, you have a textbox called `Text1` on the form. The following code would result in an error:

```
Private Sub Form_Load()
    'other initialization stuff
    Text1.SetFocus      'causes an error
End Sub
```

How to Centre a Form on the Screen

The following code will centre a Form on the screen. It is best placed in the `Form_Load` event.

```
Me.Top = (Screen.Height - Me.Height) / 2
Me.Left = (Screen.Width - Me.Width) / 2
```

In a VB (Visual Basic) project with multiple forms, it is good to centralize the form-centring logic in a Public Sub procedure of a separate standard (BAS) module. The Sub that accepts a Form object as parameter will look like this:

```
Public Sub CenterForm(pobjForm As Form)
    With pobjForm
        .Top = (Screen.Height - .Height) / 2
        .Left = (Screen.Width - .Width) / 2
    End With
End Sub
```

With the above Sub in place, any form in the project can centre itself with the following line of code:

```
CenterForm Me
```

NOTES

Setting the Startup 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 Startup of the application.

NOTES

Check Your Progress

1. Explain the term control?
2. State the one important property of control.
3. Define the term child controls.
4. State about the appearance of forms.
5. Explain the three states for loading and unloading forms.
6. Give syntax for hiding the form in Visual Basic (VB 6.0).

8.3 SUBMAIN PROCEDURE

A *procedure* is a block of Visual Basic statements enclosed by a declaration statement (Function, Sub, Operator, Get, and Set) and a matching End declaration. All executable statements in Visual Basic must be within some procedure.

Every Visual Basic application must contain a procedure called Main. This procedure serves as the starting point and overall control for your application. The .NET Framework calls your Main procedure when it has loaded your application and is ready to pass control to it. Unless you are creating a Windows Forms application, you must write the Main procedure for applications that run on their own.

Main contains the code that runs first. InMain, you can determine which form is to be loaded first when the program starts, find out if a copy of your application is already running on the system, establish a set of variables for your application, or open a DataBase that the application requires.

Requirements for the Main Procedure

A file that runs on its own (usually with extension .exe) must contain a Main procedure. A library (for example, with extension .dll) does not run on its own and does not require a Main procedure. The requirements for the different types of projects you can create are as follows:

- Console applications run on their own, and you must supply at least one Main procedure.
- Windows Forms applications run on their own. However, the Visual Basic compiler automatically generates a Main procedure in such an application, and you do not need to write one.
- Class libraries do not require a Main procedure. These include Windows Control Libraries and Web Control Libraries. Web applications are deployed as class libraries.

Declaring the Main Procedure

There are four ways to declare the `Main` procedure. It can take arguments or not, and it can return a value or not.

Note: If you declare `Main` in a class, you must use the `Shared` keyword. In a module, `Main` does not need to be `shared`.

NOTES

- The simplest way is to declare a `Sub` procedure that does not take arguments or return a value.

```
Module mainModule
    Sub Main ()
        MsgBox ("The Main procedure is starting the
        application.")
        'Insert call to appropriate starting place in
        your code.
        MsgBox ("The application is terminating.")
    End Sub
End Module
```

- `Main` can also return an `Integer` value, which the operating system uses as the exit code for your program. Other programs can test this code by examining the Windows `ERRORLEVEL` value. To return an exit code, you must declare `Main` as a `Function` procedure instead of a `Sub` procedure.

```
Module mainModule
Function Main () As Integer
    MsgBox ("The Main procedure is starting the
    application.")
    Dim return Value As Integer = 0
    'Insert call to appropriate starting place in
    your code.
    'On return, assign appropriate value to
    returnValue.
    '0 usually means successful completion.
    MsgBox ("The application is terminating with error
    level" &
        CStr (returnValue) & ".")
    Return returnValue
End Function
End Module
```

- `Main` can also take a `String` array as an argument. Each string in the array contains one of the command-line arguments used to invoke your program. You can take different actions depending on their values.

```
Module mainModule
Function Main(ByVal cmdArgs() As String) As Integer
```

NOTES

```

    MsgBox("The Main procedure is starting the
application.")
    Dim returnValue As Integer = 0
    ` See if there are any arguments.
    If cmdArgs.Length > 0 Then
For argNum As Integer = 0 To UBound(cmdArgs, 1)
        ` Insert code to examine cmdArgs(argNum)
and take
            ` appropriate action based on its value.
        Next
    End If
    ` Insert call to appropriate starting place in
your code.
    ` On return, assign appropriate value to
returnValue.
    ` 0 usually means successful completion.
    MsgBox("The application is terminating with
error level " &
        CStr(returnValue) & ".")
    Return returnValue
End Function
End Module

```

- You can declare **Main** to examine the command-line arguments but not return an exit code, as follows.

```

Module mainModule
Sub Main(ByVal cmdArgs() As String)
    MsgBox("The Main procedure is starting the
application.")
    Dim returnValue As Integer = 0
    ` See if there are any arguments.
    If cmdArgs.Length > 0 Then
For argNum As Integer = 0 To UBound(cmdArgs, 1)
        ` Insert code to examine cmdArgs(argNum)
and take
            ` appropriate action based on its value.
        Next
    End If
    ` Insert call to appropriate starting place in
your code.
    MsgBox("The application is terminating.")
End Sub
End Module

```

Check Your Progress

7. What is a procedure?
8. List the requirements for the main procedure.
9. What is the simplest way to declare Sub procedure.

NOTES

8.4 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. A control is an object that can be drawn on a Form object to enable or enhance the user interaction with an application.
2. One more important property of control is whether the control is enabled or not.
3. Controls inside containers are known as child controls. Child controls can exist completely inside their containers.
4. The main characteristic of a Form is the title bar on which the Form's caption is displayed. On the left end of the title bar is the Control Menu icon. Clicking this icon opens the control menu. Maximize, Minimize and Close buttons can be found on the right side of the Form. Clicking on these buttons performs the associated function.
5.
 - Not loaded: In this state, the Form lives on the disk file and does not take up any resources such as memory, etc.
 - Loaded and hidden: In this state, the Form is loaded into memory and is ready to be displayed.
 - Loaded and shown: In this state, the Form is shown, and the user can interact with it.
6. 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`
7. A procedure is a block of Visual Basic statements enclosed by a declaration statement (Function, Sub, Operator, Get, and Set) and a matching End declaration. All executable statements in Visual Basic must be within some procedure.
8. The requirements for the different types of projects you can create are as follows:
 - Console applications run on their own, and you must supply at least one Main procedure.
 - Windows Forms applications run on their own. However, the Visual Basic compiler automatically generates a Main procedure in such an application, and you do not need to write one.

NOTES

- Class libraries do not require a `Main` procedure. These include Windows Control Libraries and Web Control Libraries. Web applications are deployed as class libraries.
- A procedure is a block of Visual Basic statements enclosed by a declaration statement (`Function`, `Sub`, `Operator`, `Get`, `Set`) and a matching `End` declaration. All executable statements in Visual Basic must be within some procedure.

9. The simplest way is to declare a `Sub` procedure that does not take arguments or return a value is as follows:

```
Module mainModule
Sub Main ()
    MsgBox ("The Main procedure is starting the
application.")
    `Insert call to appropriate starting place in your
code.
    MsgBox ("The application is terminating.")
End Sub
End Module
```

8.5 SUMMARY

- The container for all the controls that make up the user interface is Visual Basic Form. Each window you see in a simple visual application is a shape, so the form and window terms represent the same entity.
- A control is an object that can be drawn on a Form object to enable or enhance the user interaction with an application.
- Controls have properties that define the aspects of their appearance, such as position, size and colour, and the aspects of their behaviour, such as their response to the user input.
- The Caption property of a control clearly so that a user knows what to do with that command. For example, in the calculator program, all the captions of the command buttons, such as `+`, `-`, `MC` and `MR` are commonly found in an ordinary calculator and a user should have no problem in manipulating the buttons.
- A lot of programmers like to use a meaningful name for the Name property; may be, because it is easier for them to write and read the event procedure, and debug or modify the programs later.
- One more important property is whether the control is enabled or not.
- Finally, you must also consider making the control visible or invisible at run-time or decide when it should become visible or invisible.

- A container control can hold other controls within it, for example, a Frame (there can be multiple controls inside a frame) or a PictureBox (it holds a picture) or simply your form (you can put so many controls on it).
- Controls inside containers are known as child controls. Child controls can exist completely inside their containers.
- It means that you cannot move them outside their container and if you try to drag them beyond the boundary of their container, a part of the control gets hidden. When you delete a container control, all its child controls automatically get deleted.
- In Visual Basic (VB), the Form acts as the container for all the controls that form the interface. The Form is the top-level object in a VB application, and every application starts with the Form.
- The main characteristic of a Form is the title bar on which the Form's caption is displayed. On the left end of the title bar is the Control Menu icon.
- Clicking this icon opens the control menu. Maximize, Minimize and Close buttons can be found on the right side of the Form. Clicking on these buttons performs the associated function.
- Restores a maximized form to its size before it was maximized; available only if the Form has been maximized.
- Not loaded state, the Form lives on the disk file and does not take up any resources, such as memory, etc.
- Loaded and hidden state, the Form is loaded into memory and is ready to be displayed.
- Loaded and shown state, the Form is shown, and the user can interact with it.
- The FormName variable is the name of the Form to be loaded or unloaded. Unlike the Show method, which cares for both loading and displaying the Form, the load statement does not show the Form.
- The Show method is used to display a Form. If the Form is loaded but invisible, the Show method is used to bring the Form on top of the window. If the Form is not loaded, the Show method loads it and then displays it.
- The FormName variable is the Form's name, and the optional argument mode determines whether the Form will be Modal or not.
- Modeless Forms are the normal Forms. Modeless Forms interact with the user and the user is allowed to switch to any other Form of the application.
- If you do not specify the optional mode argument, by default the mode is set to modeless.
- Modal Forms take total control of the application where the user cannot switch to any other Form unless the Form is closed.

NOTES

NOTES

- A modal Form, thus, must have a Close button or some means to close the Form in order to return to the Form where the Modal Form was loaded.
- Many events attached to a Form. But here we discuss only the most important one, i.e., the Form_Load event.
- This event occurs when the form is loaded in the memory. The Form_Load () event procedure can be used in preparing the form before showing it on the screen when the program is run.
- In the Form_Load event, you would perform tasks that are of the initialization type. However, some tasks cannot be performed in the Load event because the Form is fully loaded only after the completion of the Load event.
- For one thing, printing to the Form will not work when done in the Load event. In addition, if you want to set focus on a particular control on the Form during the Load event, you will receive the message Run-time error '5': Invalid procedure call or argument.
- In a VB project with multiple forms, it is good to centralize the form-centring logic in a Public Sub procedure of a separate standard module.
- 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 Startup of the application.
- A procedure is a block of Visual Basic statements enclosed by a declaration statement (Function, Sub, Operator, Get, Set) and a matching End declaration. All executable statements in Visual Basic must be within some procedure.
- Every Visual Basic application must contain a procedure called Main. This procedure serves as the starting point and overall control for your application.
- The .NET Framework calls your Main procedure when it has loaded your application and is ready to pass control to it.
- Unless you are creating a Windows Forms application, you must write the Main procedure for applications that run on their own.
- Main contains the code that runs first. InMain, you can determine which form is to be loaded first when the program starts, find out if a copy of your application is already running on the system, establish a set of variables for your application, or open a database that the application requires.
- A file that runs on its own (usually with extension .exe) must contain a Main procedure.
- A library (for example, with extension .dll) does not run on its own and does not require a Main procedure.
- Console applications run on their own, and you must supply at least one Main procedure.

- Windows Forms applications run on their own. However, the Visual Basic compiler automatically generates a `Main` procedure in such an application, and you do not need to write one.
- Class libraries do not require a `Main` procedure. These include Windows Control Libraries and Web Control Libraries. Web applications are deployed as class libraries.
- There are four ways to declare the `Main` procedure. It can take arguments or not, and it can return a value or not.
- The simplest way is to declare a `Sub` procedure that does not take arguments or return a value.

NOTES

8.6 KEY WORDS

- Ñ **Form:** Form allows the developer of Visual Basic to build window and layout controls.
- Ñ **Control:** It is an object that can be drawn on a Form object for enabling or enhancing user interaction with an application.
- Ñ **Child controls:** They are the controls inside containers.
- Ñ **Container control:** A control that can hold other controls within it.
- Ñ **Procedure:** A procedure is a block of Visual Basic statements enclosed by a declaration statement (`Function`, `Sub`, `Operator`, `Get`, `Set`) and a matching `End` declaration. All executable statements in Visual Basic must be within some procedure.
- Ñ **Main:** `Main` contains the code that runs first.

8.7 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. Explain the term control.
2. Define the term child form.
3. State about the form control.
4. Write the three states for loading and unloading forms.
5. What is Form load event?
6. Explain the term procedure.
7. What is the requirement for the `Main` procedure?

NOTES**Long-Answer Questions**

1. List the important points which should be kept in mind while setting up the properties.
2. Discuss briefly about the forms with the help of example and diagrams.
3. Briefly discuss about the characteristics of control menu.
4. Write notes on the following giving examples.
 - a) Form_Load event
 - b) Form_Activate event
5. Briefly explain about the centring of a form on the screen.
6. Describe the submain procedure and Requirements for the Main Procedure?
7. Discuss about the declaration of the Main Procedure giving examples.

8.8 FURTHER READINGS

- Cornell, Gary. 1998. *Visual Basic 6 from the Ground Up*. New Delhi: Tata McGraw-Hill.
- Warner, Scott L. 1998. *Teach Yourself Visual Basic 6*. New Delhi: Tata McGraw-Hill.
- Jerke, Noel. 1999. *Visual Basic 6 - The Complete Reference*. New York: McGraw-Hill.
- Smith, Eric A., Valor Whisler and Hank Marquis. 1998. *Visual Basic 6 Programming Bible*. New Jersey: John Wiley & Sons, Inc.
- Azam, M. 2001. *Programming with Visual Basic*. New Delhi: Vikas Publishing House Pvt. Ltd.
- Manchanda, Mahesh. 2009. *Visual Programming*. New Delhi: Vikas Publishing House Pvt. Ltd.
- Balena, Francesco. 1999. *Programming Microsoft Visual Basic 6.0*. Bangalore: WP Publishers and Distributors (P) Ltd.
- Petroutsos, Evangelos. 1998. *Mastering Visual Basic 6, 1st Edition*. New Delhi: BPB Publications.
- Deitel, Harvey M., Paul J. Deitel and T. Tem R. Nieto. 1999. *Visual Basic 6: How to Program*. New Jersey: Prentice-Hall.
- Norton, Peter. 1998. *Peter Norton's Complete Guide to Visual Basic 6*. New Delhi: Techmedia.
- Reselman, Bob and Richard A. Peasley. 1998. *Using Visual Basic 6*. New Jersey: Pearson Education (Que Publishing).
- Donald, Bob and Oancea Gabriel. 1999. *Visual Basic 6 from Scratch*. New Delhi: Prentice-Hall of India.

UNIT 9 ARRAYS

- 9.0 Introduction
- 9.1 Objectives
- 9.2 Arraya and Control Arrays
 - 9.2.1 Fixed Size Arrays
 - 9.2.2 Dynamic arrays
 - 9.2.3 Array Characterstics
 - 9.2.4 Processing Array Elements
 - 9.2.5 Control arrays
- 9.3 Indexing and Event Handling
- 9.4 Graphics
- 9.5 Answers to Check Your Progress Questions
- 9.6 Summary
- 9.7 Key Words
- 9.8 Self-Assessment Questions and Exercises
- 9.9 Further Readings

NOTES

9.0 INTRODUCTION

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

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

In Visual Basic, a control array is a group of related controls in a Visual Basic form that share the same event handlers. Control arrays are always single-dimensional arrays, and controls can be added or deleted from control arrays at runtime. One application of control arrays is to hold menu items, as the shared event handler can be used for code common to all of the menu items in the control array.

Control arrays are a convenient way to handle groups of controls that perform a similar function. All of the events available to the single control are still available to the array of controls, the only difference being an argument indicating the index of the selected array element is passed to the event. Hence, instead of writing individual procedures for each control (i.e. not using control arrays) write one procedure for each array.

Event-driven programming is a programming paradigm in which the flow of the program is determined by events such as user actions (mouse clicks, key

NOTES

presses), sensor outputs, or messages from other programs or threads. Event-driven programming is the dominant paradigm used in graphical user interfaces and other applications that are centred on performing certain actions in response to user input.

Creating graphics was relatively easy in earlier versions of Visual Basic because they have built-in drawing tools. For example, In Visual Basic 6.0, the drawing tools are included in the Toolbox where the programmer just needs to drag the shape controls into the form to create rectangle, square, ellipse, circle and more. However, its simplicity has the shortcomings, do not have many choices for creating customized drawings.

Before drawing anything on a Form in Visual Basic VB 6.0, create the Graphics object in Visual Basic 6.0. A graphics object is created using the `CreateGraphics()` method.

In this unit, you will study about the control array, indexing and event handling and graphics in Visual Basic.

9.1 OBJECTIVES

After going through this unit, you will be able to:

- Know about the control arrays
 - Understand the basics of arrays and its types
 - Define the characteristics of array
 - Discuss about the processing of elements of an array
 - Elaborate on the concept of event handling and different types of events in Visual Basic
 - Explain about the graphics
-

9.2 ARRAYA AND CONTROL ARRAYS

An array is a set of similar items. All items in an array have the same name and are identified by an index. Arrays allow you to refer to a series of variables by the same name and to use a specific number (an index) to uniquely identify these variables.

Syntax for Control Array is given as follows:

```
Dim Varname ([[subscripts]]) as [New] type [,varname....]
```

Example of declarations of arrays.

```
Dim nums(10) as integer
```

`nums` is an array containing 11 integers. `nums(0)` is the first element of the array and `nums(10)` is the last element of the array. In this declaration 10 is known as the upper bound of the array.

```
Dim x(10 to 20) as integer.
```

`x` is an array containing 11 integers. `x(10)` is the first element of the array and `x(20)` is the last element of the array. In this declaration, 10 is the lower declaration and 20 is the upper declaration.

For example, you want to compute the monthly sales of an organization. Sales figures for each month have to be calculated. This means that you need to have at least 12 variables, such as `SaleJan`, `SaleFeb`, `SaleMar` and `SaleDec`, etc. You will agree that it is a cumbersome task to declare and manage twelve variables when you can do so with one variable. For this, declare the variable in the following manner:

```
Dim Saleval(11) as Long
```

Since there are twelve months, `Saleval(0)` will hold the sales figures of the first month and `Saleval(5)` will hold the sales figures of the sixth month. Arrays can have more than one dimension. A table of data will be represented by a multidimensional array. To continue with the above example, if you want to record the sales figures for twelve months of three departments of the organization, you would declare the array as follows:

```
Dim Saleval(11,2) As Integer
```

Here, the subscript 11 indicates the months and the subscript 2 indicates the departments.

This is a two-dimensional array. You can also have three-dimensional arrays. If you want to record the sales of five products of three departments for twelve months, then you need to declare a three-dimensional array as follows.

```
Dim Saleval(11,2,4) As Integer
```

Where the subscript 11 indicates months, the subscript 2 indicates the three departments and the subscript 4 indicates the 5 products.

The arrays that we have just declared are fixed sized arrays. We know the total number of items that the array will hold. There are three types of arrays:

- Fixed Sized
- Dynamic
- Control

9.2.1 Fixed Size Arrays

Fixed arrays have a fixed size which can not be changed at run time. These are also known as array.

In the case of fixed sized arrays, it is compulsory to enter the upper bound of an array in the parenthesis. The upper bound is the upper limit for the size of the array. A declaration of fixed size arrays is as follows:

```
Private Counters(10) As Integer
```

The above code declares an array of 11 elements. In this code, 10 is the upper bound of the array.

NOTES

NOTES

To specify the lower bound of an array, provide it explicitly (as a Long data type) using the `To` keyword:

```
Dim Counter(1 To 10) As Integer
```

In the above declaration, the index numbers of `Counter` range from 1 to 10.

9.2.2 Dynamic Arrays

Dynamic arrays are used when you do not know the number of elements for an array. For example, when you are reading a string into an array, you may want to have the capability of changing the size of the array at runtime.

A dynamic array can be resized at any time and this helps to manage the memory efficiently. For example, you can use a large array for a short time and then free memory to the system when the array is not in use. Alternatively, you can increase the size of the array after declaring a smaller array.

9.2.3 Array Characteristics

Various characteristics of a VB array are as follows:

- An array contains several data elements, which are of the same data type and share the same name, i.e., name of the array.
- An array index starts from 0 to $n-1$ when accessing the array elements or assigning values to the array elements. For example, for an array `x(10)`, `x(0)` refers to the first element of the array and `x(9)` refers to the last element for the array.
- The number of subscripts in an array determines the dimension of the array. For example:

```
Dim arr(5) As Integer ' Defines one- dimensional array of size '5.
```

```
Dim arr(5,6) As Integer ' Defines two dimensional array of 'size 5x6.
```

9.2.4 Processing Array Elements

An array is a collection of different elements and each element of an array is called subscripted variable. Using the following syntax, you can access these subscripted variables:

```
<array name>(<element number>)
```

In this syntax, `array name` refers to the name of the array and the `element number` specifies the array index number, which you want to access. Now, consider the following code that processes of array elements:

```
Private Sub Form_Load()  
Dim x As Integer  
Dim a(5) As Integer 'Declare an Integer array, array(5),  
of ' 6 elements
```

```

a(0) = 0
a(1) = 10
a(2) = 20
a(3) = 30
a(4) = 40
a(5) = 50
x = 0
For i = 0 To 5
x = a(i) + x
Next i
MsgBox (x)
End Sub

```

NOTES

Figure 9.1 shows the Code window for processing array elements.

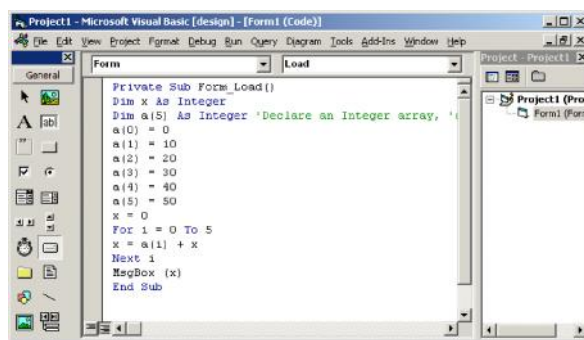


Fig. 9.1 Displaying the Code Window for Adding Array Elements

Compile and execute the above program. Figure 9.2 shows the sum of the array elements.



Fig. 9.2 Output for Adding Array Elements

In the above code, first we have created an array `a (5)` and then entered different values for each element of the array. Then, we initialize a `For...Next` loop from `i=0` to `i=5`. This loop extracts all the elements from the array one by one. The line `x = a(i) + x` adds all these elements and stores into the variable `x`. After extracting and adding all the elements of the array, the `msgbox ()` function prints the value of `x` on the message box. The box displays the value 150 which is the sum of all the array elements.

NOTES**9.2.5 Control Arrays**

A control array is a collection of more than one VB control that shares the same name, which is the name of the array. All the controls in an array must be same, such as an array of Labels. In a control array, each control has a unique index number and you can access the control by using these unique numbers only. By increasing the index number, you can access all the subsequent controls in an array.

You can create a control array in the following two ways:

- Set the `Index` property of a control during the design time. For example, if you set the index of a `Label1` to 0, then it will be the first `Label1` in the control array of labels.
- Create a control component on the design window. Now, click the control component and press `Ctrl+C` to copy that control. After copying the component, press `Ctrl+V` to paste that copied control. When you paste the component, the Visual Basic (VB) compiler asks to create a control array.

Figure 9.3 shows Microsoft Visual Basic warning dialog box.

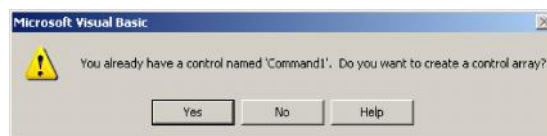


Fig. 9.3 Warning for Creating a Control Array

Now, click Yes to create a control array, else click No.

You can use the following index to access each element of the control array:

`<control array name>(array index)`

You can also change the properties of the control array elements by using the following syntax:

`<control array name>(array index).<property name> = <property value>`

Check Your Progress

1. Give the definition of an arrays.
2. Why arrays are used?
3. How many types of array are used?
4. Define the term fixed size arrays.
5. What is the use of dynamic arrays?
6. Write one characteristics of array.
7. Explain the term control arrays.
8. Define the term subscripted variable.

9.3 INDEXING AND EVENT HANDLING

Event Handling means performing some actions by using events. By using Event Handling, we can perform Validations in windows application which is a built in feature in the web application. For Web applications, we have several controls which are used to perform validations. But there are no controls in windows applications. By using this Event Handling Mechanism, we can achieve validations in Windows Application also. There are two Types of Events as follows.

- Mouse Events
- Keyboard Events

The Events which are raised due to **Mouse** are called as **Mouse Events**. The following table shows you some of the **Mouse Events**.

Event	Description
MouseUp	Raised when the Mouse pointer over the control
MouseDown	Raised when the Mouse button released
MouseEnter	Raised when the Mouse pointer enters the control
MouseHover	Raised when the Mouse pointer moves on the control
MouseLeave	Raised when the Mouse pointer leaves the control

The following table shows you some of the **Properties** of the Mouse.

Properties	Description
Click	Indicates the Number of Clicks
X	Indicates the x-Coordinates of the mouse click
Y	Indicates the y-Coordinates of the mouse click
Buttons	Indicates the mouse button pressed

The Events which are raised due to **Keyboard** are called as **KeyBoard Events**. The following table shows you some of the **Keyboard Events**.

Event	Description
KeyUp	Raised when the Key is released
KeyDown	Raised when the Key is pressed down
KeyPress	Raised when the Key is pressed

NOTES

NOTES

The following table shows you some of the **Properties** of the Mouse.

Properties	Description
KeyData	Used to store the Key Data for the Event
KeyCode	Used to store the Key Code for the Event
Handled	Used to check whether the event is handled or not
KeyValue	Used to store the Key Value for the Event
KeyChar	Used to store the Character related to that key which is pressed for the Event

Example

Following is an example, which shows how to handle mouse events. Consider the following steps—

- Add three labels, three text boxes and a button control in the form.
- Change the text properties of the labels to - Customer ID, Name and Address, respectively.
- Change the name properties of the text boxes to txtID, txtName and txtAddress, respectively.
- Change the text property of the button to 'Submit'.
- Add the following code in the code editor window —

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs)
        Handles MyBase.Load
            ' Set the caption bar text of the form.
            Me.Text = "tutorialsont.com"
        End Sub

    Private Sub txtID_MouseEnter(sender As Object, e As
        EventArgs)_
        Handles txtID.MouseEnter
            'code for handling mouse enter on ID textbox
            txtID.BackColor = Color.CornflowerBlue
            txtID.ForeColor = Color.White
        End Sub

    Private Sub txtID_MouseLeave(sender As Object, e As
        EventArgs) _
        Handles txtID.MouseLeave
            'code for handling mouse leave on ID textbox
            txtID.BackColor = Color.White
        End Sub
End Class
```

```

        txtID.ForeColor = Color.Blue
    End Sub

    Private Sub txtName_MouseEnter(sender As Object, e As
EventArgs) _
        Handles txtName.MouseEnter
        'code for handling mouse enter on Name textbox
        txtName.BackColor = Color.CornflowerBlue
        txtName.ForeColor = Color.White
    End Sub

    Private Sub txtName_MouseLeave(sender As Object, e As
EventArgs) _
        Handles txtName.MouseLeave
        'code for handling mouse leave on Name textbox
        txtName.BackColor = Color.White
        txtName.ForeColor = Color.Blue
    End Sub

    Private Sub txtAddress_MouseEnter(sender As Object, e
As EventArgs) _
        Handles txtAddress.MouseEnter
        'code for handling mouse enter on Address textbox
        txtAddress.BackColor = Color.CornflowerBlue
        txtAddress.ForeColor = Color.White
    End Sub

    Private Sub txtAddress_MouseLeave(sender As Object, e
As EventArgs) _
        Handles txtAddress.MouseLeave
        'code for handling mouse leave on Address textbox
        txtAddress.BackColor = Color.White
        txtAddress.ForeColor = Color.Blue
    End Sub

    Private Sub Button1_Click(sender As Object, e As
EventArgs) _
        Handles Button1.Click
        MsgBox("Thank you " & txtName.Text & ", for your
kind cooperation")
    End Sub
End Class

```

NOTES

When the above code is executed and run using **Start** button available at the Microsoft Visual Studio tool bar, it will show the following window.

Try to enter text in the text boxes and check the mouse events.

NOTES

Handling Keyboard Events

Following are the various keyboard events related with a Control class –

- **KeyDown**– occurs when a key is pressed down and the control has focus
- **KeyPress**– occurs when a key is pressed and the control has focus
- **KeyUp**– occurs when a key is released while the control has focus

The event handlers of the KeyDown and KeyUp events get an argument of type **KeyEventArgs**. This object has the following properties–

- **Alt**– it indicates whether the ALT key is pressed
- **Control**– it indicates whether the CTRL key is pressed
- **Handled**– it indicates whether the event is handled
- **KeyCode**– stores the keyboard code for the event
- **KeyData**– stores the keyboard data for the event
- **KeyValue**– stores the keyboard value for the event
- **Modifiers**– it indicates which modifier keys (Ctrl, Shift, and/or Alt) are pressed
- **Shift**– it indicates if the Shift key is pressed

The event handlers of the KeyDown and KeyUp events get an argument of type **KeyEventArgs**. This object has the following properties–

- **Handled**– indicates if the KeyPress event is handled
- **KeyChar**– stores the character corresponding to the key pressed

Example

Let us continue with the previous example to show how to handle keyboard events. The code will verify that the user enters some numbers for his customer ID and age.

- Add a label with text Property as 'Age' and add a corresponding TextBox named TextAge.
- Add the following codes for handling the KeyUP events of the TextBox TextID.

```
Private Sub txtID_KeyUP(sender As Object, e As
KeyEventArgs) _
    Handles txtID.KeyUp

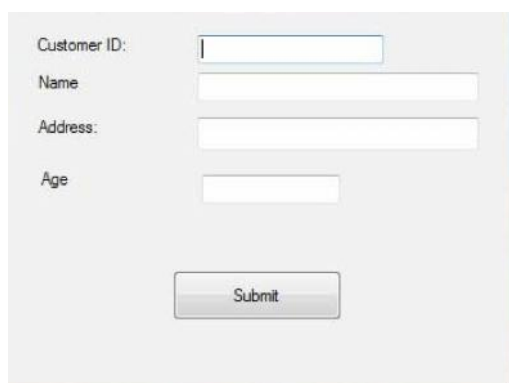
    If (Not Char.IsNumber(ChrW(e.KeyCode))) Then
        MessageBox.Show("Enter numbers for your Customer
ID")
        txtID.Text = " "
    End If
End Sub
```

- Add the following codes for handling the KeyUP events of the TextBox txtID.

```
Private Sub txtAge_KeyUP(sender As Object, e As
KeyEventArgs) _
    Handles txtAge.KeyUp

    If (Not Char.IsNumber(ChrW(e.keyCode))) Then
        MessageBox.Show("Enter numbers for age")
        txtAge.Text = " "
    End If
End Sub
```

When the above code is executed and run using **Start** button available at the Microsoft Visual Studio ToolBar, it will show the following window –

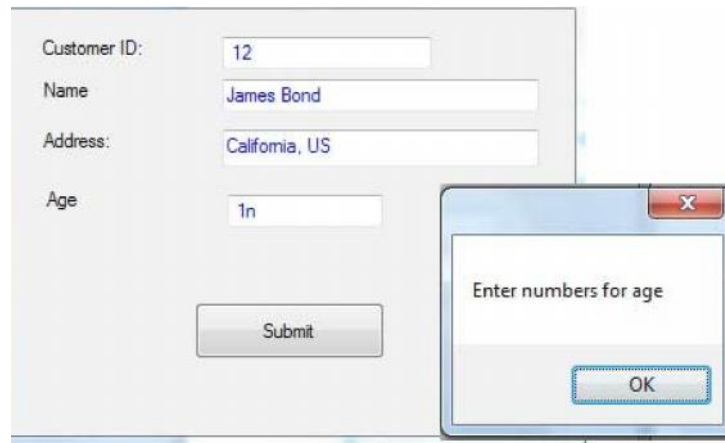


The screenshot shows a simple Windows application window. It contains four text input fields arranged vertically, each preceded by a label: 'Customer ID:', 'Name', 'Address:', and 'Age'. Below these fields is a single button labeled 'Submit'. The window has a standard Windows title bar and a light gray background.

NOTES

If you leave the text for age or ID as blank or enter some non-numeric data, it gives a warning message box and clears the respective text.

NOTES



9.4 GRAPHICS

In Visual Basic 6.0, working with graphics is easy. In simple steps, VB 6.0 gives you the versatility and power to render graphical applications. It is rich in functions related to graphics. Methods, properties and events that are built-in allow you to use these features most effectively. You can perform some graphical operations via the graphic methods and properties. They allow you, more clearly, to draw points, lines, circles, rectangles, ellipses, and other types.

Graphic Methods

You can draw on the shape and the PictureBox control using the graphic methods. In Visual Basic 6, only the Shape object and the PictureBox control support graphics methods. Later Visual Basic versions, however, allow you to use them with other objects and controls.

The common graphic methods are explained below.

- **Print:** Print is the simplest graphic method in Visual Basic 6.0. This method has been used throughout the earlier versions of the language. It prints some text on the form or on the PictureBox control. It displays texts.
- **Cls:** The Cls method is another simple graphic method that is used to clear the surface of the form or the PictureBox control. If some texts are present, you can use the Cls method to remove the texts. It clears any drawing created by the graphic methods.

- **Point:** The Point method returns the color value from an image for a pixel at a particular point. This method is generally used to retrieve color values from bitmaps.
- **Refresh:** The refresh method redraws a control or object. In other words, it refreshes the control. Generally, controls are refreshed automatically most of the times. But in some cases, you need to refresh a control's appearance manually by explicitly invoking the Refresh method.
- **PSet:** The PSet method sets the color of a single pixel on the form. This method is used to draw points.
- **Line:** The Line method draws a line. Using the Line method, you can also draw other geometric shapes such as rectangle, triangle etc.
- **Circle:** The Circle method draws a circle. Using the Circle method, you can also draw other geometric shapes such as ellipses, arcs etc.
- **PaintPicture:** The PaintPicture method displays an image on the form at run-time.
- **TextHeight:** The TextHeight method returns the height of a string on the form at run-time.
- **TextWidth:** The TextWidth method returns the width of a string on the form at run-time.

NOTES

The LoadPicture Function

The LoadPicture function loads a picture to the form or to the PictureBox control. It sets the picture to the control in order to display it. The function takes the file path as an argument. The LoadPicture function allows you to set pictures at run-time.

Example:

Code:

```
Picture1 = LoadPicture("C:\MyPic.JPG")
```

Here, Picture1 is the PictureBox control. When this code is executed the picture is loaded and set to the PictureBox control. If Visual Basic cannot find the picture in the specified location, it throws a run-time error '53'.

The RGB function

The RGB function returns an integer, a color code which is used to set colors in Visual Basic code. The RGB color code is a combination of red, green and blue colors. Consider the following example to understand RGB function in Visual Basic 6.0 (VB 6.0)

NOTES

Example Code:

```
Form1.BackColor = RGB (120, 87, 55)
```

The RGB color is set as the background color of the form object. The first, second and the third arguments represent red, green and blue colors respectively. The color value is an integer. These values are in a range of 0 to 255. So you can use any value between 0 and 255 to obtain a color.

Graphic Properties

The graphic properties are useful while working with the graphic methods. Some of the form's properties and some of the PictureBox's properties are the graphics properties.

The common graphic properties are discussed in this section.

Consider the following graphic properties.

- **DrawMode:** The DrawMode property sets the mode of drawing for the appearance of output from the graphic methods. In the DrawMode property, you can choose from a variety of values.
- **DrawStyle:** The DrawStyle property sets the line style of any drawing from any graphic methods. It allows you to draw shapes of different line styles such as solid, dotted, dashed shapes etc.
- **DrawWidth:** The DrawWidth property sets the line width of any drawing from any graphic methods. While drawing shapes, you can control the thickness of the lines using this property.
- **FillColor:** The FillColor property is used to fill any shapes with a color. You may use the symbolic color constants to fill your shapes. You may also use the color codes as well as the RGB function.
- **FillStyle:** The FillStyle property lets you fill shapes in a particular filling style.
- **ForeColor:** The ForeColor property is used to set or return the foreground color.
- **AutoRedraw:** Set the AutoRedraw property to True to get a persistent graphics when you're calling the graphic methods from any event, but not from the Paint event.
- **ClipControls:** Set the ClipControls property to True to make the graphic methods repaint an object.
- **Picture:** The Picture property is used to set a picture. Pictures can be set both at design time and run-time.

Run Time Graphic Properties

CurrentX and CurrentY are the run-time properties which are used to set and return the position of a shape or point at run-time.

- **CurrentX:** The CurrentX property sets or returns the horizontal coordinate or X-coordinate of the current graphic position at run-time.
- **CurrentY:** The CurrentY property sets or returns the vertical coordinate or Y-coordinate of the current graphic position at run-time.

NOTES

Printing On The Form

The following code prints some text on the form.

Example Code 1:

```
Private Sub cmdPrint_Click()
    Form1.Print "Hello world"
    Form1.Print "Welcome to Visual Basic 6"
    Form1.Print "Visual Basic is awesome!"
End Sub
```

The above code can be written in the following way too.

Example Code 2:

```
Private Sub cmdPrint_Click()
    Print "Hello world"
    Print "Welcome to Visual Basic 6"
    Print "Visual Basic is awesome!"
End Sub
```

In the above code, the Print method is called without the object name. Here 'Form1' is the object name. When you're writing code inside the form module, you may omit the form's name while invoking its methods.

Drawing lines

The Line method lets you draw lines in Visual Basic 6.0. You need to specify the starting point and the finishing point of the line in the argument. You may also specify the color of the line.

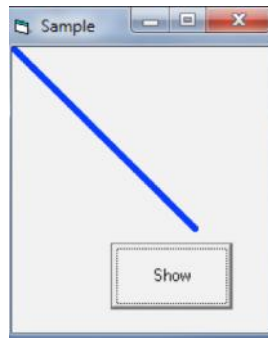
A Simple Line

The following code example shows how to draw a simple line using the Line method in Visual Basic 6.0.

NOTES

Code:

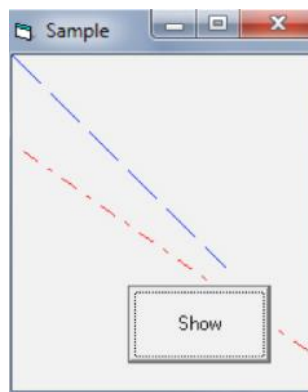
```
Private Sub cmdShow_Click()  
    DrawWidth = 5  
    'A hyphen is required between the points  
    Line (0, 0)-(2000, 2000), vbBlue  
End Sub
```



A Line with Drawing Styles

Form's DrawStyle property help you to draw lines using a particular style. The constant values of the DrawStyle property are 0 (vbSolid), 1 (vbDash), 2 (vbDot), 3 (vbDashDot), 4 (vbDashDotDot), 5 (vbTransparent) and 6 (vbInsideSolid). The default value is 0, vbSolid. You may use the numeric constant or the symbolic constant, such as vbSolid, vbDash etc., to change drawing styles in your code.

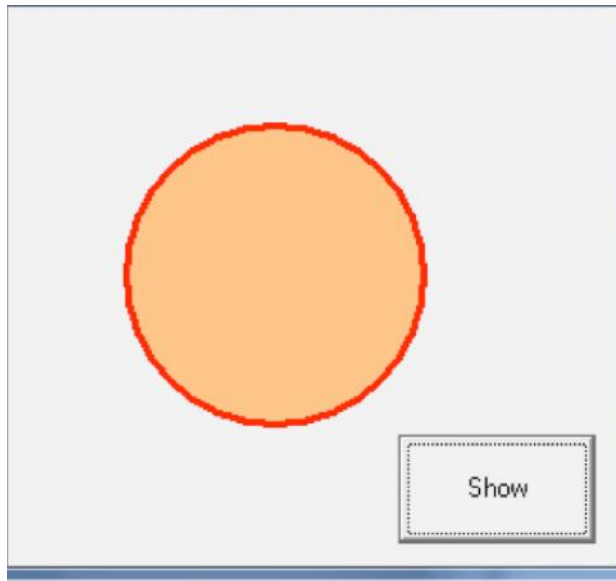
NOTE: The DrawStyle property does not work if the value of DrawWidth is other than 1.



A Circle Filled with Color

The following code example shows how to fill a circle with color in Visual Basic 6.0.

```
Private Sub cmdShow_Click()  
    FillStyle = vbSolid  
    FillColor = &H80C0FF  
    DrawWidth = 3  
    Circle (1800, 1800), 1000, vbRed  
End Sub
```



NOTES

Check Your Progress

9. Elaborate on the event handling in Visual Basic.
10. Write the types of events used in VB.
11. What is Graphics?
12. Explain the term LoadPicture function.
13. Elaborate on the drawing lines.

9.5 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. An array is a set of similar items. All items in an array have the same name and are identified by an index.
2. Arrays allow you to refer to a series of variables by the same name and to use a specific number (an index) to uniquely identify these variables.

NOTES

3. There are three types of arrays:
 - Fixed Sized Array
 - Dynamic Array
 - Control Array
5. Fixed arrays have a fixed size which cannot be changed at run-time. These are also known as Static Arrays.
6. An array contains several data elements, which are of the same data type and share the same name, i.e., name of the array.
7. A control array is a collection of more than one VB control that shares the same name, which is the name of the array. All the controls in an array must be same, such as an array of Labels. In a control array, each control has a unique index number and you can access the control by using these unique numbers only. By increasing the index number, you can access all the subsequent controls in an array.
8. An array is a collection of different elements and each element of an array is called subscripted variable.
9. Event Handling means performing some actions by using events. By using Event Handling, we can perform Validations in windows application which is a built in feature in the web application.
10. There are two types of events as follows.
 - Mouse Events
 - Keyboard Events
11. In Visual Basic 6.0, various graphics methods and properties are used to draw on a Form or PictureBox control. Graphics in Visual Basic 6.0 are based on the Windows Graphics Device Interface (GDI) APIs. In Visual Basic 6.0, graphics methods apply only to the Form object and to the PictureBox control.
12. The LoadPicture function loads a picture to the form or to the PictureBox control. It sets the picture to the control in order to display it. The function takes the file path as an argument. The LoadPicture function allows you to set pictures at run-time.
13. The Line method lets you draw lines in Visual Basic 6. You need to specify the starting point and the finishing point of the line in the argument. You may also specify the color of the line. This is optional, though.

9.6 SUMMARY

- An array is a set of similar items. All items in an array have the same name and are identified by an index.

- Arrays allow you to refer to a series of variables by the same name and to use a specific number (an index) to uniquely identify these variables. You want to compute the monthly sales of an organization. Sales figures for each month have to be calculated. This means that you need to have at least 12 variables, such as SaleJan, SaleFeb, SaleMar and SaleDec, etc.
- You will agree that it is a cumbersome task to declare and manage twelve variables when you can do so with one variable. For this, declare the variable in the following manner.
- This is a two-dimensional array. You can also have three-dimensional arrays. If you want to record the sales of five products of three departments for twelve months, then you need to declare a three-dimensional array.
- In the case of fixed sized arrays, it is compulsory to enter the upper bound of an array in the parenthesis. The upper bound is the upper limit for the size of the array.
- Dynamic arrays are used when you do not know the number of elements for an array.
- A dynamic array can be resized at any time and this helps to manage the memory efficiently.
- A control array is a collection of more than one VB control that shares the same name, which is the name of the array.
- An array index starts from 0 to n-1 when accessing the array elements or assigning values to the array elements. For example, for an array x(10), x(0) refers to the first element of the array and x(9) refers to the last element for the array.
- An array is a collection of different elements and each element of an array is called subscripted variable.
- A control array is a collection of more than one VB control that shares the same name, which is the name of the array.
- In a control array, each control has a unique index number and you can access the control by using these unique numbers only. By increasing the index number, you can access all the subsequent controls in an array.
- An array contains several data elements, which are of the same data type and share the same name, i.e., name of the array.
- Event Handling means performing some actions by using events. By using Event Handling, we can perform Validations in windows application which is a built in feature in the web application.
- For Web applications, we have several controls which are used to perform validations. But there are no controls in windows applications.

NOTES

NOTES

- By using this Event Handling Mechanism, we can achieve validations in Windows Application also.
- In Visual Basic 6.0, working with graphics is easy. In simple steps, VB6 gives you the versatility and power to render graphical applications.
- It is rich in functions related to graphics. Methods, properties and events that are built-in allow you to use these features most effectively.
- You can perform some graphical operations via the graphic methods and properties. They allow you, more clearly, to draw points, lines, circles, rectangles, ellipses, and other types.
- You can draw on the shape and the PictureBox control using the graphic methods.
- In Visual Basic 6.0, only the Shape object and the PictureBox control support graphics methods.
- Print is the simplest graphic method in Visual Basic 6. This method has been used throughout the earlier versions of the language. It prints some text on the form or on the PictureBox control. It displays texts.
- The Cls method is another simple graphic method that is used to clear the surface of the form or the PictureBox control. If some texts are present, you can use the Cls method to remove the texts. It clears any drawing created by the graphic methods.
- The Point method returns the color value from an image for a pixel at a particular point. This method is generally used to retrieve color values from bitmaps.
- The refresh method redraws a control or object. In other words, it refreshes the control. Generally, controls are refreshed automatically most of the times. But in some cases, you need to refresh a control's appearance manually by explicitly invoking the Refresh method.
- The PSet method sets the color of a single pixel on the form. This method is used to draw points.
- The Line method draws a line. Using the Line method, you can also draw other geometric shapes such as rectangle, triangle etc.
- The Circle method draws a circle. Using the Circle method, you can also draw other geometric shapes such as ellipses, arcs etc.
- The PaintPicture method displays an image on the form at run-time.
- The TextHeight method returns the height of a string on the form at run-time.
- TextWidth method returns the width of a string on the form at run-time.

- The LoadPicture function loads a picture to the form or to the PictureBox control. It sets the picture to the control in order to display it.
- The function takes the file path as an argument. The LoadPicture function allows you to set pictures at run-time.
- The RGB function returns an integer, a color code which is used to set colors in Visual Basic code. The RGB color code is a combination of red, green and blue colors.
- The graphic properties are useful while working with the graphic methods. Some of the form's properties and some of the PictureBox's properties are the graphics properties.
- The common graphic properties are discussed in this section. You'll learn more about them using code examples later in this tutorial.
- The DrawMode property sets the mode of drawing for the appearance of output from the graphic methods. In the DrawMode property, you can choose from a variety of values.
- The DrawStyle property sets the line style of any drawing from any graphic methods. It allows you to draw shapes of different line styles such as solid, dotted, dashed shapes etc.
- The DrawWidth property sets the line width of any drawing from any graphic methods. While drawing shapes, you can control the thickness of the lines using this property.
- The FillColor property is used to fill any shapes with a color. You may use the symbolic color constants to fill your shapes. You may also use the color codes as well as the RGB function.
- The FillStyle property lets you fill shapes in a particular filling style.
- The ForeColor property is used to set or return the foreground color.
- Set the AutoRedraw property to True to get a persistent graphics when you're calling the graphic methods from any event, but not from the Paint event.
- Set the ClipControls property to True to make the graphic methods repaint an object.
- The Picture property is used to set a picture. Pictures can be set both at design time and run-time.
- CurrentX and CurrentY are the run-time properties which are used to set and return the position of a shape or point at run-time.
- The Line method lets you draw lines in Visual Basic 6. You need to specify the starting point and the finishing point of the line in the argument. You may also specify the color of the line. This is optional, though.

NOTES

NOTES

9.7 KEY WORDS

- **Array:** A set of similar items.
 - **Control array:** A collection of more than one VB control that shares the same name which is the name of the array.
 - **Indexing:** Array indexing is the same as accessing an array element. You can access an array element by referring to its index number. The indexes in arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.
 - **Event:** An event is a message sent by an object within a program to the main program loop, informing it that something has happened.
 - **Graphics:** In Visual Basic 6.0, various graphics methods and properties are used to draw on a Form or PictureBox control. Graphics in Visual Basic 6.0 are based on the WindowsGraphics Device Interface (GDI) APIs. In Visual Basic 6.0, graphics methods apply only to the Form object and to the PictureBox control.
 - **LoadPicture:** LoadPicture function loads a picture to the form or to the PictureBox control.
-

9.8 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. Why we use control array?
2. Define the term fixed size arrays.
3. What are dynamic arrays?
4. Elaborate on the processing array elements.
5. Explain the term of indexing handling.
6. Define the term of event handling.
7. Explain the various types of events.
8. Elucidate on the graphics.
9. Explain the term RGB function.
10. Write the properties for run-time graphic.
11. How the lines are drawn?

Long-Answer Questions

1. Discuss briefly the control arrays and its types.
2. Explain control arrays. How array elements can be processed in VB program?
3. Difference between dynamic arrays and control arrays.
4. Discuss briefly characteristics of arrays.
5. Describe the indexing and event handling and types of event.
6. Differentiate between mouse and keyboard events, giving appropriate example.
7. Explain the significance of Keyboard Events with the help of example?
8. Briefly discuss about the common method for graphics method.
9. Elaborate briefly on the LoadPicture and RGB function, giving examples.
10. Write the properties of graphics support your ans with the help of appropriate example.
11. What are printing text? And how to print the text on the form?
12. Discuss about the drawing lines with the help of diagram.

NOTES

9.9 FURTHER READINGS

- Cornell, Gary. 1998. *Visual Basic 6 from the Ground Up*. New Delhi: Tata McGraw-Hill.
- Warner, Scott L. 1998. *Teach Yourself Visual Basic 6*. New Delhi: Tata McGraw-Hill.
- Jerke, Noel. 1999. *Visual Basic 6 - The Complete Reference*. New York: McGraw-Hill.
- Smith, Eric A., Valor Whisler and Hank Marquis. 1998. *Visual Basic 6 Programming Bible*. New Jersey: John Wiley & Sons, Inc.
- Azam, M. 2001. *Programming with Visual Basic*. New Delhi: Vikas Publishing House Pvt. Ltd.
- Manchanda, Mahesh. 2009. *Visual Programming*. New Delhi: Vikas Publishing House Pvt. Ltd.
- Balena, Francesco. 1999. *Programming Microsoft Visual Basic 6.0*. Bangalore: WP Publishers and Distributors (P) Ltd.
- Petroutsos, Evangelos. 1998. *Mastering Visual Basic 6, 1st Edition*. New Delhi: BPB Publications.

NOTES

Deitel, Harvey M., Paul J. Deitel and T. Tem R. Nieto. 1999. Visual Basic 6: How to Program. New Jersey: Prentice-Hall.

Norton, Peter. 1998. Peter Norton's Complete Guide to Visual Basic 6. New Delhi: Techmedia.

Reselman, Bob and Richard A. Peasley. 1998. Using Visual Basic 6. New Jersey: Pearson Education (Que Publishing).

Donald, Bob and Oancea Gabriel. 1999. Visual Basic 6 from Scratch. New Delhi: Prentice-Hall of India.

BLOCK IV

MENUS AND MDI FORMS

UNIT 10 MENUS

NOTES

- 10.0 Introduction
- 10.1 Objectives
- 10.2 Menus
 - 10.2.1 Using the Visual Basic Application Wizard
 - 10.2.2 Using the Visual Basic Menu Editor
- 10.3 Answers to Check Your Progress Questions
- 10.4 Summary
- 10.5 Key Words
- 10.6 Self-Assessment Questions and Exercises
- 10.7 Further Readings

10.0 INTRODUCTION

The main objective in designing a program is to enhance the ease of use of the features provided. Well-designed menus aid in accomplishing this objective. Menus are perhaps the most noticeable feature of an application. Programmers cannot always know what standard features they might wish to utilize in an application until they have reached a much advanced stage in the development process. Thus, while using the Application Wizard, it is preferable to select every program that might be used. It can always be deleted later. The main advantage offered by the Visual Basic (VB) Application Wizard is that it facilitates the making of menus that already have the standard features of Windows loaded in them.

The primary aim of Microsoft is to standardize the Key Windows features across all Windows Graphical User Interface (GUI). This implies that regardless of the program in use – MS Access, MS Excel or MS Word – the Key features will always appear and work similarly. For example, the manner of accessing the printer feature is the same across all programs in the entire platform.

As you have worked with menus of various GUI environments, such as Windows-XP or MS Word, you already know that there are generally two types of menus: a horizontal menu known as menu bar and a vertical menu known as pop-up menu or pull-down menu or simply submenu.

In this unit, you will study about the menus, create menus and adding a code for menus in Visual Basic.

NOTES

10.1 OBJECTIVES

After going through this unit, you will be able to:

- Discuss the significance of menus in Visual Basic programming
- Use the Visual Basic Application Wizard and Menu Editor
- Build simple menu with the application wizard
- Create complex menus and toolbars in Visual Basic (VB)
- Define the Pop-Up menu
- Explain Toolbar styles

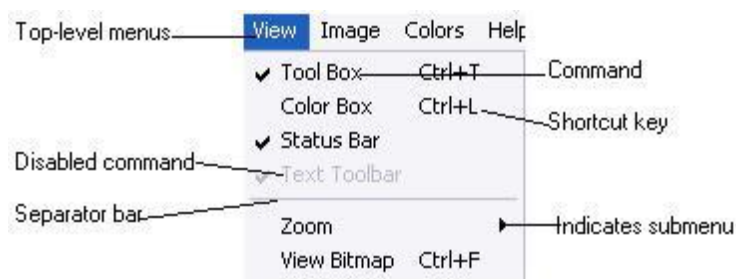
10.2 MENUS

Adding well-built menus benefits all programs that have multiple features and perform more than just a few simple functions. The key objective, while designing a program, is to enhance the ease of use of the features provided. Well-designed menus aid in accomplishing this objective. Menus are perhaps the most noticeable feature of an application.

Forms and controls constitute the basic interface for creating an application. However, you can make your program more user-friendly by adding menus to them. Menus are a conventional and consistent way of grouping commands so that they are readily and easily accessible to users.

The primary aim of Microsoft is to standardize the key Windows features across all Windows Graphical User Interface (GUI). This implies that regardless of the program in use – Access, Excel or Word – the key features will always appear and work similarly. For example, the manner of accessing the printer feature is the same across all programs in the entire platform.

As you have worked with menus of various GUI environments, such as Windows-XP or MS Word, there are generally two types of menus: a horizontal menu known as **Menu Bar** and a vertical menu known as **Pop-Up Menu** or **Pull-Down Menu** or simply **Submenu**.



Although you are familiar with various components of menus, let us summarize them once again.

Table 10.1 Windows Standard Menu Conventions

Feature	Convention
Menu Bar	The horizontal bar containing different menu options. Each menu option may have another submenu attached to it.
Pop-Up Menu or Pull Down Menu Submenu	The vertical menu containing different menu options. A menu attached to a menu item.
Separator Bar	A bar on a menu that divides menu items into logical groups.
Caption	One or more short specific words to describe a command.
Organization	Menu items must be grouped logically according to their function and a minimal number of levels should be allowed for accessing each feature.
Access Keys	All menu items must be assigned an access key (the letter that is underlined in a menu or menu selection) that allows the menu choices to be accessed through the keyboard. These keys must, as a rule, be unique in every section of the menu and are usually the first letter of the caption.
Shortcut Keys	Shortcut keys should be assigned to menu features that are used frequently or that need to be accessible from any part of the program. A particular shortcut key can only be assigned to a single menu item.
Check Box	A checked feature should directly be assigned in the menu for each menu item that simply sets or clears a single program option.
Ellipsis	Menu items opening dialogs must be succeeded by ellipses (...).

NOTES

You can see that there are some grey menu items as well. These menu items cannot be selected.

After talking about different components of menus, let us learn to create menus in Visual Basic (VB). There are two different ways to create menus in VB.

10.2.1 Using the Visual Basic Application Wizard

With the improvements Visual Basic (VB 6.0) has brought to the Application Wizard, fully customized menus can now be created directly in the menus. This is different

from the previous versions of VB, in which the standard menu options that could be included in the application were limited to a select few.

Application Wizard Options

NOTES

Programmers cannot always know what standard features they might wish to utilize in an application until they have reached a much later stage in the development process. Therefore, while using the Application Wizard, it is preferable to select every program that might be used. It can always be deleted later.

The chief advantage offered by the VB Application Wizard is that it facilitates the making of menus that already have the standard features of Windows loaded in them. The user merely has to select the required features from a pre-given template. Further, they come arranged in the Windows standard layout.

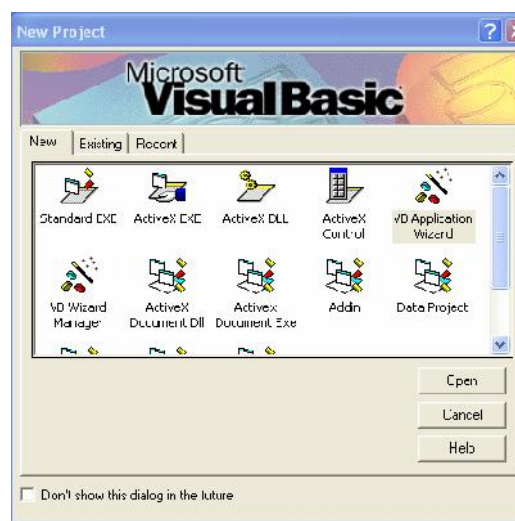
Limitations of the Application Wizard

The Application Wizard cannot be used to make modifications in existing projects. Other third-party support programs are available that append additional functionality to the existing project without necessitating further programming. However, add-ins, such as the Menu Editor, that are included in Visual Basic (VB) can be used.

The Application Wizard functions as a tool that builds new applications. This implies that it is used to build a functional application shell that has the standard features. If additional features are required, they have to be programmed or the Menu Editor can be used. Once Finish is clicked in the Application Wizard, and the base program gets generated, any modification in the program will have to be made using the Menu Editor.

Building a Simple Menu with the Application Wizard

1. Start the Application Wizard through the default dialog that opens when Visual Basic (VB 6.0) starts or select New Project from the File menu.



New Project Dialog Box

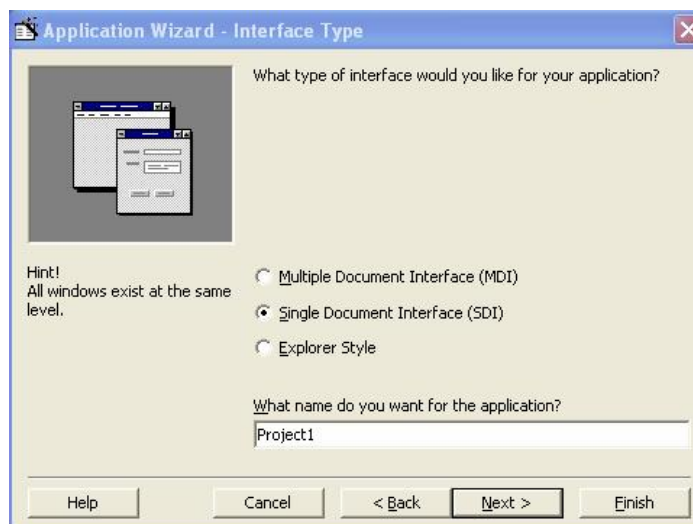
2. Double-click the Application Wizard Icon to start the Wizard.
3. The Application Wizard Introduction Dialog Box will appear, which will allow you to reuse the answers you saved during a previous Application Wizard session. Ignore the default choice and click Next.

NOTES



Application Wizard Introduction Dialog

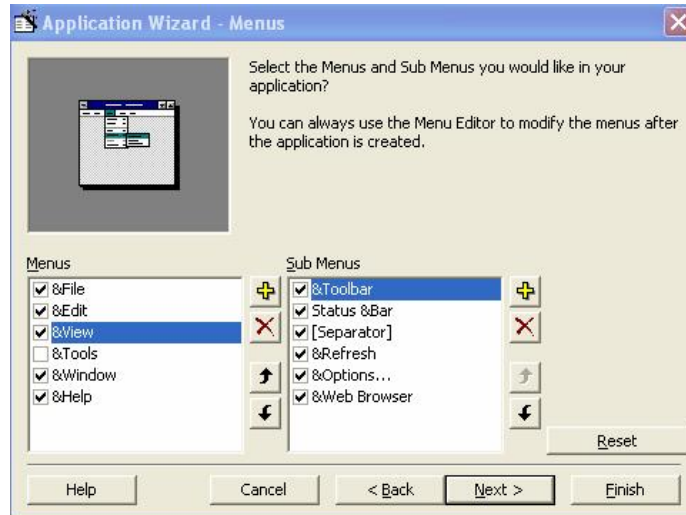
If your settings are saved in the Application Wizard, then you need not re-enter the choices that you have already made, and this saves valuable time. The Toolbar and Menu settings are saved by the Application Wizard in a profile file (.RWP). Not only can a consistent feel and look be achieved for your programs through such profiles, but duplication of effort in building menus can also be reduced. This is particularly helpful for large assignments that involve multiple teams of developers working on disparate elements of a single huge application, as it provides a common starting point.



Application Wizard Interface Type Dialog

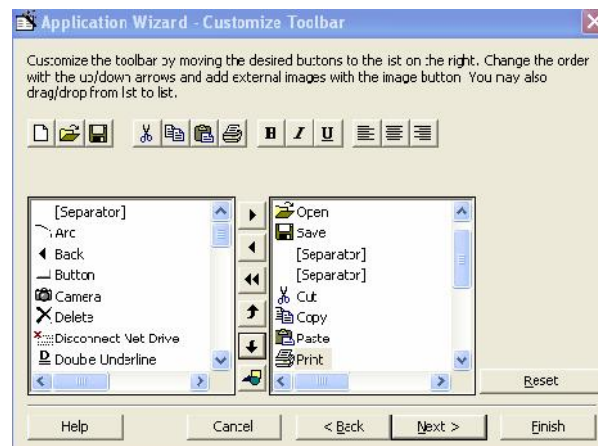
NOTES

4. In the Interface Type dialog, select the type for the initial application screen. For this sample application, select Single Document Interface (SDI). Ignore the default project name and click Next.



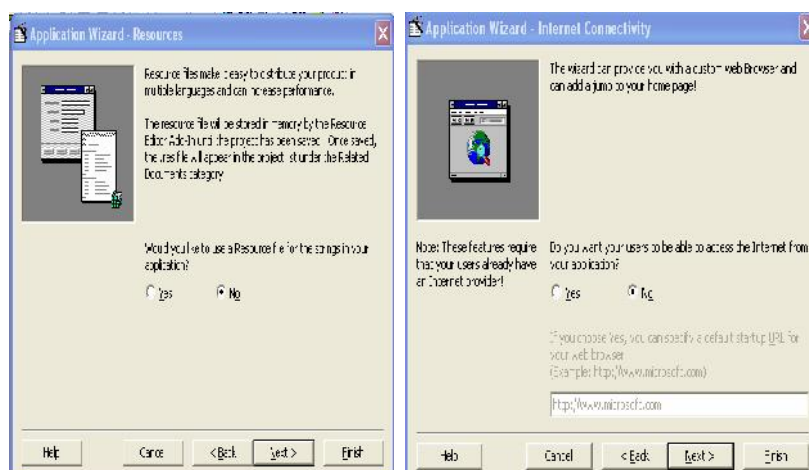
Window Standard Default Menu

5. A Windows standard default menu is created for you to begin to modify. Not all possible choices are selected in the initial menu. After you make your modifications, click Next.
6. The Application Wizard allows a browser, toolbar, resource file, database connectivity and other templates to be customized. For this example, skip these dialogs and click Next five times to get to the final dialog.



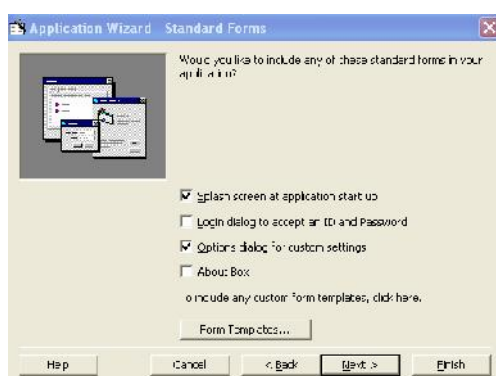
Customize Toolbar

7. In the last dialog, save your profile. Enter a name for your profile that relates to your application. Then, click Finish to complete the Application Wizard. (If you click Finish in any of the previous dialog boxes, you will be unable to save the profile for future use.)

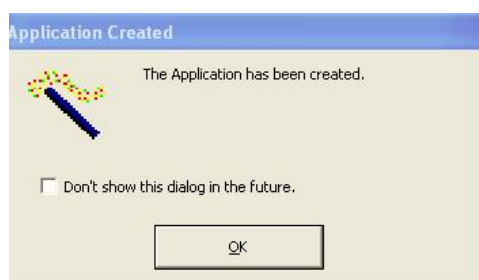


Profile—Resources

Profile—Internet Connectivity



Profile—Standard Forms



The Created Application

10.2.2 Using the Visual Basic Menu Editor

To add menus to a VB application, invoke the **Menu Editor**. The **Menu Editor** of Visual Basic (VB) is an interactive way to create and modify a menu and that too with minimal coding. With it you can even create shortcut menus.

The Menu Editor is used for building menus with VB programs. It is visible as an icon on the Visual Basic Integrated Development Environment (VB IDE) toolbar.

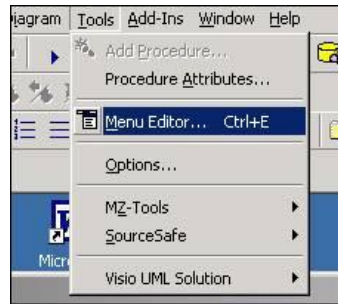
NOTES

NOTES



Menu Editor Icon in Toolbar

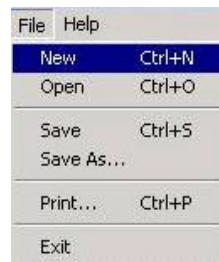
Alternatively, it can also be invoked from the **Tools** menu as shown below:



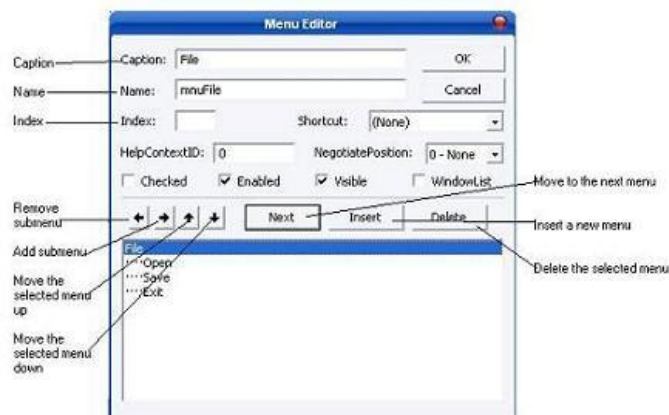
Invoking the Menu Editor from Tools Menu Item

Building a Menu

The following steps must be carried out to build a menu:



1. Start a new Visual Basic (VB) project and open the Menu Editor using either of the two methods mentioned above. The Menu Editor screen will appear as follows:



Menu Editor Screen

- In the 'Caption' column, type **&File** (locating the ampersand on the left hand side of 'F' establishes 'F' as an access key for the File item. This means the user can drop down the File menu by pressing 'Alt+F' or by clicking the 'File' item with the mouse).

In the 'Name' column, type **mnuFile**.

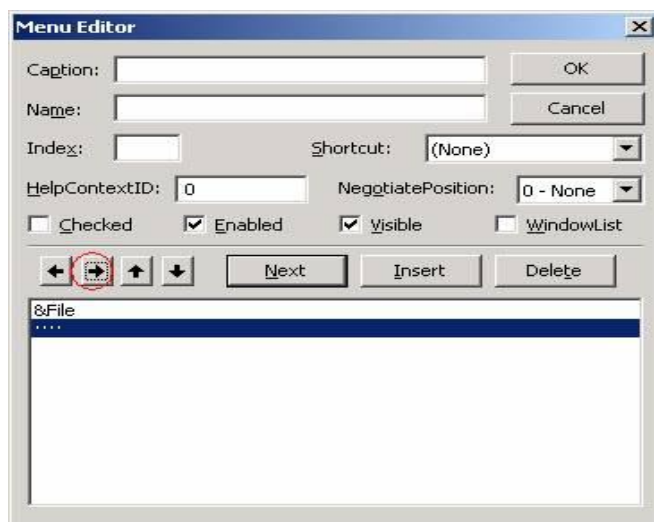
The following screenshot shows what your Menu Editor will appear as:



Adding Caption and Name in the Menu Editor

Click Next.

- Clicking the 'right-arrow' button (shown circled below) will make ellipses (...) appear as the next item in the menu list. This indicates that it is a level-two item (below 'File').



Showing Ellipses as the Next Item

NOTES

NOTES

In the 'Caption' column, type **&New**; in the 'Name' column, type **mnuNew** and in the 'Shortcut' roll down list, select **Ctrl+N**. Specifying a shortcut allows the user to access the related menu item by pressing that particular combination of keys. So the user is provided with three options with which to invoke the 'New' function: (i) clicking New on the File menu (ii) pressing Alt+F,N (because an access key is set up for N by locating an ampersand on the left hand side of 'N' in 'New') and (iii) pressing Ctrl+N. The following screenshot shows what the screen will look like:



Specifying a Shortcut

Click Next.

4. In the 'Caption' column, type **&Open**; in the 'Name' column, type **mnuOpen** and in the 'Shortcut' roll down list, select **Ctrl+O**. The following screenshot shows what your screen will look like at this point:



Changing Caption, Name and Shortcut

Click Next.

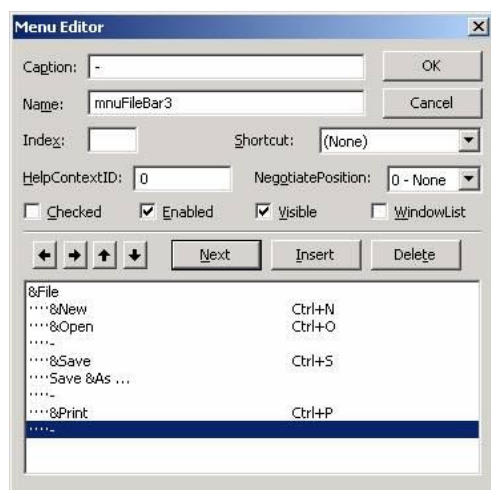
5. In the 'Caption' column, type - (hyphen) and in the 'Name' column, type **mnuFileBar1**. The hyphen provides for a separator bar to be created at that point. The following screenshot shows what the screen will appear as at this juncture:



Using Single Hyphen as Caption to Create a Separator Bar

Click Next.

6. In the 'Caption' column, type **&Save**; in the 'Name' column, type **mnuSave** and in the 'Shortcut roll down list', select **Ctrl+S**. Replicate this step for the rest of the menu items.
7. In the 'Caption' column, type - (hyphen) and in the 'Name' column, type **mnuFileBar3**. The following screenshot shows what you screen should appear as:



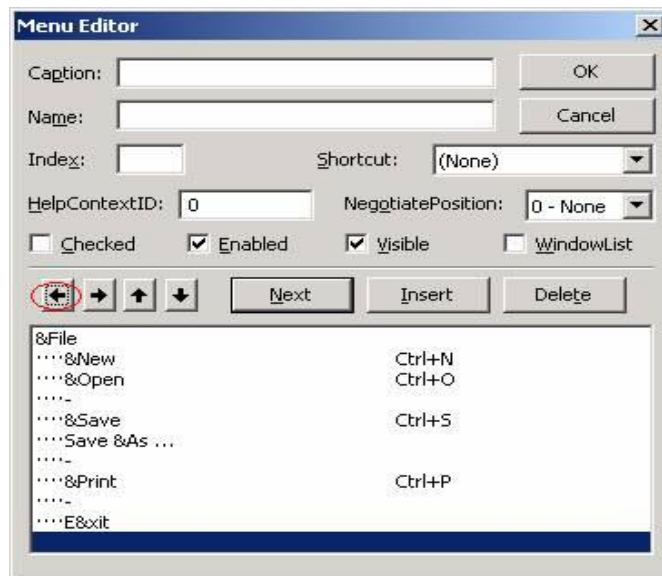
Using Hyphen as Caption

Click Next.

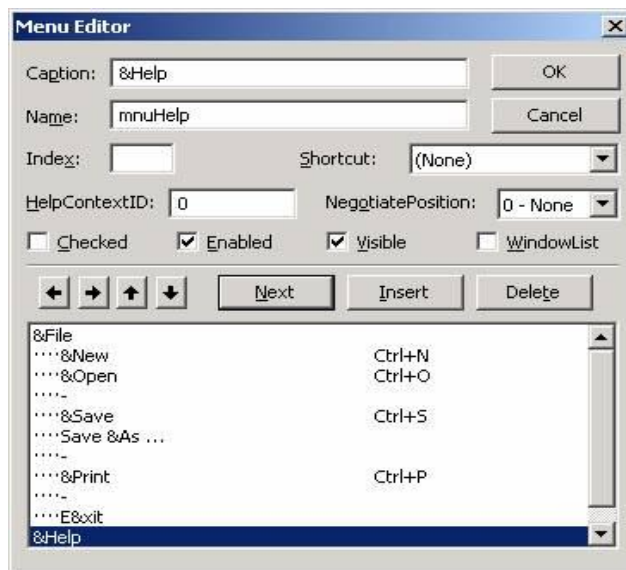
NOTES

NOTES

8. Click the 'left-arrow' button (shown circled below). The fact that ellipses (...) are not visible any more means that we are back to the top-level items.

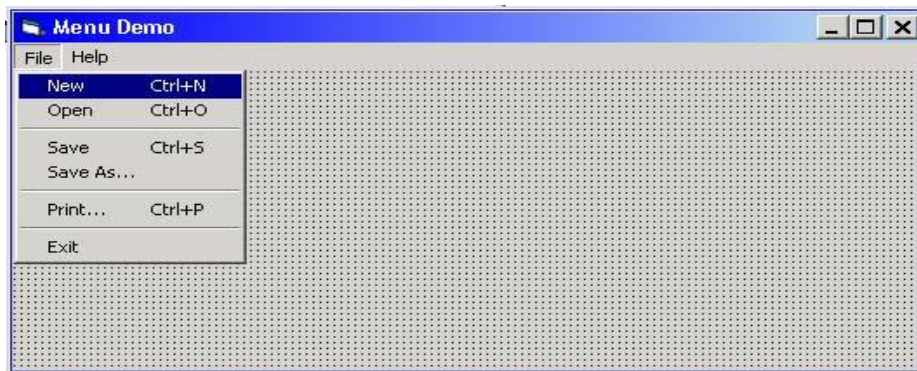


In the 'Caption' column, type **&Help** and in the 'Name' column, type **mnuHelp**. The following screenshot shows is what your screen should appear as:



Click Next.

9. This completes the creation of our menu entries. Click **OK**. This will close the Menu Editor and return to Visual Basic Integrated Development Environment (VB IDE).
10. In VB IDE, you will see your form having a menu depending on your Menu Editor set up. Clicking on a top-level menu item (**File**, for example) will drop down the level-two menu:



NOTES

11. Click on **New**. The code window for the **mnuFileNew_Click** event will open, as shown in the following figure:

Note: The single event that a menu item responds to is a **Click**.



In the Place **mnuFileNew_Click** event, place the code you want to be executed when **New** is clicked. For our example, we will place a simple MsgBox statement in the event procedure:

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



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

```
Private Sub mnuFileOpen_Click()
    MsgBox "Code for 'Open' goes here.", vbInformation,
    "Menu Demo"
End Sub

Private Sub mnuFileSave_Click()
    MsgBox "Code for 'Save' goes here.", vbInformation,
    "Menu Demo"
End Sub

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

NOTES

```
Private Sub mnuFilePrint_Click()
    MsgBox "Code for 'Print' goes here.", vbInformation,
    "Menu Demo"
End Sub
```

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

```
Private Sub mnuFileExit_Click()

    Unload Me

End Sub
```

14. Run the program. Observe the execution of the code as the various menu items are clicked. Test the use of the shortcut keys (e.g., **Ctrl-O**) and access keys (e.g., **Alt+F, N**) as well.

15. Save the program and exit VB.

Creating Pop-Up Menus

Pop-Up Menus are floating menus that are displayed over forms, independent of the menu bar.

The items will be displayed on the Pop-Up Menu depending on the position of the pointer when the right mouse is clicked. This is the reason why Pop-Up Menus are also known as context menus. In Windows 9x, menus are activated by clicking the right mouse button.

For a Pop-Up Menu to be created, a menu has to be first defined through the Menu Editor. In order to make certain that this menu is not displayed on the Menu Bar, it is made invisible, that is, the visible check box in the Menu Editor is kept unchecked for this menu.

It is possible to display any menu with a minimum of one menu item at run-time as a Pop-Up menu. The **PopupMenu** method can be used for displaying Pop-Up Menus. The syntax for this method is as follows:

Syntax:

```
PopupMenu <menuname>
```

For example, the menu name mnuFile is displayed by the following code when a user clicks a form with the right mouse button. The MouseUp or MouseDown events too can be used for detecting when a user clicks the right mouse button:

```
Private Sub Form_MouseUp(Button As Integer, Shift As
Integer, X As Single, Y As Single)
    If Button = 2 Then          ' Button =2 indicate the right
mouse button is clicked.
```



```
PopupMenu mnuFile
```

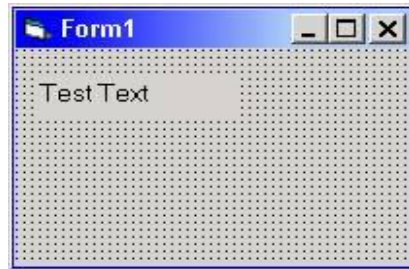
```
End If
```

```
End Sub
```

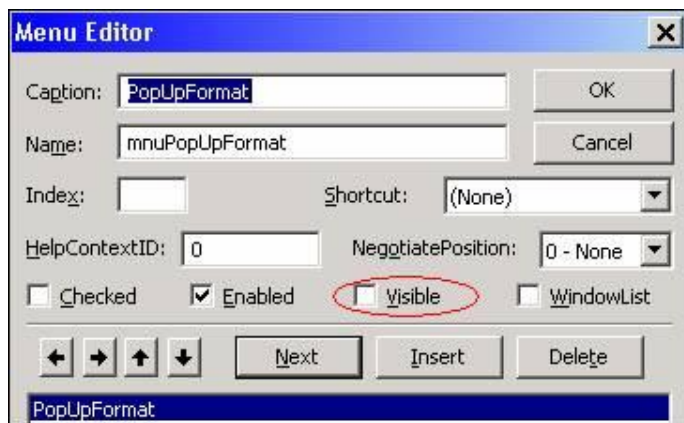
Menus

The following example illustrates how to create a **Pop-Up Menu** (sometimes called a **right-click Menu** or a **Context Menu**).

1. Begin a new Visual Basic (VB) project and place a label on the form. Give the name **lblTestText** to the label. Give it the Caption **Test Text**.



2. Open the Menu Editor, and create a top-level item with a Caption value of **PopUpFormat** and the Name **mnuPopUpFormat**. Also—importantly—**uncheck** the **Visible** checkbox (see the circled item below). A menu has to be invisible if it must be a Pop-Up Menu.



Unchecking the Visible Checkbox

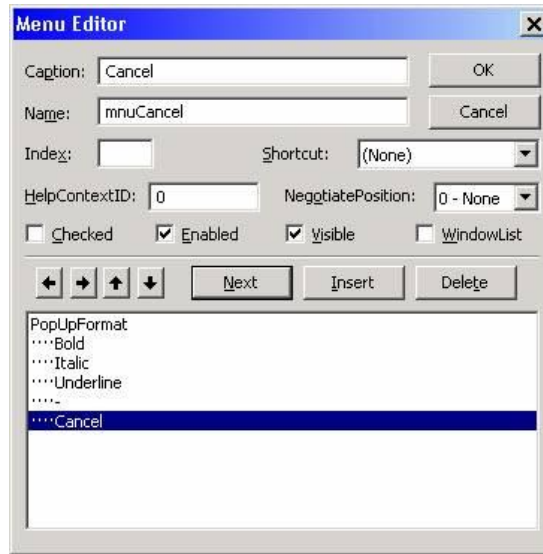
3. Create the following level-two menu items below the Pop-UpFormat top-level menu. (The Visible box must be kept checked when these level-two items are created.)

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

NOTES

This is what your Menu Editor will appear as when you have finished:

NOTES



4. Click **OK** to save your changes.

Note: When you return to Integrated Development Environment (IDE), you will not be able to see this menu on the form (Because it is a pop-up menu, it will be visible only when it is invoked through the code).

5. Code the **lblTestText_MouseDown** event as follows. Note that the **Button** parameter is tested for **vbRightButton**; as is conventional; we wish the menu to pop up only if the user right-clicks the label. If the right mouse button is clicked, the **Pop-Up Menu** statement will be executed. This statement will make the Pop-Up Menu appear.

```
Private Sub lblTestText_MouseDown(Button As Integer,
```

```
—
```

```
    Shift As Integer, _
```

```
    X As Single, _
```

```
    Y As Single)
```

```
    If Button = vbRightButton Then
```

```
        PopupMenu mnuPopUpFormat, vbPopupMenuRightButton
```

```
    End If
```

```
End Sub
```

5. Code the **mnuBold_Click** event as shown below. Observe the use of the **Checked** property of the menu item. A checkmark appears on the left side of the menu item when it is set to True. The Checked property is typically used as a toggle.

```
Private Sub mnuBold_Click()
```

```
    If mnuBold.Checked Then
```

```
        lblTestText.FontBold = False
```

```
        mnuBold.Checked = False
```

```

Else
    lblTestText.FontBold = True
    mnuBold.Checked = True
End If
End Sub

```

6. Code the **mnuItalic_Click** and **mnuUnderline_Click** events in a similar fashion as follows:

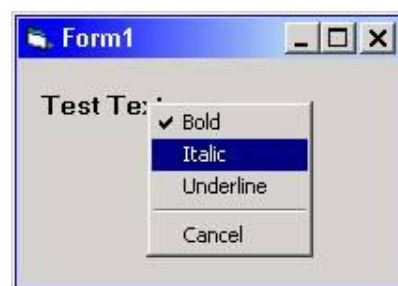
```

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

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

```

7. Run the program and look at the several options you have coded.



8. Save the program and exit VB.

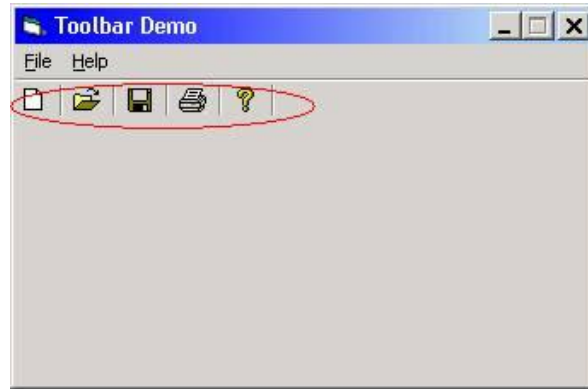
Note: If you so desire, it is possible to have both a ‘regular’ menu and several pop-up menus on the same form. Wherever the Visible box of a top-level menu is checked in the Menu Editor, it will appear at the top of the form in the menu bar that is created. If this box is left unchecked, then that particular top-level menu will not appear at the top of the form in the menu bar, but the **Pop-Up Menu** method can be used to invoke it as a Pop-Up Menu.

NOTES

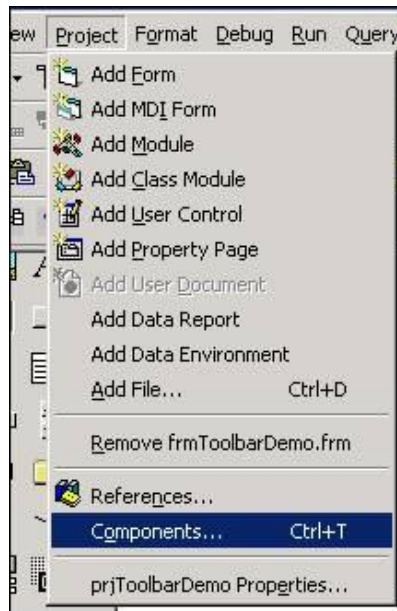
NOTES

Adding a Toolbar

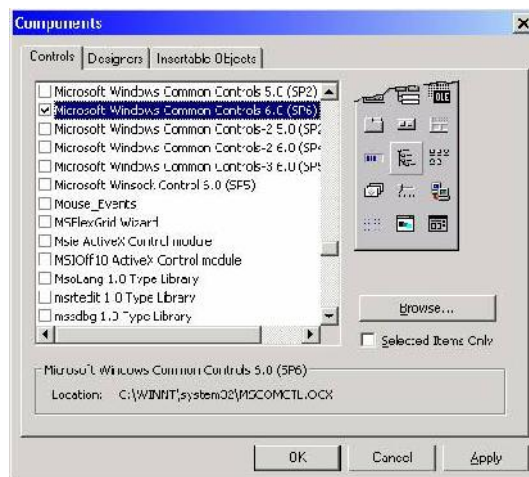
A Toolbar provides the user with quick access to the more commonly used functions of a program. It either complements a program's menu structure or functions as a stand-alone. A Toolbar control consists of an assortment of button objects that a user can use to create toolbars that can easily be associated with an application.



1. Two controls are required to create a Toolbar in VB: the **ImageList** control, containing the images that are to be used for the Toolbar buttons, and the **Toolbar** itself. These controls form two of the nine controls of **Microsoft Windows Common Controls**. You can make these controls accessible to your VB project by going to the **Project** menu in Visual Basic Integrated Development Environment (VB IDE) and selecting **Components**.

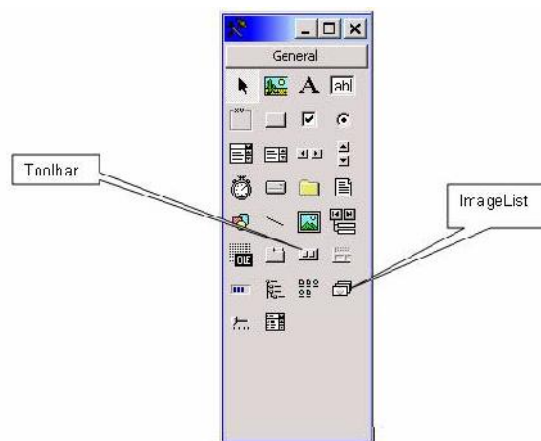


The Components Dialog Box appears. Check **Microsoft Windows Common Controls 6.0 (SP6)** here.



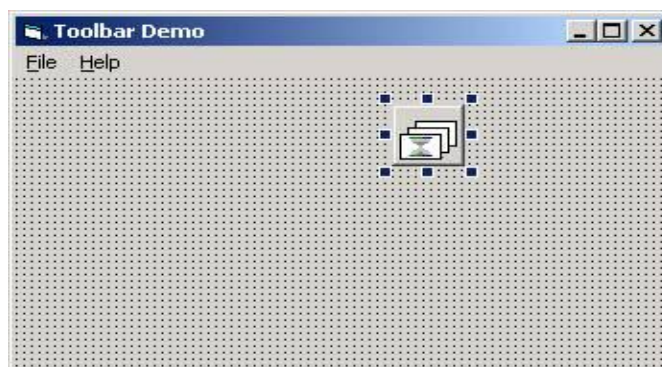
NOTES

Click the **OK** button. **ImageList** and **ToolBar** will appear in your toolbar along with **TreeView**, **ListView**, **Slider**, **StatusBar**, **ProgressBar**, **TabStrip** and **ImageCombo**.



Toolbox with the Windows Common Controls Added

2. Double-click the **ImageList** control to bring it onto the form.

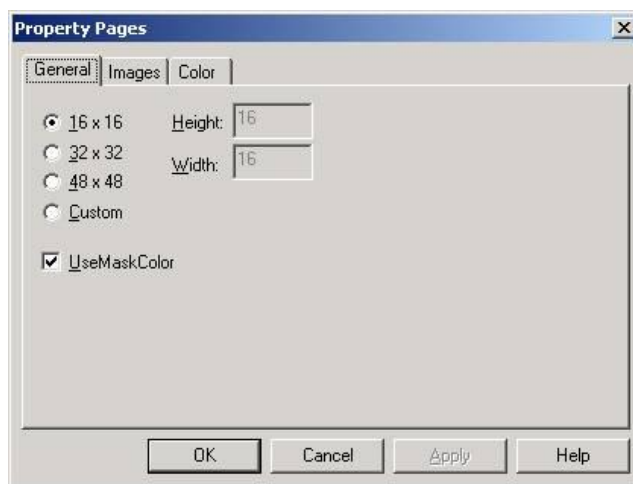


NOTES

The ImageList control is not visible at run-time and cannot be accessed directly by the user. The objective of the ImageList control is to provide a repository for images that are used by a number of other controls. ImageList is largely used with the intent of supplying images to other controls such as ListView and TreeView and to the Toolbar. Images are added to the ImageList through the design environment (it can also be done in code), then the index or the key of the image is referenced to be used in other controls.

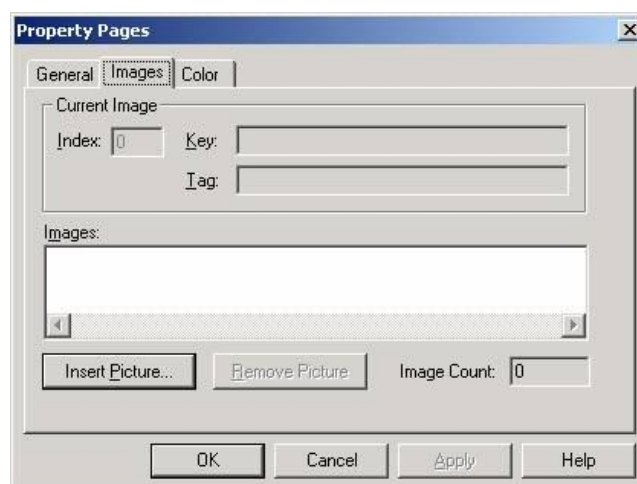
Select ImageList, then click **F4** to bring up the property list and give the ImageList the name **imlToolbarIcons**.

3. Select the ImageList control on the form, right-click it and choose Properties from the Context Menu. This displays the Property Pages of ImageList:



On the **General** tab, select **16 × 16**.

4. Click on **Images**. The following screenshot shows what the Property Pages dialog will appear as:



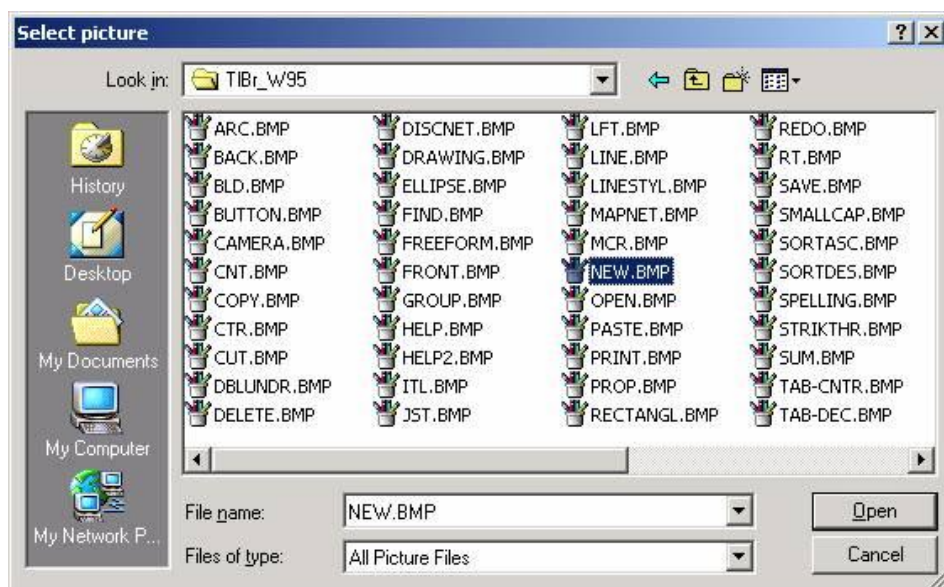
The next step is to click the **Insert Picture** button repeatedly, choose an image and set the Key property until all the desired images have been loaded.

When Insert Picture is clicked for the first time, the Select picture dialog box will appear. Navigate to the following folder:

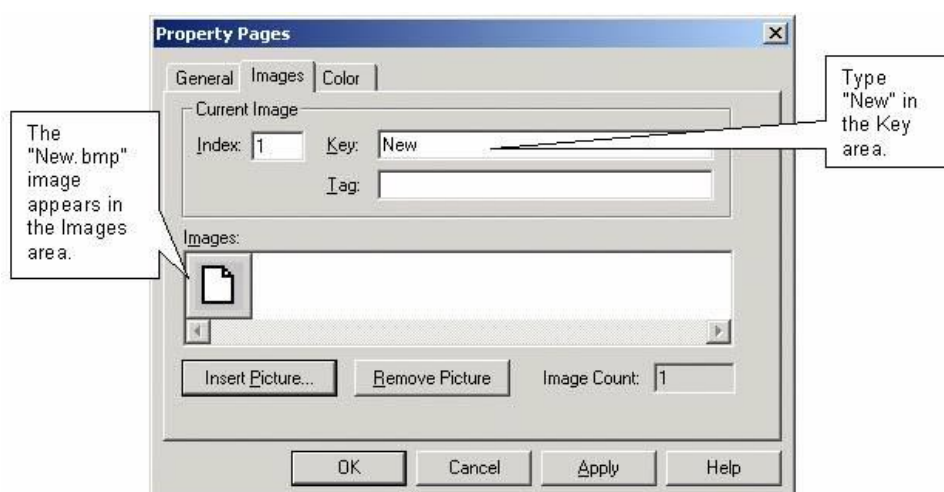
\Program Files\Microsoft Visual Studio\Common\Graphics\Bitmaps\TIBr_W95.

Various standard Toolbar images are contained in this folder. For the first image, click **New.bmp** as shown in the following screenshot:

NOTES



Click on **Open** to select the image and dismiss the 'Select picture' dialog box (double-clicking the 'New.bmp' file has the same effect). Now, the image will appear in the Images area of the Images tab of the Property Pages as shown in the following screenshot. Type **New** in the Key area.



Repeat the above process as you add the following images (type 'Open', 'Save', 'Print' and 'Help', respectively, for Key):

Open.bmp
 Save.bmp
 Print.bmp
 Help.bmp

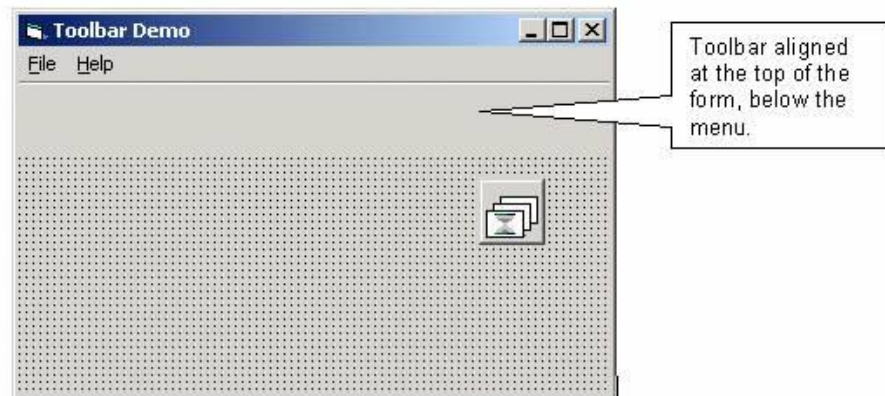
NOTES

The Property Pages dialog will appear as follows:

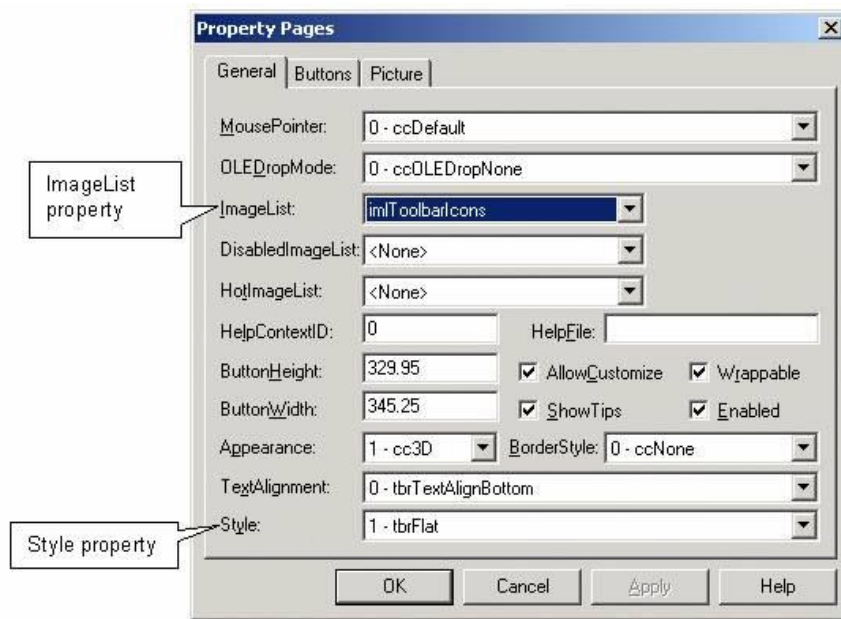


Here, click **OK** to dismiss the Property Pages dialog.

5. The task is to bring the Toolbar control onto the form. Double-click the Toolbar control on the toolbox. By default, the Toolbar will be aligned with the top of the form, below the menu, if you have one.





6. Keeping the Toolbar selected, press **F4** to bring up the regular properties window and set the (Name) property to **tbrMenuOpts**. Following this, right-click the toolbar and select Properties to bring up the Property Pages dialog. On the **General** tab, set **ImageList** to 'imlToolbarIcons' (the ImageList that was added to this form)—remember, the toolbar must get its images from the images stored in an ImageList control. Also, set the **Style** property to '**1 – tbrFlat**' (the default is '0–tbrStandard').



NOTES

Table 2.2 explains the difference between the Standard and Flat Toolbar Styles:

Table 2.2 **Standard versus Flat Toolbar Styles**

Standard	Flat
	
<p>In the Standard style, which is also the classic, older style, the buttons of the toolbar appear raised at all times. This was the only style available up through VB 5.0.</p>	<p>With the Flat style, the toolbar buttons remain flat. When you pass over one of the buttons with the mouse, that button will appear raised. This was the prevalent style in the Microsoft products of the late 1990s.</p>

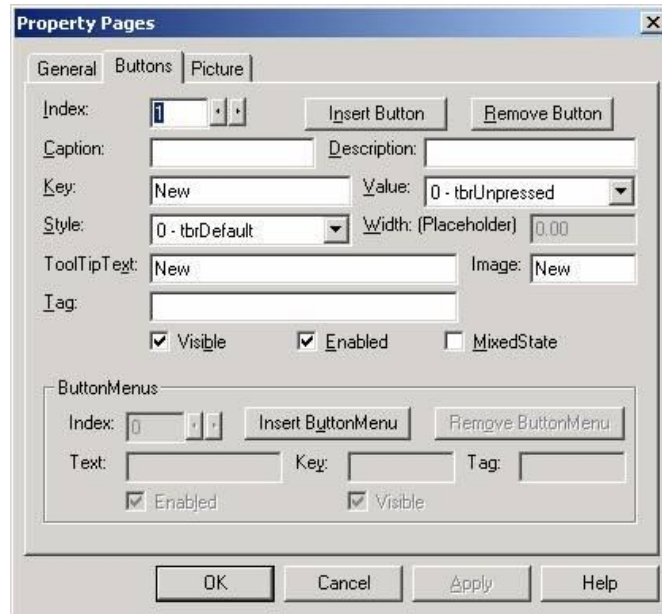
7. On the **Buttons** tab of the Property Pages dialog, click **Insert Button** and set the following properties:

- Set the **Key** property to 'New'. The Key property of a toolbar button is a string that distinctively classifies that particular button among the many buttons of a given toolbar. In the sample program given here, the Key property has been used to make out which button has been pressed by the user (the Index property too can be used to identify the button, but it is more difficult to do so this way).
- Set the **ToolTipText** property to 'New'. This is text that pops up in a little yellow label when the mouse is moved over the button.
- Set the **Image** property to '1' or 'New'. This is how an image is tied in the ImageList control to a button on the Toolbar. The value that is

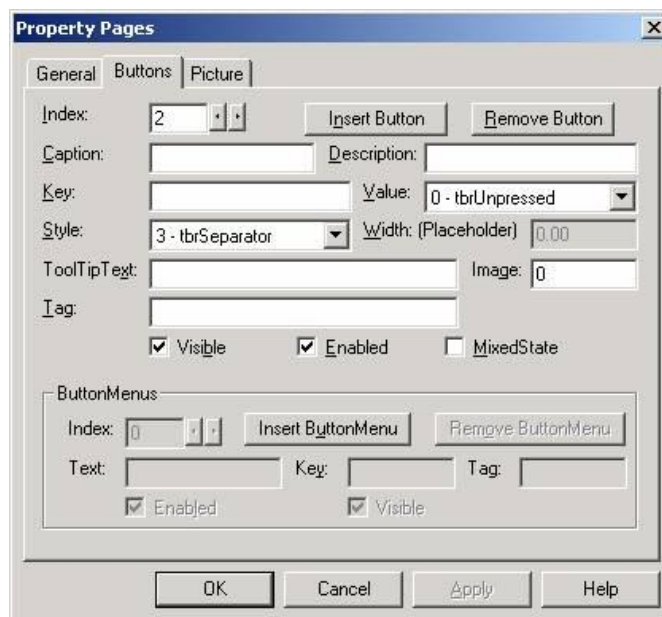
typed for the Image property here refers to either the Index or the Key property of the desired image in the ImageList control.

The Property Pages dialog box should appear as follows:

NOTES



Click the 'Insert Button' button again. Ignore the Key, ToolTipText and Image properties and set the **Style** property to '3-tbrSeparator'. If you look back at the screenshot showing the Flat Toolbar style, you will notice a vertical line separating each button in the Toolbar. This is achieved by setting the **Style** property of every other button to '**3-tbrSeparator**'. The following screenshot shows what the Property Pages Dialog Box will appear as:



To finish setting up the toolbar, set the following properties upon each click of **'Insert Button'**:

Index (do not type in—VB will automatically supply these)	Key	Style	ToolTipText	Image
3	Open	0—tbrDefault	Open	2 (or 'Open')
4		3—tbrSeparator		
5	Save	0—tbrDefault	Save	3 (or 'Save')
6		3—tbrSeparator		
7	Print	0—tbrDefault	Print	4 (or 'Print')
8		3—tbrSeparator		
9	Help	0—tbrDefault	Help	5 (or 'Help')
10		3—tbrSeparator		

NOTES

When you complete the setting up of the Buttons, click **OK** to dismiss the Property Pages dialog box.

- The final task now is to code the Click event for the Toolbar that will test which button the user presses and accordingly call the corresponding menu click event procedure (the menu events have already been coded). For instance, if the the Open button on the Toolbar is clicked, the **mnuFileOpen_Click** event procedure will have to be called. Double-clicking the toolbar will open the code window with the **ButtonClick** event for the Toolbar, which is the event we want to code for. The Sub Procedure header will look as follows:

```
Private Sub tbrMenuOpts_ButtonClick(ByVal Button As
MSComctlLib.Button)
```

A button object is passed by VB to the event procedure, which represents the Toolbar button that the user clicks. To test which button has been clicked by the user, the Index or Key property of the button object must be tested (using a standard *object.property* syntax, like 'Button.Index' or 'Button.Key'). In the sample program, 'Button.Key' is used. The coding is as follows:

```
Private Sub tbrMenuOpts_ButtonClick(ByVal Button As
MSComCtlLib.Button)
Select Case Button.Key
Case "New"
mnuFileNew_Click
Case "Open"
mnuFileOpen_Click
```

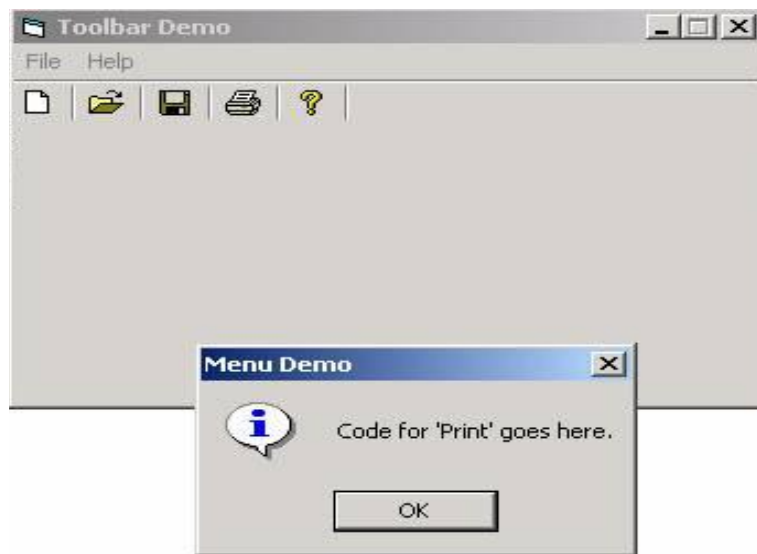
NOTES

```

Case "Save"
mnuFileSave_Click
Case "Print"
mnuFilePrint_Click
Case "Help"
mnuHelpAbout_Click
End Select
End Sub

```

9. Run the program. When a Toolbar button (such as **'Print'**) is clicked, the corresponding menu procedure code will be executed:



10. Once you are done, save the program and exit VB.

Check Your Progress

1. Define the term menu.
2. How many types of menus are in Visual Basic?
3. What is separator bar in menus?
4. Elaborate on grey menu items.
5. Give the definition of Profile File.
6. How the menus are added to a Visual Basic application?
7. Explain about the Pop-Up menus.
8. What are the function of a Toolbar?
9. Elaborate on the purpose of the ImageList control.

10.3 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Menus are perhaps the most noticeable feature of an application.
2. Generally, there are two types of menus: horizontal menu, known as menu bar, and vertical menu, known as Pop-Up menu or Pull-Down menu or simply Submenu.
3. A bar on a menu that divides menu items into logical group.
4. The menu items that are disabled and cannot be selected are known as grey menu items.
5. Profiles provide the basis for a consistent look and feel to your programs while reducing duplicate effort in building the menus. This allows for a common starting point in the development of an application, which is helpful in large projects in which many teams of developers might be working on separate parts of a single large application.
6. To add menus to a Visual Basic application, invoke the Menu Editor.
7. A Pop-Up menu is a floating menu that is displayed over a form, independent of the menu bar. The items displayed on the Pop-Up menu depend on where the pointer was located when the right mouse button was pressed; therefore, Pop-Up menus are also called Context menu.
8. The Toolbar provides the user quick access to the most commonly used functions of a program. A Toolbar can be used as a stand-alone or as a complement to the program's menu structure. A Toolbar control contains a collection of button objects used to create a ToolBar that you can associate with an application.
9. The purpose of the ImageList control is to provide a repository for images that are used by several other controls. The ImageList must be used to supply images to the toolbar as well as to other controls, such as the TreeView and ListView.

NOTES

10.4 SUMMARY

- Adding well-built menus benefits all programs that have multiple features and perform more than just a few simple functions.
- The chief objective, while designing a program, is to enhance the ease of use of the features provided. Well-designed menus aid in accomplishing this objective. Menus are perhaps the most noticeable feature of an application.
- Forms and controls constitute the basic interface for creating an application.
- Menus are a conventional and consistent way of grouping commands so that they are readily and easily accessible to users.

NOTES

- The primary aim of Microsoft is to standardize the chief Windows features across all Windows Graphical User Interface (GUI).
- This implies that regardless of the program in use – Access, Excel or Word – the chief features will always appear and work similarly. For example, the manner of accessing the printer feature is the same across all programs in the entire platform.
- As you have worked with menus of various Graphical User Interface (GUI) environments, such as Windows- XP or MS Word, you already know that there are generally two types of menus: a horizontal menu known as menu bar and a vertical menu known as Pop-Up menu or Pull-Down menu or simply Submenu.
- The horizontal bar containing different menu options. Each menu option may have another Submenu attached to it.
- A bar on a menu that divides menu items into logical groups.
- Menu items must be grouped logically according to their function and a minimal number of levels should be allowed for accessing each feature.
- All menu items must be assigned an access key (the letter that is underlined in a menu or menu selection) that allows the menu choices to be accessed through the keyboard.
- Shortcut keys should be assigned to menu features that are used frequently or that need to be accessible from any part of the program. A particular shortcut key can only be assigned to a single menu item.
- A checked feature should directly be assigned in the menu for each menu item that simply sets or clears a single program option.
- With the improvements Visual Basic 6.0 (VB 6.0) has brought to the Application Wizard, fully customized menus can now be created directly in the menus.
- This is different from the previous versions of Visual Basic (VB), in which the standard menu options that could be included in the application were limited to a select few.
- Programmers cannot always know what standard features they might wish to utilize in an application until they have reached a much later stage in the development process.
- Therefore, while using the Application Wizard, it is preferable to select every program that might be used. It can always be deleted later.
- The chief advantage offered by the Visual Basic (VB) Application Wizard is that it facilitates the making of menus that already have the standard features of Windows loaded in them.
- The user merely has to select the required features from a pre-given template. Further, they come arranged in the Windows standard layout.

- The Application Wizard cannot be used to make modifications in existing projects.
- Other third-party support programs are available that append additional functionality to the existing project without necessitating further programming.
- The Application Wizard functions as a tool that builds new applications.
- This implies that it is used to build a functional application shell that has the standard features.
- If additional features are required, they have to be programmed or the Menu Editor can be used.
- Once Finish is clicked in the Application Wizard, and the base program gets generated, any modification in the program will have to be made using the Menu Editor.
- Start the Application Wizard through the default dialog that opens when Visual Basic 6.0 (VB 6.0) starts or select New Project from the File menu.
- The Application Wizard Introduction dialog box will appear, which will allow you to reuse the answers you saved during a previous Application Wizard session.
- If your settings are saved in the Application Wizard, then you need not re-enter the choices that you have already made, and this saves valuable time.
- The Toolbar and menu settings are saved by the Application Wizard in a Profile File (.RWP).
- This is particularly helpful for large assignments that involve multiple teams of developers working on disparate elements of a single huge application, as it provides a common starting point.
- To add menus to a Visual Basic (VB) application, invoke the Menu Editor. The Menu Editor of Visual Basic (VB) is an interactive way to create and modify a menu and that too with minimal coding. With it you can even create shortcut menus.
- Pop-Up menus are floating menus that are displayed over forms, independent of the menu bar.
- The items will be displayed on the Pop-Up menu depending on the position of the pointer when the right mouse is clicked.
- This is the reason why Pop-Up menus are also known as Context menus. In Windows 9x, menus are activated by clicking the right mouse button.
- For a Pop-Up menu to be created, a menu has to be first defined through the Menu Editor.
- In order to make certain that this menu is not displayed on the menu bar, it is made invisible, and that is, the visible check box in the menu editor is kept unchecked for this menu.

NOTES

NOTES

- It is possible to display any menu with a minimum of one menu item at run-time as a pop-up menu. The Pop-Up menu method can be used for displaying Pop-Up menus.
- A ToolBar provides the user with quick access to the more commonly used functions of a program.
- It either complements a program's menu structure or functions as a standalone. A ToolBar control consists of an assortment of button objects that a user can use to create toolbars that can easily be associated with an application.
- Two controls are required to create a ToolBar in Visual Basic (VB).
- The ImageList control, containing the images that are to be used for the ToolBar buttons, and the Toolbar itself.
- The ImageList control is not visible at run-time and cannot be accessed directly by the user.
- ImageList is largely used with the intent of supplying images to other controls such as ListView and TreeView and to the toolbar.
- Images are added to the ImageList through the design environment (it can also be done in code), then the index or the key of the image is referenced to be used in other controls.
- A button object is passed by Visual Basic (VB) to the event procedure, which represents the ToolBar button that the user clicks.
- To test which button has been clicked by the user, the Index or Key property of the button object must be tested (using a standard object. property syntax, like 'Button.Index' or 'Button.Key'). In the sample program, 'Button.Key' is used.

10.5 KEY WORDS

- **Menus:** A conventional and consistent way of grouping commands so that they become readily and easily accessible to users.
- **Menu bar:** The horizontal bar containing different menu options.
- **Pop-up or pull-down menu:** A vertical menu containing different options.
- **Submenu:** A menu attached to a menu item.
- **Separator bar:** A bar on menu that divides menu items into logical groups on a menu.
- **Caption:** One or two short specific words to describe a command.
- **Access keys:** The keys that provide access to the menu choices.
- **Profile:** The feature that provides for a common starting point in the development of an application, which is helpful in large projects.

10.6 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What is Menu bar?
2. List the steps to create a Toolbar.
3. Why we use profile file in Visual Basic menus?
4. Define the term Pop-Up Menu.
5. Give the definition of Context Menu.
6. What is the significance short cut keys?
7. What will happen if two menu items are assigned the same access keys?

Long-Answer Questions

1. Create a new project with the following menu bar items
Listen, Read and Write. The Write submenu should have the following options: Pen, Pencil and Keyboard.
The Read submenu should have the following options: Magazine, Guide and Screen. The Listen submenu should have the following options: Television and Radio.
2. Insert menus to the Atm.vbp project that appears in the VB Samples\PGuide\Atm folder. On the opening form, add a File | Exit option and a Language menu bar option with the following pull-down checked choices: Spanish, French, Italian, German and English. Refrain from using special foreign characters unless they can be easily accessed from the keyboard and you are habituated to using them. When the application is first started by the user, checkmark should be put next to the English option but it should be moved when the user selects an option or clicks the corresponding command button. Add another menu to the Welcome form that includes a File | Exit option. Unlike the Welcome form's OK button, ensure that the application is completely terminated by the menu's File | Exit command on that form.
3. Create an application with a menu bar and a toolbar to create a text file, navigate and open it, edit it and save your changes.
4. Write down the various ways to open the Menu Editor Dialog Box and explain these ways.
5. Write down a program to create a menu named Numbers. The Numbers menu contains three options, Digits, Numbers and Boolean. The Boolean option has two sub options, True and False. Also assign a shortcut key, Ctrl+N, to the Numbers menu.
6. Write down a program, considering the question number 5, where you need to convert the two sub options of the Boolean option into Pop-Up menu.

NOTES

NOTES

7. John is a software application developer. His team leader assigns a task to create an application, in which whenever a user right-click the form, it should display the following options:
 - A. Copy
 - B. Paste
 - C. Delete
 - D. Properties
8. Write the steps to create a menu named Edit that contains four options:
 - A. Copy
 - B. Paste
 - C. Insert
 - D. Delete
9. Design a VB application that displays the following menus:

Student	View	Exit
New Student	Report Card	Topper's Details

10.7 FURTHER READINGS

- Cornell, Gary. 1998. *Visual Basic 6 from the Ground Up*. New Delhi: Tata McGraw-Hill.
- Warner, Scott L. 1998. *Teach Yourself Visual Basic 6*. New Delhi: Tata McGraw-Hill.
- Jerke, Noel. 1999. *Visual Basic 6 - The Complete Reference*. New York: McGraw-Hill.
- Smith, Eric A., Valor Whisler and Hank Marquis. 1998. *Visual Basic 6 Programming Bible*. New Jersey: John Wiley & Sons, Inc.
- Azam, M. 2001. *Programming with Visual Basic*. New Delhi: Vikas Publishing House Pvt. Ltd.
- Manchanda, Mahesh. 2009. *Visual Programming*. New Delhi: Vikas Publishing House Pvt. Ltd.
- Balena, Francesco. 1999. *Programming Microsoft Visual Basic 6.0*. Bangalore: WP Publishers and Distributors (P) Ltd.
- Petroutsos, Evangelos. 1998. *Mastering Visual Basic 6, 1st Edition*. New Delhi: BPB Publications.
- Deitel, Harvey M., Paul J. Deitel and T. Tem R. Nieto. 1999. *Visual Basic 6: How to Program*. New Jersey: Prentice-Hall.
- Norton, Peter. 1998. *Peter Norton's Complete Guide to Visual Basic 6*. New Delhi: Techmedia.
- Reselman, Bob and Richard A. Peasley. 1998. *Using Visual Basic 6*. New Jersey: Pearson Education (Que Publishing).
- Donald, Bob and Oancea Gabriel. 1999. *Visual Basic 6 from Scratch*. New Delhi: Prentice-Hall of India.

UNIT 11 USING MDI FORMS

- 11.0 Introduction
- 11.1 Objectives
- 11.2 Multiple Document Interface (MDI) Forms
 - 11.2.1 Accessing Child Forms
 - 11.2.2 Adding, Loading and Unloading Forms
- 11.3 Answers to Check Your Progress Questions
- 11.4 Summery
- 11.5 Key Words
- 11.6 Self-Assessment Questions and Exercises
- 11.7 Further Readings

NOTES

11.0 INTRODUCTION

Multiple Document Interface (MDI) is used to open various windows at the same time. Parent Window contains other document windows and provides Graphical User Interface (GUI) workspace in the application. Visual Basic applications contain only one MDI form which contains other child forms. To work with child form you need to set the True status in child property. At run time, child forms are displayed within its editing area of an MDI form. The MDI is designed for creating document cantered applications. This application provides a user many similar documents at the same time.

The Multiple Document Interface (MDI) was designed to simplify the exchange of information among documents, all under the same roof.

Windows applications that can open multiple documents at the same time and allow the user to switch among them with a mouse-click. Multiple Word is a typical example, although most people use it in single document mode. Each document is displayed in its own window, and all document windows have the same behaviour. The main Form, or MDI Form, is not 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 has certain properties. There can be many child forms contained within the parent Form, but there can be only one parent Form.

In this unit, you will study about the MDI forms, basic building blocks, creation of MDI form and Child forms.

11.1 OBJECTIVES

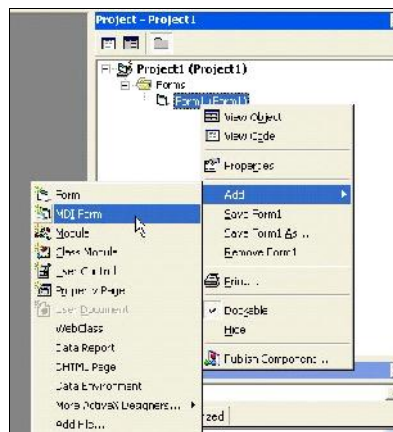
After going through this unit, you will be able to:

NOTES

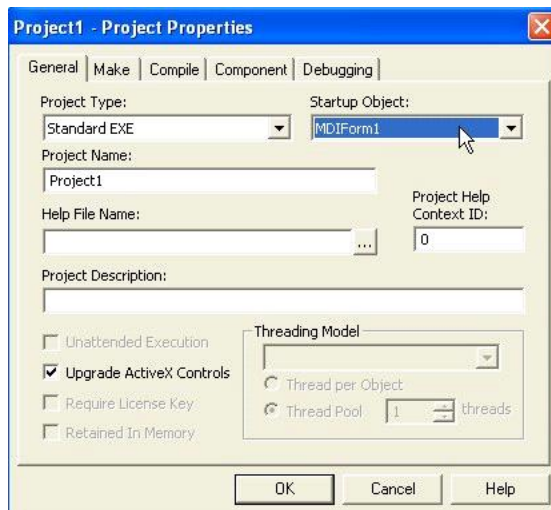
- Understand the significance of Multiple Document Interface (MDI)
- Explain about the basic MDI applications
- Discuss about the built-in capabilities of MDI
- Elaborate on the concept of Child form
- Know about the adding, loading and unloading forms in Visual Basic

11.2 MULTIPLE DOCUMENT INTERFACE (MDI) FORMS

In Visual Basic (VB), a document centered application is created with the help of two forms. These two forms refer to Multiple Document Interface (MDI) Form and a child form. The designed MDI application provides all features which are available in Notepad application in Microsoft Windows. If a user clicks on **File** → **New** menu each time a new child Window is created and displayed on the screen. The Form1 is a single form. In Visual Basic, you can use MDI Form, a form that can contain multiple forms. To create MDI Form in Visual Basic you need to select Form1 from Project1 menu list. Then you can get a shortcut menu by pressing the right mouse button as shown in the following screen.

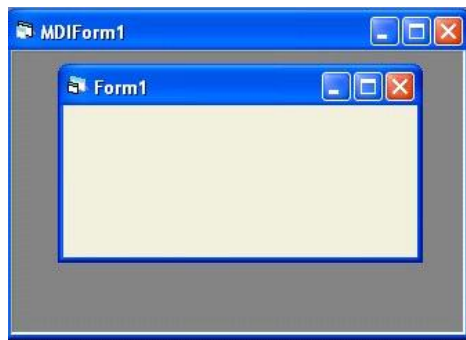


Start Visual Basic Standard Exe project. In the Project Window, select **Add** → **MDI Form** from a pop-up menu. In the dialog Window, select Open button and then go to Form1. Change the MDI Child Property of Form1 into True. This will make Form1 as child form instead of parent. In the Visual Basic menu, select **Project** → **Project1 Properties**. In the General tab, Startup Object drop-down list select MDIForm1 and then OK button. This is to make Visual Basic run for the first time by calling the MDI Form.



NOTES

After clicking on OK button you will get the MDIForm1 tab that provides you Form1 Windows.



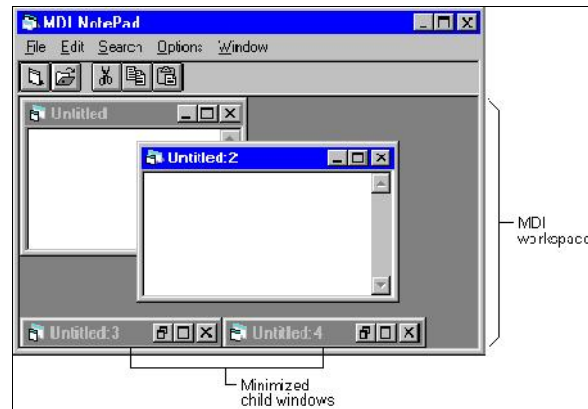
MDI is designed to exchange information between documents. With the main application you can maintain multiple open windows but not multiple copies of the application. Data exchange is easier when you can view and compare many documents simultaneously.

MDI Applications

MDI applications use multiple child forms within parent window. These applications let a user to work on various sets of data. Dragging information between child windows use a Windows menu through which you can extract information between various windows. MDI allows you to create an application that maintains multiple forms within a single container form. Applications, such as Microsoft Excel and Microsoft Word for Windows have multiple document interfaces. An MDI application allows the user to display multiple documents at the same time with each document displayed in its own window. Documents or child windows are contained in a parent window which provides a workspace for all the child windows in the application. For example, Microsoft Excel allows you to create and display multiple document windows of different types. Each individual window is confined to the area of the Excel parent window. When you minimize the Excel application

NOTES

the document windows is also minimized as well and only the parent window's icon appears in the task bar. When a child form is minimized its icon appears within the workspace of the MDI form instead of on the taskbar, as shown in the following screen in which child forms are displayed within the workspace of the MDI form.



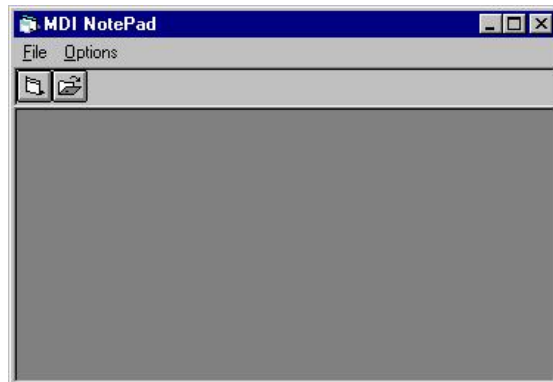
The application can also include standard, non-MDI forms that are not contained in the MDI form. A typical use of a standard form in an MDI application is to display a modal dialog box. An MDI form is similar to an ordinary form except that you cannot place a control directly on a MDI form unless that control has an `Align` property, such as a picture box control or has no visible interface, such as a timer control. The following steps are required to create an MDI form and its child forms:

- Create an MDI form. From the Project menu, choose Add MDI Form. An application can have only one MDI form. If a project already has an MDI form the 'Add MDI Form' command on the Project menu is unavailable.
- Create the application's child forms.

In Visual Basic 6.0, MDI applications are created by adding an MDI form to a project and then setting the `MDIChild` property of any child form. In Visual Basic 6.0, an MDI application having a form that is not an MDI child does not end until that form is closed even if the MDI parent is closed. The application ends when the startup form is closed, regardless of any non-MDI forms in the application. In an MDI application, the menus for each child are displayed on the MDI form rather than on the child forms themselves. If there are no child forms visible or if the child with the focus does not have a menu the MDI form's menu is displayed. It is common for MDI applications to use several sets of menus. When the user opens a document the application displays the menu associated with that type of document. Usually, a different menu is displayed when no child forms are visible. For example, when there are no files open Microsoft Excel displays only the File and Help menus. When the user opens a file other menus are displayed, such as File, Edit, View, Insert, Format, Tools, Data and Window.

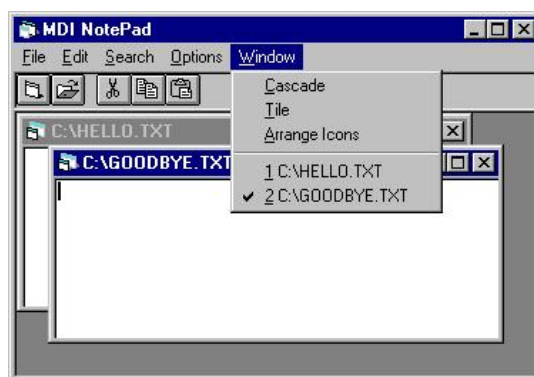
Creating Menus for MDI Applications

You can create menus for your Visual Basic application by adding menu controls to the MDI form and to the child forms. One way to manage the menus in your MDI application is to place the menu controls you want displayed all of the time, even when no child forms are visible, on the MDI form. When you run the application, the MDI form's menu is automatically displayed when there are no child forms visible, as shown in the following screen and the MDI form menu is displayed when no child forms are loaded.



Place the menu controls that apply to a child form on the child form. At run time, as long as there is at least one child form visible these menu titles are displayed in the menu bar of the MDI form. Some applications support more than one type of documents. For example, in Microsoft Access, you can open tables, queries, forms and other document types. To create an application in Visual Basic you need to use two child forms. Design one child with menus that perform spreadsheet tasks and the other with menus that perform charting tasks.

Creating a Window Menu: Most MDI applications, for example, Microsoft Word for Windows and Microsoft Excel utilizes Window menu. This is a special menu that displays the captions of all open child forms, as shown in the following screen and some applications place commands on this menu that manipulates the child Windows by various arrangements, such as Cascade, Tile and Arrange Icons.



NOTES

NOTES

The Window menu displays the name of each open child form and any menu control on an MDI form or MDI child form can be used to display the list of open child forms by setting the WindowList property for that menu control to True. At run time, Visual Basic automatically manages and displays the list of captions and displays a check mark next to the one that had the focus most recently. In addition, a separator bar is automatically placed above the list of Windows.

To Set the WindowList Property: The following steps are required to set the WindowList property:

- Select the form where you want the menu to appear and from the Tools menu, choose Menu Editor. The WindowList property applies only to MDI forms and MDI child forms.
- In the Menu Editor List box select the menu where you want the list of open child forms to display.
- Select the WindowList check box.

At run time, this menu displays the list of open child forms. In addition, the WindowList property for this menu control returns as True.

Built-in Capabilities of MDI, Parent and Child Menus

MDI is a popular interface because it allows you to have multiple documents (or forms) open in one application and hence considered as built-in interface. Each application consists of one (or more) parent Windows, each containing an MDI client area. It is the area where the child forms (or documents) will be displayed. Code you write displays as many instances of each of the child forms that you want displayed and each child form can only be displayed within the confines of the parent Window, i.e., you cannot drag the child forms outside the MDI container. Following screen shows a basic MDI application in use. For example, Main Form tab as shown in the following screen contains Product Information 1 tab which has four fields, such as Product ID, Product Name, Unit Price and Units In Stock. The user can enter valid data using the text bars which are assigned for the fields.



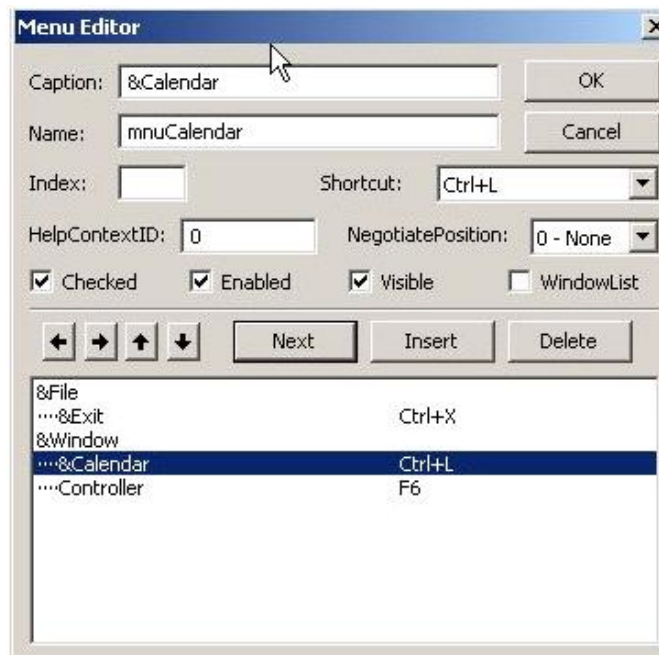
You need to use MDI to open multiple Windows and all child Windows are contained within the parent area. To create an MDI parent form, you can simply take one of your existing forms and set its `IsMDIContainer` property to `True`. This form will now be able to contain other forms as child forms. You may have one or many container forms within your application. You may have as many different child forms, the forms that remain contained within the parent form as you want in your project. A child form is nothing more than a regular form for which you dynamically set the `MdiParent` property to refer to the MDI container form. To create child menu in Visual Basic 6, the following steps are to be performed:

- Continue the MDI Form project, in the Project Window, double click the MDI Form1 to put the MDI Form in front
- In the Menu of Visual Basic, click **Tools** → **Menu Editor** or you can use CTRL+E. The menu Editor will appear.
- In the Caption, type `&File` and in the Name: bar type `mnuFile` then click Next button.
- In the Caption: bar, type `&Exit` and in the Name: bar type `mnuExit` then click right arrow button. In the shortcut list, choose CTRL+X and then select Next button.
- In the Caption: bar, type `&Window` and in the Name: bar type `mnuWindow` then click the Window List CheckBox and then the **Next** button.
- In the Caption: bar, type `&Calendar` and in the Name: bar type `mnuCalendar` then click right arrow button. In the Shortcut list, choose CTRL+L then select **Next** button.
- In the Caption, type `&Controller` and in the Name: bar type `mnuController` then click right arrow button. In the shortcut list, choose F6 and then select **Next** button.
- The menu editor will be the same as shown in screen below. After setting all the above setting select **OK** button.

Now you have menu in the MDI Form where you need to select File → **Exit** and select **ENTER** button. You will get the Menu Editor tab in which all the above steps are to be performed.

NOTES

NOTES



In the `mnuExit_Click` procedure type the following code:

```
Private Sub mnuExit_Click()
End
End Sub
```

In the menu of MDI form, select **Window ð Controller** and click to write the following code:

```
Private Sub mnuController_Click()
If mnuController.Checked = True Then

'now is checked, make it unchecked

frmController.Hide
mnuController.Checked = False
Else
' now is unchecked, make it checked
frmController.Show
mnuController.Checked = True
End If
End Sub
```

Run the program and look at the Window menu and then select the Exit menu. Windows applications provide groups of related commands in Menus. These commands depend on the application but some menus, such as Open and Save are frequently found in the applications. Menus are considered as intrinsic

controls. On the other hand, menus behave differently from other controls. You can design them in the Menu Editor Window. You invoke this tool from the Menu Editor Button on the standard toolbar. Visual Basic provides an easy way to create menus with the modal Menu Editor dialog. The below dialog is displayed when the Menu Editor is selected in the Tool Menu. The Menu Editor command is grayed unless the form is visible. And also you can display the Menu Editor Window by right clicking on the Form and selecting Menu Editor. You enter the item's Caption and Name, set other properties and press ENTER to move to the next item. When you want to create a submenu, you press the Right Arrow button (ALT+R). When you want to return to work on top-level menus those items that appear in the menu bar you need to click the Left Arrow button (ALT+L) when the application runs. 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. You can create up to five levels of submenus in Visual Basic. You can insert a separator bar using the hyphen (-) character for the Caption property. If you forget to enter a menu item's Name, the Menu Editor complains when you decide to close it. The convention used in this book is that all menu names begin with the three letters mnu.

NOTES

11.2.1 Accessing Child Forms

A **child form** is an ordinary form that has its `MDIChild` property set to `True`. Your application can include many MDI child forms of similar or different types. At run time child forms are displayed within the workspace of the MDI parent form which refers to the area inside the form's borders and below the title and menu bars. MDI applications use multiple child forms within parent Window. These applications let a user to work on various sets of data. Project in Visual Basic can have any number of child forms contained in an MDI form. The child form becomes a child of MDI form in the Visual Basic project. The program code is responsible for loading the child forms at runtime. At design time, these children forms act as standard forms. Child forms used in MDI application have following characteristics:

- All child forms are restricted to MDI parent's form and reside in the client (internal) area.
- A minimized child form's icon always appears on user's desktop. It not only minimizes the parent Window but also all the children Window. A maximized child form's caption is declared in MDI form but it appears on MDI form's title bar.

To access the child form within MDI form you need to write the following code:

```
Me.ActiveMdiChild.TextBox1.Text
```

In the above declaration, `Me` keyword refers to MDI parent form because the code resides under it.

11.2.2 Adding, Loading and Unloading Forms

NOTES

Visual Basic projects have a special object that can be automatically loaded when the program is run. This object is referred to as the Startup object. The Startup object varies on the type of project you are creating. A form can be selected with the help of `Sub Main` procedure. When a form is specified as the Startup object the form automatically loads into memory when the application starts. No other form loads unless it is referenced in program code or is explicitly loaded into memory. Use the `Load` statement to load a form into memory. The `Load` statement will take only one argument which is the name of the object to be loaded. The declaration in Visual Basic is done in the following way:

```
Load Form1
```

```
Load frmTest
```



The `Load` statement in both cases accepts a valid object name. This causes the object to load into memory. Although the object loads into memory it does not mean that the object will be visible to the user. This enables the programmer to load multiple forms that may be required to prepare them with information before display. Once loaded into memory, the form's controls and any program code can be used. When working with forms in Visual Basic it is important to note that any reference to an object will cause that object to load. The `Load` statement does not have to be explicitly used before an object can be used. An example of this would be if a form's `Caption` property is set as follows:

```
Form1.Caption = "My Notepad"
```

There is no `Load` statement before the `Caption` property is set. This code directly sets the form's property. This single line of code automatically causes the `Form1` object to be loaded into memory. This is often referred to as implicit

loading. Implied loading can often cause problems when working on a multiform project. The programmer does not notice that one form calls or sets information on another form. The form then automatically loads. Later when you attempt to unload by name all forms that you remember using, your project continues to run. The `End` statement can be used to force the application to terminate regardless of which forms were explicitly or implicitly loaded. However, the `End` statement will have an undesirable effect because it will end the application so abruptly that the `QueryUnload` and `Unload` events of forms will not have a chance to run. When an individual form is no longer required and you can unload it from memory. This will release the graphic components from the memory. The following code unloads two forms:

```
Unload Form1
Unload frmTest
```

The `Unload` statement accepts a valid object name. This causes the design time graphic components of a form to be released. Statements, such as `Load` and `Unload` are used to control the memory status of a form. These two statements always appear before the name of the object to be affected. They are often confused with the `Show` and `Hide` methods that take place after the object name. `Show` and `Hide` are used to control whether a form is visible to the user. The control menu contains the following commands:

- **Restore:** It restores a maximized Form to the size it was before it was maximized; available only if the Form has been maximized.
- **Move:** It lets the user moves the Form around with the mouse.
- **Size:** It lets the user resize the control with the mouse.
- **Minimize:** It minimizes the Form.
- **Maximize:** It maximizes the Form.
- **Close:** It closes the Form.

Check Your Progress

1. What is the main application of Multiple Document Interface (MDI)?
2. In what ways does an MDI form differ from an ordinary form?
3. State the steps to create MDI form and its child forms.
4. Write the property for `WindowList`.
5. Give the definition of child form.
6. Write the of characteristics MDI child forms.
7. Give the example how form's caption property is set?
8. Explain the term restore.

NOTES

11.3 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

NOTES

1. MDI is designed to exchange information between documents.
2. An MDI form is similar to an ordinary form except that you cannot place a control directly on a MDI form unless that control has an `Align` property, such as a picture box control or has no visible interface, such as a timer control.
3. The following steps are required to create an MDI form and its child forms:
 - Create an MDI form. From the Project menu, choose Add MDI Form. An application can have only one MDI form. If a project already has an MDI form the 'Add MDI Form' command on the Project menu is unavailable.
 - Create the application's child forms.
4. To Set the `WindowList` Property: The following steps are required to set the `WindowList` property:
 - Select the form where you want the menu to appear and from the Tools menu, choose Menu Editor. The `WindowList` property applies only to MDI forms and MDI child forms.
 - In the Menu Editor List box select the menu where you want the list of open child forms to display.
 - Select the `WindowList` check box.
5. A child form is an ordinary form that has its `MDIChild` property set to `True`.
6. Child forms used in MDI application have following characteristics:
 - All child forms are restricted to MDI parent's form and reside in the client (internal) area.
 - A minimized child form's icon always appears on user's desktop. It not only minimizes the parent Window but also all the children Window. A maximized child form's caption is declared in MDI form but it appears on MDI form's title bar.
7. An example of this would be if a form's `Caption` property is set as follows:
`Form1.Caption = "My Notepad"`
8. **Restore:** It restores a maximized Form to the size it was before it was maximized; available only if the Form has been maximized.

11.4 SUMMERY

- In Visual Basic, a document centred application is created with the help of two forms. These two forms refer to Multiple Document Interface (MDI) Form and a child form.
- The designed MDI application provides all features which are available in Notepad application in Microsoft Windows.
- MDI is designed to exchange information between documents. With the main application you can maintain multiple open windows but not multiple copies of the application.
- MDI applications use multiple child forms within parent window. These applications let a user to work on various sets of data.
- Dragging information between child windows use a Windows menu through which you can extract information between various windows.
- MDI allows you to create an application that maintains multiple forms within a single container form. Applications, such as Microsoft Excel and Microsoft Word for Windows have multiple document interfaces.
- An MDI application allows the user to display multiple documents at the same time with each document displayed in its own window. Documents or child windows are contained in a parent window which provides a workspace for all the child windows in the application.
- The application can also include standard, non-MDI forms that are not contained in the MDI form.
- An MDI form is similar to an ordinary form except that you cannot place a control directly on a MDI form unless that control has an Align property, such as a picture box control or has no visible interface.
- Create an MDI form. From the Project menu, choose Add MDI Form. An application can have only one MDI form. If a project already has an MDI form the 'Add MDI Form' command on the Project menu is unavailable.
- You can create menus for your Visual Basic application by adding menu controls to the MDI form and to the child forms.
- One way to manage the menus in your MDI application is to place the menu controls you want displayed all of the time, even when no child forms are visible, on the MDI form.
- When you run the application, the MDI form's menu is automatically displayed when there are no child forms visible, as shown in the following screen and the MDI form menu is displayed when no child forms are loaded.
- Place the menu controls that apply to a child form on the child form.

NOTES

NOTES

- At run time, as long as there is at least one child form visible these menu titles are displayed in the menu bar of the MDI form.
- Most MDI applications, for example, Microsoft Word for Windows and Microsoft Excel utilizes Window menu.
- The Window menu displays the name of each open child form and any menu control on an MDI form or MDI child form can be used to display the list of open child forms by setting the WindowList property for that menu control to True.
- At run time, Visual Basic automatically manages and displays the list of captions and displays a check mark next to the one that had the focus most recently. In addition, a separator bar is automatically placed above the list of Windows.
- Select the form where you want the menu to appear and from the Tools menu, choose Menu Editor. The WindowList property applies only to MDI forms and MDI child forms.
- In the Menu Editor ListBox select the menu where you want the list of open child forms to display.
- MDI is a popular interface because it allows you to have multiple documents (or forms) open in one application and hence considered as built-in interface.
- Each application consists of one (or more) parent Windows, each containing an MDI client area. It is the area where the child forms (or documents) will be displayed.
- Code you write displays as many instances of each of the child forms that you want displayed and each child form can only be displayed within the confines of the parent Window, i.e., you cannot drag the child forms outside the MDI container.
- You need to use MDI to open multiple Windows and all child Windows are contained within the parent area.
- To create an MDI parent form, you can simply take one of your existing forms and set its IsMDIContainer property to True. This form will now be able to contain other forms as child forms.
- You may have one or many container forms within your application. You may have as many different child forms, the forms that remain contained within the parent form as you want in your project.
- A child form is nothing more than a regular form for which you dynamically set the MdiParent property to refer to the MDI container form.
- Windows applications provide groups of related commands in Menus. These commands depend on the application but some menus, such as Open and save are frequently found in the applications.

- Menus are considered as intrinsic controls. On the other hand, menus behave differently from other controls.
- You can design them in the Menu Editor Window. You invoke this tool from the Menu Editor Button on the standard toolbar. Visual Basic provides an easy way to create menus with the modal Menu Editor dialog.
- A child form is an ordinary form that has its MDIChild property set to True.
- Your application can include many MDI child forms of similar or different types. At run time child forms are displayed within the workspace of the MDI parent form which refers to the area inside the form's borders and below the title and menu bars.
- Which refers to the area inside the form's borders and below the title and menu bars. MDI applications use multiple child forms within parent Window. These applications let a user to work on various sets of data. Project in Visual Basic can have any number of child forms contained in an MDI form.
- The child form becomes a child of MDI form in the Visual Basic project. The program code is responsible for loading the child forms at runtime. At design time, these children forms act as standard forms.
- Visual Basic projects have a special object that can be automatically loaded when the program is run.
- This object is referred to as the Startup object. The Startup object varies on the type of project you are creating.
- A form can be selected with the help of Sub Main procedure. When a form is specified as the Startup object the form automatically loads into memory when the application starts.
- No other form loads unless it is referenced in program code or is explicitly loaded into memory.
- The Load statement in both cases accepts a valid object name. This causes the object to load into memory.
- This enables the programmer to load multiple forms that may be required to prepare them with information before display. Once loaded into memory, the form's controls and any program code can be used.
- When working with forms in Visual Basic it is important to note that any reference to an object will cause that object to load. The Load statement does not have to be explicitly used before an object can be used.
- There is no Load statement before the Caption property is set. This code directly sets the form's property.
- This single line of code automatically causes the Form1 object to be loaded into memory. This is often referred to as implicit loading. Implied loading can often cause problems when working on a multiform project.

NOTES

NOTES

- The programmer does not notice that one form calls or sets information on another form. The form then automatically loads. Later when you attempt to unload by name all forms that you remember using, your project continues to run.
- The End statement can be used to force the application to terminate regardless of which forms were explicitly or implicitly loaded.
- The Unload statement accepts a valid object name. This causes the design time graphic components of a form to be released.
- Statements, such as Load and Unload are used to control the memory status of a form.
- These two statements always appear before the name of the object to be affected. They are often confused with the Show and Hide methods that take place after the object name. Show and Hide are used to control whether a form is visible to the user.

11.5 KEY WORDS

- **Application:** A collection of objects that work together to accomplish something useful. In VB, the application is called Project.
- **Forms: form** is used in VB.NET to create a form-based or window-based application. Using the form, we can build an attractive user interface. It is like a container for holding different control that allows the user to interact with an application.
- **Multiple Document Interface (MDI):** MDI allow users to work with multiple documents by opening more than one document at a time.
- **Menu window:** Menu is a Pull-Down or Pop-Up list of commands. In the following illustration, the window has a menu bar with two menus: File and Edit. Each menu can have zero or more menu-items. For the File menu there are 6 menu items. The horizontal line is a special type of menu-item called a separator.
- **Child form:** A child form is an ordinary form that has its MDI Child property set to true.

11.6 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. Give the definition of MDI application.
2. List the steps to create an MDI form.

3. State the steps for setting the WindowList property.
4. Define about on accessing a child form.
5. Write the syntax to access the child form.
6. Explain about the load forms.

NOTES

Long-Answer Questions

1. Explain the various built-in capabilities of Multiple Document Interface (MDI) giving appropriate example.
2. Describe the method of accessing child forms with the appropriate example.
3. Briefly discuss about the MDI parent and child forms.
4. Discuss about the procedure to load and unload forms with the help of code.
5. Elaborate on the control menu commands.
6. Briefly discusses about the MDI forms along with their properties used in Visual Basic.
7. Write the VB code to resize MDI Child forms.

11.7 FURTHER READINGS

- Cornell, Gary. 1998. *Visual Basic 6 from the Ground Up*. New Delhi: Tata McGraw-Hill.
- Warner, Scott L. 1998. *Teach Yourself Visual Basic 6*. New Delhi: Tata McGraw-Hill.
- Jerke, Noel. 1999. *Visual Basic 6 - The Complete Reference*. New York: McGraw-Hill.
- Smith, Eric A., Valor Whisler and Hank Marquis. 1998. *Visual Basic 6 Programming Bible*. New Jersey: John Wiley & Sons, Inc.
- Azam, M. 2001. *Programming with Visual Basic*. New Delhi: Vikas Publishing House Pvt. Ltd.
- Manchanda, Mahesh. 2009. *Visual Programming*. New Delhi: Vikas Publishing House Pvt. Ltd.
- Balena, Francesco. 1999. *Programming Microsoft Visual Basic 6.0*. Bangalore: WP Publishers and Distributors (P) Ltd.
- Petroutsos, Evangelos. 1998. *Mastering Visual Basic 6, 1st Edition*. New Delhi: BPB Publications.
- Deitel, Harvey M., Paul J. Deitel and T. Tem R. Nieto. 1999. *Visual Basic 6: How to Program*. New Jersey: Prentice-Hall.

NOTES

Norton, Peter. 1998. Peter Norton's Complete Guide to Visual Basic 6. New Delhi: Techmedia.

Reselman, Bob and Richard A. Peasley. 1998. Using Visual Basic 6. New Jersey: Pearson Education (Que Publishing).

Donald, Bob and Oancea Gabriel. 1999. Visual Basic 6 from Scratch. New Delhi: Prentice-Hall of India.

BLOCK IV
DATA ACCESS OBJECT (DAO)
AND PROPERTIES

*Data Access Object
(DAO) and Properties*

NOTES

**UNIT 12 DATA ACCESS OBJECT
(DAO) AND PROPERTIES**

- 12.0 Introduction
- 12.1 Objectives
- 12.2 Data Access Object (DAO)
- 12.3 Accessing Data Through Transaction Method
- 12.4 Answers to Check Your Progress Questions
- 12.5 Summary
- 12.6 Key Words
- 12.7 Self-Assessment Questions and Exercises
- 12.8 Further Readings

12.0 INTRODUCTION

Data Access Objects (DAO) enable you to manipulate the structure of the DataBase and the data it contains from Visual Basic. In dataBase DAO Object correspond to objects, for example, a `TableDef` object corresponds to a Microsoft Access table. A `Field` object corresponds to a field in a table.

Most of the properties can set for DAO objects are DAO properties. These properties are defined by the Microsoft Access DataBase engine and are set the same way in any application that includes the Access DataBase engine. Some properties set for DAO objects are defined by Microsoft Access, and are not automatically recognized by the Access DataBase engine. To set a property that's defined by the Access DataBase engine, refer to the object in the DAO hierarchy. The easiest and fastest way to do this is to create object variables that represent the different objects you need to work with, and refer to the object variables in subsequent steps in your code.

This is an object model that has a collection of objects using. This model gives complete control on the DataBase. This model uses Jet Engine, which is the native DataBase engine used by Visual Basic and MS-Access. This was the first model to be used in Visual Basic. Though it is possible to access any DataBase using this, it is particularly suitable for MS-Access DataBase and not suitable for ODBC data sources such as Oracle and MicroSoft Structure Query Language (MS-SQL) Server. So, Microsoft later introduced Remote Data Object (RDO).

DAO 12.0 is the latest version, shipped with Access 2007, and is the version used by the new ACCDB file format. This new release was written for use with

NOTES

the Access DataBase engine, which is an updated version of the Microsoft Jet DataBase engine and is 100% compatible with Jet. The new features added to DAO and the Access DataBase engine include new objects and properties that support multi-value lookup fields, a new Attachment data type, append-only memo fields, and DataBase encryption using the DataBase password.

DAO has evolved right alongside Jet and the Access DataBase engine, and has become the best model for accessing and manipulating Access DataBase engine objects and structure. Because of its tight integration with Access, DAO also provides much faster access to Access DataBases than does ADO or the Jet Replication Objects (JRO).

In this unit, you will study about the Data Access Object (DAO), Move First, MoveLast, MovePrevious, Move Next method, accessing data through Microsoft Access files.

12.1 OBJECTIVES

After going through this unit, you will be able to:

- Know about the Data Access Objects (DAO)
- Understand about the RecordSet
- Access a RecordSet
- Define data access technology in Visual Basic (VB)
- Connect to data source, retrieve data from a data source
- Access ADO Data Control with the help of transaction methods

12.2 DATA ACCESS OBJECT (DAO)

When Visual Basic first started working with DataBases, it used the Microsoft Jet DataBase engine, which is what Microsoft Access uses. Using the Jet engine represented a considerable advance for Visual Basic, because you could now work with all kinds of data formats in the fields of a DataBase: text, numbers, integers, longs, singles, doubles, dates, binary values, Object Linking And Embedding (OLE) objects, currency values, Boolean values and even memo objects (up to 1.2GB of text). The Jet engine also supported Structured Query Language (SQL) which DataBase programmers found attractive.

To support the Jet DataBase engine, Microsoft added data control to Visual Basic, and you can use that control to open Jet DataBase (.mdb) files. Microsoft also added the following set of Data Access Objects (DAO) to Visual Basic:

DBEngine — The Jet DataBase engine.

Workspace — An area can hold one or more DataBases.

Database — A collection of tables.

TableDef — The definition of a table.

QueryDef — The definition of a query.

RecordSet — The set of records that makes up the result of a query.

Field — A column in a table.

Index — An ordered list of records.

Relation — Stored information about the specific relationship between tables.

For opening a DataBase in DAO, just open a Database object or form a new one. This object can represent an ISAM DataBase (for example, Paradox), a Microsoft Jet DataBase (.mdb) file or an ODBC DataBase connected through the Microsoft Jet DataBase engine. When the Database object is available, you create a RecordSet object and use that object's methods, like **MoveFirst** and **MoveNext**, to work with the DataBase.

DAO also supports a client-server connection mode referred to as *ODBCDirect*. A connection is directly established to an ODBC data source by ODBCDirect without loading the Microsoft Jet DataBase engine into memory. It is a good solution when you need ODBC features in your program.

In the ODBCDirect object model, the Connection object has information server name, and so on. It is like a Database object; in fact, a Connection object and a Database object represents different references to the same object.

Using DAO Data Control

In order to access a DataBase, you need to first draw Data Control on your form the way you draw other controls. That is, first click at Data Control on the ToolBox and then drag it on your form. After placing a data control on your form, you can set its properties, just like any other control. You can also set them from code at runtime. Some important properties of the Data control are shown in Table 12.1.

Table 12.1 Properties of Data Control

Data Control	Properties
DatabaseName	Determines the path and filename of the data base file(.MDB file for Microsoft Access).
Exclusive	If this property is set to True , then when you open the database you will have exclusive access to it. If set to False (the default), other users (or programs) can access the database while you are working on it.
Options	Determines access constraints for the tables you are working on.
Read Only	When True you cannot update the database (default is False).
RecordSource	Determines the source of the set of records (the RecordSet) retrieved by the data control. This property will be set either to the name of one of the tables in database, to the text of a valid SQL query or to the name of a predefined query in the Access database.

NOTES

NOTES

Another important property is `Connect` which indicates the source of an open DataBase. This property is not used if you are accessing an MS Access DataBase. Visual Basic incorporates the same DataBase engine (Microsoft Jet Database Engine) that powers Microsoft Access. This enables Visual Basic to access DataBase in a number of standard formats in a relatively simple manner. Database formats supported by VB are:

- Microsoft Access
- Paradox
- dBase
- Btrieve
- FoxPro

In the following examples, we shall learn to use Data control of VB for accessing an MS Access 2003.

Before we actually connect to a DataBase and retrieve its records, some important concepts and term must be talked about.

The RecordSet

RecordSet refers to the set of records (or simply a table) that are retrieved from an object as determined by **RecordSource** property.

The `RecordSet` object can be of five types. Out of these five, only the first three are available to a data control at design time by default.

i. Table Type RecordSet: It represents the base table (i.e. underlying table) in a code form. It can be used for adding, changing or deleting records from a single DataBase table.

ii. Dynaset Type RecordSet: It signifies the result of a query that can have records that are updatable. The dynaset type `RecordSet` object is a dynamic set of records that can be used for adding, changing or deleting records from an underlying DataBase table or tables. A dynaset type

`RecordSet` object can have fields from one or more tables in a DataBase.

This type correspondence to an ODBC keyset cursor.

iii. Snapshots Type RecordSet: It is a fixed copy of a set of records that can be used for finding data or generating reports. A snapshot type

`RecordSet` object can have fields from one or more tables in a DataBase but it cannot be updated. This type correspondence to an ODBC static cursor.

- iv. **Forward only Type RecordSet:** This `RecordSet` is the same to that of snapshot except that there is no cursor. You can only move forward through records.
- v. **Dynamic Type RecordSet:** It represents a result of a query set from one or more tables in which can added, changed or deleted records from a row returning query.

NOTES

The Bound Control [Supported by DAO]

A bound control (also called data aware control) is a control that can provide access to a specific column or columns in a data source through a Data Control. A data aware control can be bound to a Data Control through its **DataSource** and **DataField** properties.

The easiest way of viewing the data is by using one or more of the following bound controls:

- `CheckBox`
- `Image`
- `Label`
- `PictureBox`
- `TextBox`

Steps for Creating a Data Aware Application (using DAO Control)

1. Draw a Data control on your form.
2. Set the **DatabaseName** property to the path and filename of your DataBase.
3. Set the **RecordSource** property to the name of the DataBase table you wish to access.
4. Draw one `TextBox` on your form for each field you want to access.
5. After this, for each `TextBox`:
 - Set its **Datasource** property to the name of the data control.
 - Set its **Datafield** property to the name of the field you want that `TextBox` to access.
6. Run the program.

Accessing Data through RecordSet Object

The **RecordSet** is a table created by the data control, based on the contents of its `RecordSource` property. Depending on the `RecordSource` property, this table may be:

- One of the tables from the DataBase.
- A subset of records from such a table.

NOTES

- A subset of the fields from such a table.
- A join of two or more tables from the DataBase.
- Anything that can be returned by an SQL query.

Just like other tables in a relational DataBase, each row of the **RecordSet** is referred to as a record, and each column is referred to as *field*. The **RecordSet** can be referred to in your code as property (available at runtime only) of the data control as follows:

`Data1.RecordSet` 'data is the name of DAO data control'

However, `RecordSet` is generally referred to along with any of its properties, such as

`Data1.RecordSet.BOF` 'accessing BOF property of RecordSet'

Or for applying one of its methods, such as

`Data1.recordSet.MoveFirst`

Moving Through the Recordset

Recordset allows you to move or to navigate records. The move method of recordset is used to do so. When one uses the ADODC or data control, these commands are issued automatically when one of the navigation buttons on the ADODC or data control is clicked by the end user. With the active data object application, one has to handle functions or record navigation.

There are five move methods that could be used for record navigation. These five methods are as follows:

1. `MoveFirst` method
2. `MoveLast` method
3. `MovePrevious` method
4. `MoveNext` method
5. `Move` method

1. **The MoveFirst Method:** The `MoveFirst` method moves or positions the record pointer at the first record of the recordset. In your Form, write the following code on the command button `Cmdmovefirst`:

```
` Move to the first record
Private Sub Cmdmovefirst_Click ()
rs.MoveFirst
Call Display
End Sub
```

2. **The MoveLast Method:** The `MoveLast` method moves or positions the record pointer at the last record of the recordset. In your Form, write the following code on the command button `CmdmoveLast`:

```
` Move to the last record
```

```
Private Sub Cmdmovelast_Click ()  
rs.MoveLast  
Call Display  
End Sub
```

In the above code, `rs.MoveLast` would move the pointer to the first record and `Call Display` would associate the record to the corresponding fields.

3. **The MovePrevious Method:** The `MovePrevious` method moves the record pointer at the previous record to the current record of the recordset. In your Form, write the following code on the command button `Cmdmoveprevious`:

```
` Move to the pervious record  
Private Sub Cmdmoveprevious_Click ()  
rs.MovePrevious  
If rs.BOF = True Then  
rs.MoveFirst  
End If  
Call Display  
End Sub
```

In the above code, `rs.MovePrevious` would move the pointer to the previous record and `Call Display` would associate the record to the corresponding fields. The `rs.BOF` would check if the pointer is at the Beginning of File (BOF) it would move the pointer to the first record.

4. **The MoveNext Method:** The `MoveNext` method moves the record pointer at the next record to the current record of the recordset. In your Form, write the following code on the command button `Cmdmovenext`:

```
` Move to the next record  
Private Sub Cmdmovenext_Click ()  
rs.MoveNext  
If rs.EOF = True Then  
rs.MoveLast  
End If  
Call Display  
End Sub
```

In the above code `rs.MoveNext` would move the pointer to the next record and `Call Display` would associate the record to the corresponding fields. The `rs.EOF` would check if the pointer is at the End of File (EOF) it would move the pointer to the last record.

NOTES

5. **The Move Method:** The Move method enables you to specify a number of records forward or backward of the current position that you want to move in your recordset.

NOTES

Editing and Adding Records

The main objective to use DataBase through the VB application is to update, add or delete any record.

To edit or add a record, you need to perform the following tasks :

- Prepare the recordset to receive data changes.
- Post the new values of the fields in the recordset.
- Post the changes to the records in the DataBase.

Write the following code to add a new record:

```
` add a new record
Private Sub cmdnew_Click ()
rs!p_code = txtpid.Text
rs!p_name = txtpname.Text
rs!Qty_On_Hand = Val ( txtqoh.Text )
rs!Unit_Price = Val ( txtppu.Text )
rs.Update
End Sub
```

Write the following code to edit a record:

```
`edit record
Private Sub cmdedit_Click ()
rs!p_code = txtpid.Text
rs!p_name = txtpname.Text
rs!Qty_On_Hand = Val ( txtqoh.Text )
rs!Unit_Price = Val ( txtppu.Text )
rs.Update
End Sub
```

To delete a record, you must use the delete method of recordset.

Write the follwiong code to delete a record:

```
` delete a record
Private Sub cmddelete_Click ()
rs.Delete
End Sub
```

12.3 ACCESSING DATA THROUGH TRANSACTION METHOD

If your application has transactions that update more than one table, then it is a good idea to use the 'Transaction' method. This is to ensure that the related operations that depend on each other, either all occurred successfully or all were cancelled.

There are three types of transaction methods:

- **BeginTrans:** To be invoked when you start working on a recordset. This method begins a new transaction. Once the BeginTrans method has been invoked, the OLE DB provider will not continuously commit the changes made to a data source unless you call CommitTrans to commit the changes or RollbackTrans to reverse the changes and end the transaction.
- **CommitTrans:** To be invoked when you want to commit the changes to a data source. CommitTrans saves any changes made to a recordset and ends the current transaction.
- **RollbackTrans:** This method is to be invoked to cancel any changes made within the current transaction. This method also ends the current transaction.

The CommitTrans and the RollbackTrans may also start a new transaction.

The following code will show you how to use the transaction methods.

Add a Module to your project.

Declare the variables as follows:

```
Public adcon As New ADODB.Connection
```

```
Public rs As New ADODB.Recordset
```

In the Form_Load event add the following code:

```
Private Sub Form_Load ()  
    'Set the DSN for the Connection object  
        adcon.ConnectionString = "DSN=Invoice"  
    'Open the connection  
        adcon.Open  
    'Indicate the beginning of the transaction  
        adcon.BeginTrans  
    'Set the properties for the Recordset object  
        Set rs = New ADODB.Recordset  
        rs.LockType = adLockPessimistic
```

NOTES

NOTES

```
'Create the Recordset
rs.Open "Customer", adcon,,, adCmdTable
rs. MoveFirst
```

```
End Sub
```

In the code Module, create a function to display the current row of the Recordset.

```
Sub Showfields ()
Form1.Text1.Text = rs! Customer_Name
Form1.Text2.Text = rs! Customer_City
End Sub
```

To the Next Record Command button add the following code

```
rs.MoveNext
If rs.EOF Then rs.MoveFirst
Call Showfields
```

This code will display the next record every time you click the Next button.

Now let us assume that a user wants to edit the data that is displayed. To the Edit button add the following code:

```
rs!Customer_Name = Form1.Text1.Text
rs!Customer_City = Form1.Text2.Text
rs.Update
```

This segment of code is enough to update the recordset. In case you want the System to prompt you about the changes made, you can write another procedure called UpdateRecord. This procedure will have the following code:

```
If MsgBox("Save all changes?", vbYesNo) = vbYes
Then
adcon.CommitTrans
Else
adcon.RollbackTrans
End If
```

The above code segment will ask for your confirmation before committing the changes made by you. If you answer Yes, the changes will be committed to the data source and if you answer No, then the changes will be rolled back.

The following code example changes the job title of all sales representatives in the Employees table. After the **BeginTrans** method starts a transaction that isolates all of the changes made to the Employees table, the **CommitTrans** method

saves the changes. Be aware that you can use the **Rollback** method to undo changes that you saved with the `Update` method.

*Data Access Object
(DAO) and Properties*

```
Sub ChangeTitle()  
  
Dim wrkCurrent As DAO.Workspace  
Dim dbsNorthwind As DAO.Database  
Dim rstEmployee As DAO.Recordset  
  
On Error GoTo ErrorHandler  
  
Set wrkCurrent = DBEngine.Workspaces(0)  
Set dbsNorthwind = CurrentDB  
Set rstEmployee = dbsNorthwind.OpenRecordset  
    ("Employees")  
  
wrkCurrent.BeginTrans  
Do Until rstEmployee.EOF  
    If rstEmployee!Title = "Sales Representative"  
Then  
        rstEmployee.Edit  
        rstEmployee!Title = "Sales Associate"  
        rstEmployee.Update  
    End If  
    rstEmployee.MoveNext  
Loop  
  
If MsgBox("Save all changes?", vbQuestion +  
vbYesNo) = vbYes Then  
    wrkCurrent.CommitTrans  
Else  
    wrkCurrent.Rollback  
End If  
  
rstEmployee.Close  
dbsNorthwind.Close  
wrkCurrent.Close
```

NOTES

NOTES

```
Set rstEmployee = nothing
Set dbsNorthwind = Nothing
Set wrkCurrent = Nothing
```

```
Exit Sub
```

```
ErrorHandler:
```

```
MsgBox "Error #: " & Err.Number & vbCrLf &
vbCrLf & Err.Description
```

```
End Sub
```

When you use transactions, all DataBases and Recordset objects in the specified **Workspace** object are affected; transactions are global to the workspace, not to a specific DataBase or **Recordset**. If you perform operations on more than one DataBase or within a workspace transaction, the **Commit** and **Rollback** methods affect all the objects changed within that workspace during the transaction.

Check Your Progress

1. How many data types are used in the fields of DataBase.
2. Define the term RecordSet.
3. Explain the term of bound control.
4. State about the MoveFirst method.
5. Explain the steps of editing and adding records.
6. What are the types of transaction methods.

12.4 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Data formats used fields of a DataBase: text, numbers, integers, longs, singles, doubles, dates, binary values, OLE objects, currency values, Boolean values and even memo objects (up to 1.2GB of text).
2. RecordSet refers to the set of records (or simply a table) that are retrieved from an object as determined by RecordSource property.
3. A bound control (also called data aware control) is a control that can provide access to a specific column or columns in a data source through a Data Control.

4. The **MoveFirst Method**: The **MoveFirst** method moves or positions the record pointer at the first record of the recordset. In your Form, write the following code on the command button **Cmdmovefirst**:

```
` Move to the first record  
Private Sub Cmdmovefirst_Click ()  
rs.MoveFirst  
Call Display  
End Sub
```

5. To edit or to add a record, you need to perform the following steps :
- Prepare the recordset to receive data changes.
 - Post the new values of the fields in the recordset.
 - Post the changes to the records in the DataBase.
6. There are three types of transaction methods:
- i. **BeginTrans**
 - ii. **CommitTrans**
 - iii. **RollbackTrans**

NOTES

12.5 SUMMARY

- When Visual Basic first started working with DataBases, it used the Microsoft Jet DataBase engine, which is what Microsoft Access uses.
- Using the Jet engine represented a considerable advance for Visual Basic, because you could now work with all kinds of data formats in the fields of a DataBase: text, numbers, integers, longs, singles, doubles, dates, binary values, OLE objects, currency values, Boolean values and even memo objects.
- The Jet engine also supported SQL which DataBase programmers found attractive.
- DAO also supports a client-server connection mode referred to as *ODBCDirect*.
- A connection is directly established to an ODBC data source by *ODBCDirect* without loading the Microsoft Jet DataBase engine into memory. It is a good solution when you need ODBC features in your program.
- In order to access a DataBase, you need to first draw Data Control on your form the way you draw other controls.
- That is, first click at Data Control on the ToolBox and then drag it on your form.

NOTES

- Visual Basic incorporates the same DataBase engine (Microsoft Jet Database Engine) that powers Microsoft Access.
- Table Type `RecordSet` represents the base table (i.e. underlying table) in a code form. It can be used for adding, changing or deleting records from a single DataBase table.
- Dynaset Type `RecordSet` signifies the result of a query that can have records that are updatable. The dynaset type `RecordSet` object is a dynamic set of records that can be used for adding, changing or deleting records from an underlying DataBase table or tables.
- Snapshots Type `RecordSet` is a fixed copy of a set of records that can be used for finding data or generating reports. A snapshot type `RecordSet` object can have fields from one or more tables in a DataBase but it cannot be updated. This type corresponds to an ODBC static cursor.
- This `RecordSet` is the same to that of snapshot except that there is no cursor. You can only move forward through records.
- Dynamic Type `RecordSet` represents a result of a query set from one or more tables in which can added, changed or deleted records from a row returning query.
- A bound control (also called data aware control) is a control that can provide access to a specific column or columns in a data source through a Data Control.
- The data control maintains a pointer to one record from the `RecordSet`.
- `Recordset` allows you to move or to navigate records. The `move` method of `recordset` is used to do so.
- When one uses the ADODC or data control, these commands are issued automatically when one of the navigation buttons on the ADODC or data control is clicked by the end user. With the active data object application, one has to handle functions or record navigation.
- The `MoveFirst` method moves or positions the record pointer at the first record of the `recordset`.
- The `MoveLast` method moves or positions the record pointer at the last record of the `recordset`.
- The `MovePrevious` method moves the record pointer at the previous record to the current record of the `recordset`.
- The `MoveNext` method moves the record pointer at the next record to the current record of the `recordset`.
- The `Move` method enables you to specify a number of records forward or backward of the current position that you want to move in your `recordset`.
- The main objective to use DataBase through the VB application is to update, add or delete any record.

- If your application has transactions that update more than one table, then it is a good idea to use the 'transaction' method.
- This is to ensure that the related operations that depend on each other, either all occurred successfully or all were cancelled.
- BeginTrans to be invoked when you start working on a recordset.
- This method begins a new transaction. Once the BeginTrans method has been invoked, the OLE DB provider will not continuously commit the changes made to a data source unless you call CommitTrans to commit the changes or RollbackTrans to reverse the changes and end the transaction.
- CommitTrans to be invoked when you want to commit the changes to a data source. CommitTrans saves any changes made to a recordset and ends the current transaction.
- RollbackTrans method is to be invoked to cancel any changes made within the current transaction. This method also ends the current transaction.
- The CommitTrans and the RollbackTrans may also start a new transaction.
- This segment of code is enough to update the recordset. In case you want the System to prompt you about the changes made, you can write another procedure called UpdateRecord.
- The above code segment will ask for your confirmation before committing the changes made by you. If you answer Yes, the changes will be committed to the data source and if you answer No, then the changes will be rolled back.
- After the BeginTrans method starts a transaction that isolates all of the changes made to the Employees table, the CommitTrans method saves the changes.
- Be aware that you can use the Rollback method to undo changes that you saved with the Update method.
- When you use transactions, all DataBases and Recordset objects in the specified Workspace object are affected; transactions are global to the workspace, not to a specific DataBase or Recordset.
- If you perform operations on more than one DataBase or within a workspace transaction, the Commit and Rollback methods affect all the objects changed within that workspace during the transaction.

NOTES

12.6 KEY WORDS

- **Record:** A logical section of a file to hold a related set of data.
- **Field:** A column in a table.
- **Index:** An ordered list of records.

NOTES

- **Relation:** Stored information about the specific relationship between tables.
- **I/O:** Stands for Input/Output. Whenever you work with a file you should have ways of reading data from the file (**Input**) and ways of writing data to the file (**Output**). I/O operations include all the commands that make reading and writing files possible.
- **Database management system:** A software system that allows users to not only define and create a DataBase but also maintain it and control its access.
- **Transaction:** An action used to perform some manipulation on data stored in the DataBase.
- **Database schema:** The overall description of a DataBase which is specified during DataBase design and is not expected to be changed frequently.
- **Data dictionary:** It stores the information about the organization and usage of data contained in the DataBase.
- **Data control:** A connection between information and the bound controls that is used for displaying the information.
- **Connect:** An important property of DAO control which indicates the source of an open DataBase.

12.7 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What are Data Access Objects (DAO)?
2. Which data formats you can use in the DataBase?
3. Define the term RecordSet.
4. Explain the term bound control.
5. Differentiate between MoveFirst and MoveLast method.
6. State about the BeginTrans.

Long-Answer Questions

1. Discuss about the types of Data Access Object (DAO) that are added by Microsoft to Visual Basic.
2. Explain about the of RecordSet types that are available for data control at design time giving appropriate examples?
3. Briefly discuss the five move methods that could be used for record navigation, Explain with the help of examples.

4. Discuss briefly the methods of moving recordset giving appropriate example.
5. Describe about the editing and adding records method giving examples.
6. Analyse the transaction methods and its types briefly.
7. Design an application that displays the records of Department Table (DEPT) one-by-one. For each department record, the lower part of the form, i.e., the subform, should display all the records in the EMP table that have the same Deptno as shown below.

NOTES

empno	ename	job	deptno	salary	manager
101	harsh	clerk	1	35000	anal
102	aman	officer	1	20000	anal

12.8 FURTHER READINGS

- Cornell, Gary. 1998. *Visual Basic 6 from the Ground Up*. New Delhi: Tata McGraw-Hill.
- Warner, Scott L. 1998. *Teach Yourself Visual Basic 6*. New Delhi: Tata McGraw-Hill.
- Jerke, Noel. 1999. *Visual Basic 6 - The Complete Reference*. New York: McGraw-Hill.
- Smith, Eric A., Valor Whisler and Hank Marquis. 1998. *Visual Basic 6 Programming Bible*. New Jersey: John Wiley & Sons, Inc.
- Azam, M. 2001. *Programming with Visual Basic*. New Delhi: Vikas Publishing House Pvt. Ltd.
- Manchanda, Mahesh. 2009. *Visual Programming*. New Delhi: Vikas Publishing House Pvt. Ltd.
- Balena, Francesco. 1999. *Programming Microsoft Visual Basic 6.0*. Bangalore: WP Publishers and Distributors (P) Ltd.

NOTES

Petroutsos, Evangelos. 1998. *Mastering Visual Basic 6*, 1st Edition. New Delhi: BPB Publications.

Deitel, Harvey M., Paul J. Deitel and T. Tem R. Nieto. 1999. *Visual Basic 6: How to Program*. New Jersey: Prentice-Hall.

Norton, Peter. 1998. *Peter Norton's Complete Guide to Visual Basic 6*. New Delhi: Techmedia.

Reselman, Bob and Richard A. Peasley. 1998. *Using Visual Basic 6*. New Jersey: Pearson Education (Que Publishing).

Donald, Bob and Oancea Gabriel. 1999. *Visual Basic 6 from Scratch*. New Delhi: Prentice-Hall of India.

UNIT 13 ACTIVE DATA OBJECTS (ADO) AND ADO PRIMER

NOTES

- 13.0 Introduction
- 13.1 Objectives
- 13.2 Active Data Objects (ADO): An Introduction
- 13.3 Answers to Check Your Progress Questions
- 13.4 Summary
- 13.5 Key Words
- 13.6 Self-Assessment Questions and Exercises
- 13.7 Further Readings

13.0 INTRODUCTION

In VB, you can access databases through three different data access mechanisms or interfaces: Data Access Objects (DAO), Remote Data Objects (RDO) and Active X Data Objects (ADO). DAO, which was developed before RDO and ADO, refers to a set of objects that enables client applications to programmatically access data. ADO is developed as an easy-to-use application level interface to Microsoft's latest and most dominant data access paradigm, OLE DB. The function of Object Linking and Embedding DataBase (OLE DB) is to provide a high-performance access to any data source, including relational and non-relational databases, e-mails, text and graphics, custom business objects, and so on. The concept behind VB ADO was Universal Data Access (UDA), in which one database access method could be used for any data source. ADO has been developed to replace both DAO and RDO.

The ActiveX Data Objects (ADO) data control is the principal interface between a Visual Basic (VB) application and a database. It can be used without writing any code at all or it can be the main part of a complicated database management system. This icon may not appear in your VB ToolBox. If it does not, select Project from the main menu, then click Components. The Components dialog box window will appear. Select Microsoft ADO Data Control and then click OK. The control will be added to your toolbox.

In this unit you will study about the ActiveX Data Object (ADO), OLE DB, and ADO Object Model.

13.1 OBJECTIVES

After going through this unit, you will be able to:

- Access data in VB using ADO
- Describe the technology used for accessing data in VB

NOTES

- List the advantages of OLE DB and ADO
- Understand Activex Data Object Model
- Understand the Advantages of ADO model
- Analysis the significance of ADO data control
- Describe the mechanism of locking data to prevent access to it

13.2 ACTIVE DATA OBJECTS (ADO): AN INTRODUCTION

ActiveX Data Objects or ADO

ActiveX Data Objects (ADO) is the new data access technology offered by Microsoft. ADO is meant to replace Data Access Objects (DAO), Remote Data Objects (RDO) and Open Database Connectivity (ODBC). This means that all new programmers must embrace this new offering from Microsoft. The older technologies have been moved to the 'Maintenance' mode which is a nice way of saying that these technologies will not see any further development, nor will any of the reported bugs be fixed.

Why ADO?

Why do we need another data access engine when we already have DAO and RDO? To answer this question, let us take a look at the DAO's object model vis-a-vis the current data access requirements. The DAO with the help of ODBC can let you connect to Jet, Index Sequential Access Method (ISAM) databases and other Relational databases.

However, today's data access requirements are not limited to handling only relational data. We need to access data from other sources as well, such as mail, the Internet content, directory data from other machines etc. The technology required to access information from these different data sources is different. Therefore, the data access model will have to change to accommodate the new requirements.

We have a choice. We can access any type of data and store it locally in a new type of database and tackle the various types of data using its native method or we can implement the various technologies present in our data access table. Both these alternatives have their own problems. Ultimately, you are going to spend a lot more time learning various data access methods than developing applications.

Moreover, our requirements do not end with merely getting the data from the source and downloading it on our machine. We would also like to make changes to the data and update the data source with these changes.

What we need is a simple, consistent Application Programming Interface (API) that enables applications to gain access and modify a wide variety of data sources. A data source may be a database, a text file, a spreadsheet, a graphics application, a cluster of heterogeneous databases or something yet to be invented.

Advantages of Object Linking and Embedding DataBase (OLE DB) and ActiveX Data Object (ADO)

The following are the advantages of OLE DB and ADO

OLE DB

OLE DB is a set of Component Object Model (COM) interfaces that provide uniform access to the data stored in diverse information sources. OLE DB is defined as a new low level interface that is a part of the Universal Data Access (UDA) platform. It is defined as a general purpose set of interfaces designed to let developers build data access tools as components using the COM. OLE DB enables applications to have uniform access to the data stored in DBMS and non DBMS information containers while continuing to take advantage of the benefits of database technology without having transferred data from its place of origin to a DBMS.

This means OLE DB is not restricted to ISAM, Jet or even relational data sources, but is capable of dealing with any type of data, regardless of its format or storage method. In practice, this versatility means you can access data that resides in an Excel spreadsheet, text files or even on a mail server, such as Microsoft Exchange.

OLE DB has what it called 'Providers' which let you access the different data sources. For different data sources you have different data providers. OLE DB provides four services that you will be using in your application. They are:

1. OLE DB provides cursor service which is defined as a temporary, read only table that saves the results of a query with an assigned name. The cursor is available for browsing, reporting or other uses until it is closed.
2. A service to perform batch updates.
3. A shape service to build the data in the form of a hierarchy.
4. A remote data service provider for managing data in multi-tier environments over connected or disconnected networks.

Unfortunately, Visual Basic cannot access the OLE DB directly because of its sophistication. This is where ADO comes into the picture. The ADO acts like the intermediary between the application and the OLE DB. Now that you understand why we need ADO, let us see what ADO is all about.

NOTES

NOTES

ADO

ADO enables your client applications to access and manipulate data in a database server through any of the OLE DB providers. According to Microsoft, “ADO’s primary benefits are ease of use, high speed, low memory overheads and a small disk footprint. ADO supports key features for building client-server and Web based applications.”

The ADO features an object model like the DAO and the RDO but it is much flatter. In the case of the DAO, you had seventeen objects and in the case of the ADO, you have only seven objects. Besides, you do not have to follow a strict hierarchy when working with these objects in ADO. For example, unlike in the case of ADO, in the DAO you cannot create a recordset without using the database object.

Let us see how the ADO allows you to access data from a database. Regardless of the data access method, working on the data from a database involves the following steps:

1. Establishment of a connection to a data source.
2. Extraction of the required data with a suitable command.
3. Having extracted the data and worked on it, we may have to keep the data source up-to-date.
4. Keep an eye on the errors that may occur and take suitable action.

The ADO programming model illustrated in the following heading allows you to do all these and more.

The advantages of DAO, RDO and ADO are given below:

DAO	RDO	ADO
Data Access Objects (DAO) increases the cohesiveness of the objects by extracting out the data access code from them.	Remote Data Objects (RDO) is an object-oriented data access interface to ODBC combined with the easy-to-use style of Data Access Objects (DAO).	One key advantage of ADO is its universality. ADO can be used with both relational and non relational databases, as well as file systems, text, and other sources.
DAO reduces network overhead by caching queries results.	RDO also handles all types of result sets including those generated by multiple result set procedures, those returning output arguments and those requiring complex input parameters.	The primary benefits of ADO are ease of use, high speed, low memory overhead and a small disk footprint. ADO supports key features for building client/server and Web based applications.
DAO enhances loose coupling between the business tier and the data sources.	RDO can execute ordinary table-based queries, but it is especially adept at building and executing queries against stored procedures.	ADO is easy to use and it is language-independent.

The disadvantages of DAO, RDO and ADO are given below:

DAO	RDO	ADO
DAO requires large volumes of implementation code to be written.	RDO does not support Jet or Indexed Sequential Access Method (ISAM) databases very well.	ADO is also the most recent addition to the data access options offered by Microsoft. Its object model is more compact than those of DAO and RDO.
The DAO control the data access in a much stricter way than with the Data control.	RDO includes complex cursors including batch.	When ADO first came out, many developers were complaining about bugs and speed. However, ADO is improving, and with ADO.NET, developers will get even more flexibility, scalability, and options.

NOTES

The ActiveX Data Object Model

The goal of ADO is to gain access, to edit and up-to-date data sources.

Figure 13.1 shows the ADO programming model.

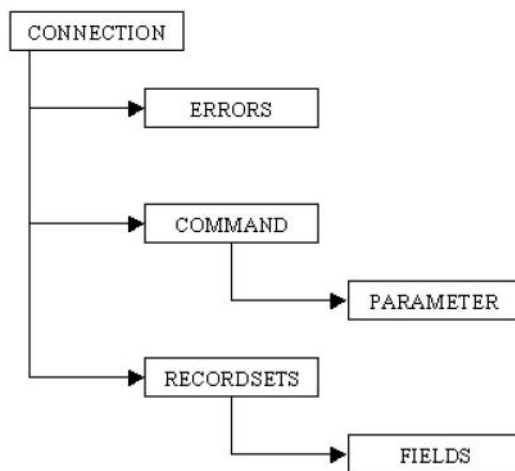


Fig. 13.1 Displaying ADO Programming Model

The ADO programming model provides classes and objects to perform each of the following activities:

- **Connection:** Makes a connection to a data source.
- **Command:** Creates an object to represent an SQL command.
- **Parameter:** Specifies columns, tables and values in an SQL command as variable parameters.

NOTES

- **Recordset:** Executes a command and store the result, if the command is returning row, in a cache. `Recordset` also allows a user to sort, view or edit the data and up-to-date the data source.

Apart from these objects, there are other objects that we will discuss shortly.

- **Connection:** You can access a data source using a `Connection` object. A `Connection` represents an open session or `Connection` to a data source. Unless a connection is made, data cannot be exchanged between the data source and the application. A `Connection` object specifies the name of a data source, the provider that will be used to access the data and other parameters.

Your application can gain access to a data source directly (sometimes called a two-tier system) or indirectly (sometimes called a three-tier system) through an intermediary like the Internet Information Server.

After gaining access to a data source, ADO ensures that the updates made to the data source are made such that there is data consistency and integrity. A transaction delimits the beginning and the end of a series of data access operations that transpire across a connection. If you cancel the transaction or one of its operations fails, ADO ensures that the changes made are 'rolled back' so that the ultimate result will be as if none of the operations in the transaction had occurred.

- **Command:** Once a connection has been established with the data source, the data has to be extracted. This is done by using a `Command` object. A `Command` adds, deletes and updates data in the data source or retrieves data in the form of rows in a table.
- **Parameter:** A command to retrieve data can be qualified by using parameters. Parameters are arguments to a command that alter the result of the execution of the command. For example, you could issue the same data retrieval command repeatedly, varying your specification of the information to be retrieved each time.

Parameters are especially useful for executing commands that behave like functions. In this case, you know what the command does but not necessarily how it works. For example, you issue a bank transfer command that debits one account and credit another. You specify the amount of money to be transferred as a parameter.

- **Recordset:** Although ADO allows you to access any type of data, here our discussion is limited to the data from a database. The `Command` object when executed will return a set of rows from one or more tables. This set of rows is called `Recordset`. This is not different from the definition of the `Recordset` in the DAO object model. A `Recordset` is placed in local storage.

However, there is no object that represents a single row of a `Recordset`.

A `Recordset` is the primary means of examining and modifying data in the rows. A `Recordset` object allows you to:

- o Specify rows that are available for examination.
- o Traverse the rows.
- o Specify the order in which the rows may be traversed.
- o Add, change or delete rows.
- o Up-to-date the data source with changed rows.
- o Manage the overall state of the recordset.
- **Field:** A row of a `Recordset` consists of one or more fields. If you envision a `Recordset` as a two-dimensional grid, the fields line up to form columns. Each field (column) has a name, a data type and a value. It is the value that contains the actual data of a data source. To change the data of a data source, you have to modify the value of the field object. In order to make sure that every occurrence of a particular field in all the tables is modified, you can use the transaction method of the `Connection` object.
- **Error:** Errors can occur at any time in your application due to the data source being corrupted or renamed by somebody or the Password being changed or for many other reasons that a programmer can understand. This results in not being able to establish a connection, execute a command or perform an operation on an object in a suitable state (for example, attempting to use a `Recordset` object that has not been initialised). This error object behaves like the error object in DAO.
- **Property:** Each ADO object has a set of unique properties that either describes or controls the behaviour of that object. There are two types of properties: built-in and dynamic. Built-in properties are part of the ADO object and are always available. Dynamic properties are added to the ADO object's properties collection by the underlying data provider and exist only when that provider is being used.
- **Collection:** Just as in DAO, ADO provides collections, a type of object that contains other objects of a particular type. The objects in the collection can be retrieved with a collection method, either by name as a text string or by ordinal as an integer number. ADO provides four types of collections:
 - o A `Connection` object has the errors collection which contains all `Error` objects created in response to a single failure involving the data source.

NOTES

NOTES

- o A `Command` object has the `Parameters` collection which contains all `Parameter` objects that can be applied to that `Command` object.
- o A `RecordSet` object has the `Fields` collection which contains all `Field` objects that define the columns of that `Recordset` object.
- o In addition, the `Connection`, `Command`, `RecordSet` and `Field` objects all have a `Properties` collection, which contains all the `Property` objects that can be applied to their respective containing objects.
- **Events:** This is new in ADO. ADO 2.0 introduces the concept of events to a programming model. Events are notifications that certain operations are about to occur or have already occurred. You can use events in general to efficiently orchestrate an application consisting of several asynchronous tasks.

If you know that an event is about to occur, for example, a commit or a delete, you have the opportunity to examine the parameters and take suitable action. This is just like Windows asking your permission to delete the files from the Recycle Bin.

The events which inform you about the completion of a particular operation, allow the application to proceed with the next step. Event handlers are called after an operation completes to notify you about the completion of an asynchronous operation. ADO 2.0 introduces several operations that have been enhanced to optionally execute asynchronously. For example, an application that starts an asynchronous `Recordset.Open` operation is notified by an execution complete event when the operation concludes.

There are two families of events:

- **ConnectionEvents:** Events are issued when transactions on a connection begin, committed or rolled back, when commands execute and connections start or end.
- **RecordsetEvents:** Events are issued to report the progress of data retrieval in the following cases: when you navigate through the rows of a `Recordset` object, when you change a field in a row of a `Recordset`, when you change a row in a `Recordset` or make any change in the entire `Recordset`.

Accessing ADO Data Control

We can display the data from a `Recordset` (data source) using ADO code or with the help of the ADO data control. In order to use the ADO Data, we need to add the control to the form. The ADO data control works just like the Data

control that we worked on earlier. However, the Data control cannot work with ADO, so we need to add the ADO data control.

Right click the Toolbox and from the popup menu select Components.

Figure 13.2 shows the Components dialog box.

NOTES

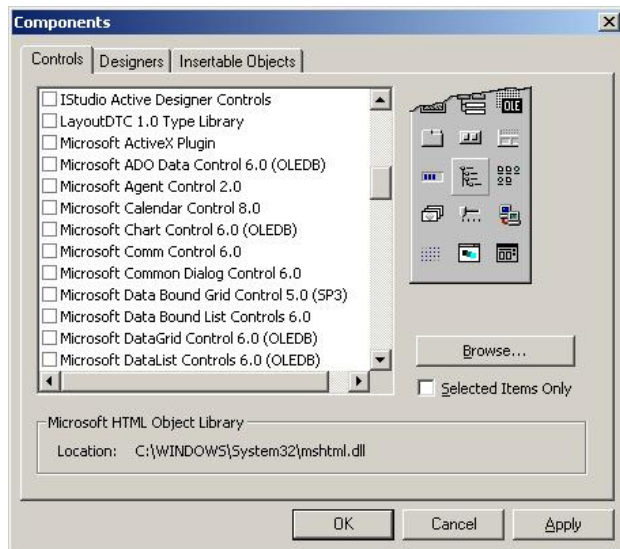


Fig. 13.2 Displaying Various VB Control Components

From this dialog box, click Microsoft ADO Data Control 6.0 (OLEDB). The ADO data control gets added to your Toolbox.

Draw the ADO data control on your form and set the properties. Right click the ADO Data control and select ADODC Properties from the menu. The Property Pages dialog box will look like this.

Figure 13.3 shows Property Pages dialog box

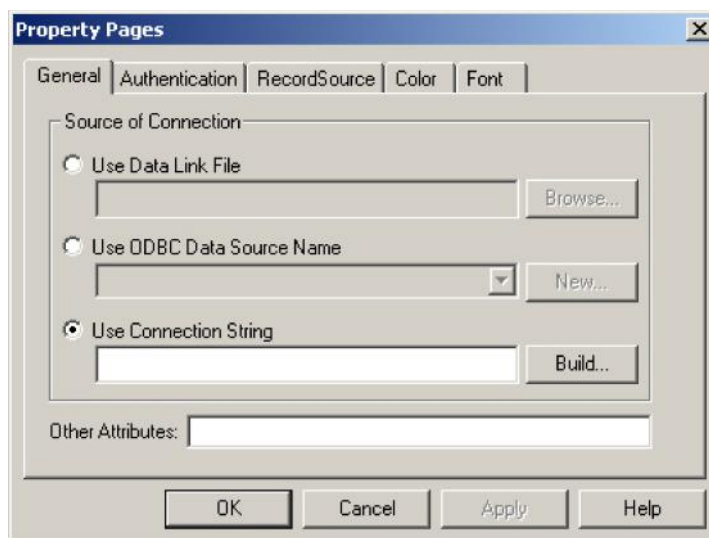


Fig. 13.3 Displaying Various ADODC Properties

NOTES

The Property Pages of the ADODC allow you to specify a lot more information than the Data control. In the case of the Data control, you only need to give the following four details:

- The type of database (Access, dBase, FoxPro...).
- The Name of the database.
- The type of recordset (Table, Dynaset, Snapshot).
- The RecordSource (A Table name, an SQL query...).

However, you may need to do a little more in the case of the ADODC, the Property Pages of the ADODC contain four tabs. They allow you to set the various properties of the ADODC. They are:

- **General:** In this tab, you specify how the ADODC should connect to a data source. There are three options.
 - o **Use Data Links File:** You will need this option if you are going to link a Textbox or a grid or some such control to an application like MS Excel or MS Word via DDE.
 - o **Use ODBC Data Source Name:** You can mention the name of the DSN that we created by using the ODBC Data Source Administrator. The DSNs already created will be displayed in a drop-down ListBox. You can select the one you need to work with or you can build a new DSN.
 - o **Use Connection String:** You can build the connection string here by clicking the 'Build' button. This will bring up a wizard and guide you along.
- **Authentication:** This lets you enter Authentication information like the User Name and Password.
- **RecordSource:** Here you can specify the method of creating a Recordset. That is, you can indicate the Command Type (`adCmdUnknown` or `adCmdText` or `adCmdTable` or `adCmdStoredProc`).

Table or Stored Procedure

If you choose `adCmdTable` in the Command Type, then in the Table or Stored Procedure you can select the table name from the database. If you choose the other options, then you have to enter the Stored Procedure or the SQL Command Text.

Font and Color

The other two tabs, Font and Color allow you to customise the appearance of the ADO data control.

Using Bound Controls

Data aware controls can be bound to the ADODC just as with the Data control. There are two new options that are made available with the ADO in Visual Basic 6.0. You can specify the Data Member and the Data Format along with the Data Source and the Data Field.

Cursors

Cursor is a set of pointer that points the data. A cursor is like a result set but the actual data is on the server. You can create a cursor by copying the data from the server to the client. However, the original data will be on the server only. A cursor is available for browsing, reporting and other uses until it is closed.

You can use various types of cursors provided by the VB and for each cursor type a specific constant is defined. The constants are used to invoke and initialise the various cursor types. Table 13.1 lists the various constants and their functioning.

Table 13.1 Various Constants and Their Description

Constant	Cursor Type	Description
adOpenDynamic	Invoke and initialise Dynamic type cursor	Enables you to view all the data changes, such as addition, deletion and changes performed by other users on a Recordset.
adOpenKeyset	Invoke and initialise Key type cursor	Enables you to view only some of the data changes on a Recordset performed by other users. For example, keytype cursor enables you to view the records that other users add to a Recordset.
adOpenStatic	Invoke and initialise static type cursor	Enables you to view a static copy of a Recordset. It means you cannot view any of the changes performed by other users on a Recordset.
adOpenForwardOnly	Invoke and initialise static type cursor	Behaves like the static type cursor except that it enables you to scroll forward only through the records in a Recordset.

NOTES

Using Cursor

The following sequence of steps implements an example to show how you can use a cursor type in your program.

NOTES

1. Open a new VB project.
2. Create four command buttons, two `TextBoxes` and two labels.
3. Change the caption and name of the four command buttons to First, Next, Previous and Last, respectively.
4. Delete the Text property of the two `TextBoxes`.
5. Change the caption of the two labels to Emp_name and Emp_ID as shown in the following figure.

Figure 13.4 shows Form1 Screen.

6. Now, create a database and create an Employee table in the database.
7. Enter some entries in the database as shown in the following figure.
Figure 13.5 shows database entries.
8. Create the Customer Data Source Name (DSN) that uses Microsoft Access driver for connecting your application to the MS Access database.

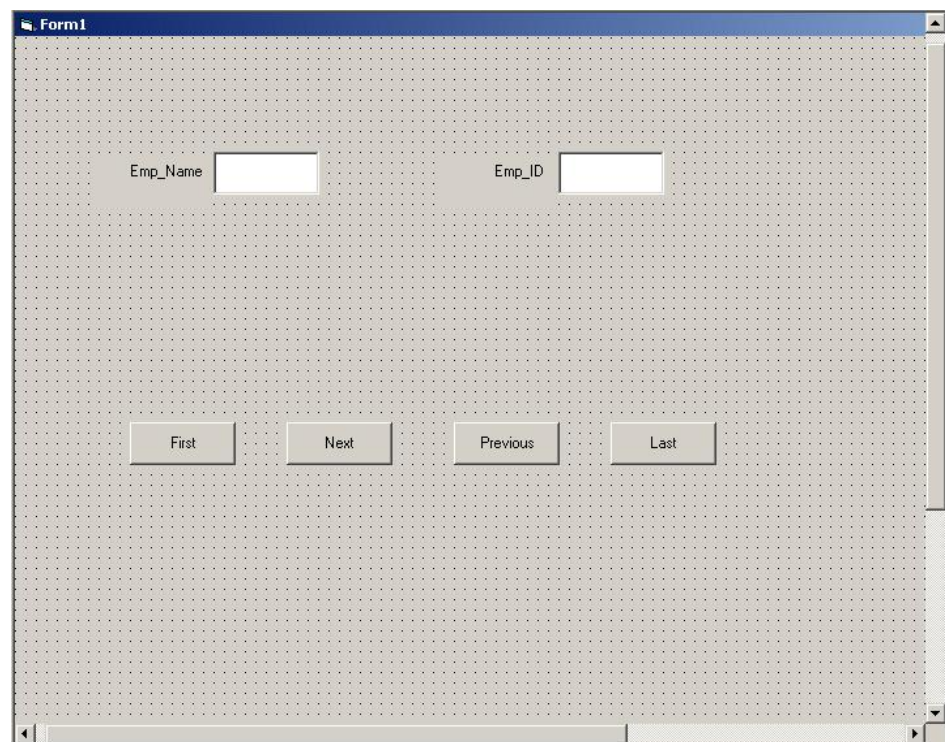


Fig. 13.4 Displaying all the Labels, TextBoxes and Command Buttons

Employee : Table		
	emp_name	emp_id
	John	23
	Ken	24
	smith	25
	akash	26
▶		◀

Fig. 13.5 Displaying all the Database Entries

NOTES

9. Now, open the Code window and write the following lines of code:

```
public con As New ADODB.Connection ' connection object is
    'declared to create a session with the database
public rs As New ADODB.Recordset
Private Sub First_Click()
rs.MoveFirst ' Moves the control to the first record in
    the 'database
Text1.Text = rs!emp_name ' print the emp_name value in
    the ' first text box
Text2.Text = rs!emp_id '' print the emp_name value in the
    'second text box
End Sub
Private Sub Form_Load()
Dim i As Integer
con.Open ("dsn=customer") ' Opens connection
rs.Open "select* from Employee", con, adOpenDynamic,
adLockOptimistic ' using adOpenDynamic cursor which enables
    'you to move in any direction in the database.
End Sub
Private Sub Last_Click()
rs.MoveLast ' Moves the control to the last record in the
    'database
Text1.Text = rs!emp_name
Text2.Text = rs!emp_id
End Sub
Private Sub Next_Click() ' implement to move the control
    to 'the next record in the database.
rs.MoveNext
```

NOTES

```
If rs.EOF = True Then
rs.MoveLast
End If

Text1.Text = rs!emp_name
Text2.Text = rs!emp_id
End Sub

Private Sub Previous_Click() ' implement to move the control
to 'the previous record in the database.
rs.MovePrevious
If rs.BOF = True Then
rs.MoveFirst
End If
Text1.Text = rs!emp_name
Text2.Text = rs!emp_id
End Sub
```

Figure 13.6 shows the Code window for using cursor

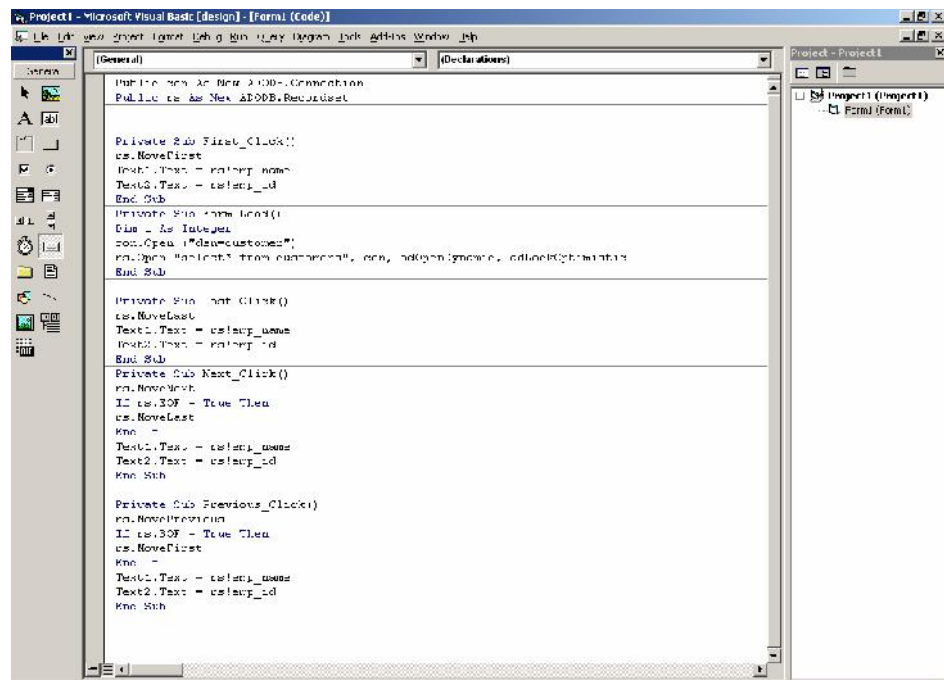


Fig. 13.6 Displaying the Code Window for Using Cursor

10. Add the Microsoft ActiveX Data Objects 2.1 Library reference to the program.

11. Compile the program. Now you can perform any of the above operations, such as First and Previous. For example, if you click the First button, then it will give the first entry data in the database. Figure 13.7 shows the first entry data in the database.

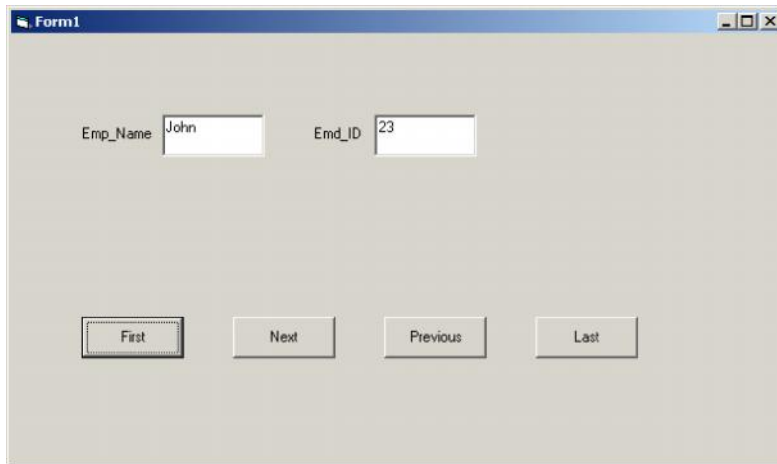
The image shows a standard Windows application window titled "Form1". Inside the window, there are two text boxes. The first text box is labeled "Emp_Name" and contains the text "John". The second text box is labeled "Emd_ID" and contains the text "23". Below these text boxes, there are four buttons arranged horizontally. From left to right, the buttons are labeled "First", "Next", "Previous", and "Last". The "First" button is highlighted with a dashed border, indicating it is the active or selected button.

Fig. 13.7 Displaying the Text Boxes Containing the First Entry Data in the Database

Locking

Locking is a technique that is used in a multiuser database environment to prevent users from editing one or more records on the database. For example, you are working in a multiuser database environment and want to edit database records. While editing the records, you need locking to prevent all the other users on the database from trying to edit the same record at the same time.

There are four different levels at which you can lock data in a database record. `LockType` property is used to control the locking levels and this property contains the following four values:

- **AdLockReadOnly:** Sets a record's data as read only type. Now, the users can only read the data on the records and cannot perform any change.
- **AdLockPessimistic:** Locks a record for all the other users when you are working on that record.
- **AdLockOptimistic:** Locks a record for all the other users when you call the `Update` method to modify that record.
- **AdLockBatchOptimistic:** Updates more than one record at a time with the `UpdateBatch` method call. Until you call the `UpdateBatch` method, all of your changes are cached locally.

NOTES

NOTES

The syntax to define values for the LockType property is:

```
<Recordset name>. LockType = <Value>
```

In the above syntax, the Recordset name refers to the name of the database Recordset. And Value refers to one of the four values, such as AdLockReadOnly or AdLockPessimistic.

Convert DAO Code to ADO

DAO to ADO Object Map

DAO	ADO (ADODB)	Note
DBEngine	None	
Workspace	None	
Database	Connection	
Recordset	Recordset	
Dynaset-Type	Keyset	Retrieves a set of pointers to the records in the recordset.
Snapshot-Type	Static	Both retrieve full records, but a Static recordset can be updated.
Table-Type	Keyset with adCmdTableDirect option.	
Field	Field	When referred to in a recordset.

DAO

Open a Recordset

```
Dim db as Database
Dim rs as DAO.Recordset
Set db = CurrentDB()
Set rs = db.OpenRecordset("Employees")
```

Edit a Recordset

```
rs.Edit
rs("TextFieldName") = "NewValue"
rs.Update
```

ADO

Open a Recordset

```
Dim rs as New ADODB.Recordset
rs.Open "Employees", CurrentProject.Connection, _
adOpenKeySet, adLockOptimistic
```

Edit a Recordset

```
rs("TextFieldName") = "NewValue"
rs.Update
```

Check Your Progress

1. What is the full form of ISAM?
2. Why we use API's in programming?
3. Write one service provided by OLE DB.
4. Explain the term Error for ActiveX Data Object model.
5. State about the Cursors for accessing ADO.
6. List the name of the properter used in locking.
7. Write the syntax to define values for the LockType property.

NOTES

13.3 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. RecordSet refers to the set of records (or simply a table) that are retrieved from an object as determined by RecordSource property.
2. Application Programming Interface (API) that enables applications to gain access and modify a wide variety of data sources.
3. OLE DB provides cursor service which is defined as a temporary, read only table that saves the results of a query with an assigned name. The cursor is available for browsing, reporting or other uses until it is closed.
4. Errors can occur at any time in your application due to the data source being corrupted or renamed by somebody or the Password being changed or for many other reasons that a programmer can understand. This results in not being able to establish a connection, execute a command or perform an operation on an object in a suitable state (for example, attempting to use a Recordset object that has not been initialised). This error object behaves like the error object in DAO.
5. Cursor is a set of pointer that points the data. A cursor is like a result set but the actual data is on the server. You can create a cursor by copying the data from the server to the client. However, the original data will be on the server only. A cursor is available for browsing, reporting and other uses until it is closed.
6. AdLockReadOnly,
AdLockOptimistic,
AdLockPessimistic,
AdLockBatchOptimistic.
7. The syntax to define values for the LockType property is:
<Recordset name>. LockType = <Value>

13.4 SUMMARY

NOTES

- The record being pointed to at any given moment is called the ‘current record’.
- The `RecordSet` object can also be used to modify a database by using its methods for it, such as `AddNew` (for adding a record), `Update` (for updating a record) and `Delete` (for deleting a record). Let us learn to use these methods through `RecordSet`.
- `EOF` (End of File) is a Boolean property of the `RecordSet` object that turns true when there is an attempt of moving forward passing the last record in a `RecordSet`.
- `BOF` (Beginning of File) is a Boolean property of the `RecordSet` object that turns true when there is an attempt of moving backward passing the first record in a `RecordSet`.
- In the above example you saw that changes to a record are automatically updated with the data control when the user moves off that record. This is called the **Update** method of the `RecordSet` object of the working data control.
- You also saw that if the `EOFAction` of the data control is set to ‘2-AddNew’, the **AddNew** method of the `RecordSet` object will be invoked by the data control, that causes the clearance of all the bound controls so that data can be entered by the user.
- The `Update` and `AddNew` methods can also be invoked through code in addition to being automatically invoked through the data control. There is also a **Delete** method of the `RecordSet` object.
- Which can only be invoked through code; it cannot be automatically accessed through the data control.
- Remote Data Objects (RDO) connect to databases using ODBC. You set up ODBC connections to databases using the ODBC item in the Windows Control Panel and then use one of those connections with the RDO.
- ActiveX Data Objects (ADO) is the new data access technology offered by Microsoft.
- ADO is meant to replace Data Access Objects (DAO), Remote Data Objects (RDO) and Open Database Connectivity (ODBC).
- This means that all new programmers must embrace this new offering from Microsoft.
- OLE DB is a set of Component Object Model (COM) interfaces that provide uniform access to the data stored in diverse information sources.

- OLE DB is defined as a new low level interface that is a part of the Universal Data Access platform.
- It is defined as a general purpose set of interfaces designed to let developers build data access tools as components using the COM.
- ADO enables your client applications to access and manipulate data in a database server through any of the OLE DB providers.
- According to Microsoft, “ADO’s primary benefits are ease of use, high speed, low memory overheads and a small disk footprint.
- ADO supports key features for building client-server and Web based applications.”
- The ADO features an object model like the DAO and the RDO but it is much flatter.
- In the case of the DAO, you had seventeen objects and in the case of the ADO, you have only seven objects. Besides, you do not have to follow a strict hierarchy when working with these objects in ADO.
- For example, unlike in the case of ADO, in the DAO you cannot create a recordset without using the database object.
- Specifies columns, tables and values in an SQL command as variable parameters.
- Executes a command and store the result, if the command is returning row, in a cache. `Recordset` also allows a user to sort, view or edit the data and up-to-date the data source.
- You can access a data source using a `Connection` object. A `Connection` represents an open session or `Connection` to a data source.
- A `Connection` object specifies the name of a data source, the provider that will be used to access the data and other parameters.
- Your application can gain access to a data source directly (sometimes called a two-tier system) or indirectly (sometimes called a three-tier system) through an intermediary like the Internet Information Server.
- Once a connection has been established with the data source, the data has to be extracted.
- This is done by using a `Command` object. A `Command` adds, deletes and updates data in the data source or retrieves data in the form of rows in a table.
- A command to retrieve data can be qualified by using parameters. Parameters are arguments to a command that alter the result of the execution of the command.

NOTES

NOTES

- Although ADO allows you to access any type of data, here our discussion is limited to the data from a database. The `Command` object when executed will return a set of rows from one or more tables. This set of rows is called `Recordset`.
- A row of a `Recordset` consists of one or more fields. If you envision a `Recordset` as a two-dimensional grid, the fields line up to form columns.
- Errors can occur at any time in your application due to the data source being corrupted or renamed by somebody or the Password being changed or for many other reasons that a programmer can understand.
- Each ADO object has a set of unique properties that either describes or controls the behaviour of that object. There are two types of properties: built-in and dynamic.
- Just as in DAO, ADO provides collections, a type of object that contains other objects of a particular type.
- This is new in ADO. ADO 2.0 introduces the concept of events to a programming model. Events are notifications that certain operations are about to occur or have already occurred.
- Events are issued when transactions on a connection begin, committed or rolled back, when commands execute and connections start or end.
- Events are issued to report the progress of data retrieval in the following cases: when you navigate through the rows of a `Recordset` object, when you change a field in a row of a `Recordset`, when you change a row in a `Recordset` or make any change in the entire `Recordset`.
- We can display the data from a `Recordset` (data source) using ADO code or with the help of the ADO data control. In order to use the ADO Data, we need to add the control to the form.
- You will need this option if you are going to link a Textbox or a grid or some such control to an application like MS Excel or MS Word via DDE.
- The other two tabs, Font and Color allow you to customise the appearance of the ADO data control.
- Cursor is a set of pointer that points the data. A cursor is like a result set but the actual data is on the server.
- Locking is a technique that is used in a multiuser database environment to prevent users from editing one or more records on the database. For example, you are working in a multiuser database environment and want to edit database records.

13.5 KEY WORDS

- **ADO Data Control:** It is the principal interface between a VB application and a database, which can be used without writing any code at all.
- **Data Control:** It is a link between information in the user's database and the bound controls that the user uses to display the information.
- **RecordSet:** It refers to the set of records (or simply a table) that are retrieved from an object as determined by RecordSource property.
- **Parameter:** A parameter represents a value that the procedure expects you to pass when you call it.
- **Snapshots-type RecordSet:** It is a static copy of a set of records that the user can use to find data or generate reports.
- **Cursor:** Cursor is a set of pointer that points the data. A cursor is like a result set but the actual data is on the server. You can create a cursor by copying the data from the server to the client.
- **Locking:** Locking is a technique that is used in a multiuser database environment to prevent users from editing one or more records on the database.

NOTES

13.6 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. Define the term ActiveX Data Objects.
2. State about the Remote Data Objects.
3. What is OLE DB?
4. Why we use OLE DB?
5. Elaborate on the ActiveX Data Object Model.
6. Explain the term Accessing ADO data control.
7. Define the term locking.

Long-Answer Questions

1. Describe the various steps of accessing data through ADO control.
2. What are RecordSet and its types? Explain accessing, navigating, modifying data in a database through RecordSet .
3. Discuss briefly adding, updating and deleting records with the help of example.

NOTES

4. Write the advantages and disadvantage of DAO, RDO and ADO.
5. Discuss ActiveX data object model and the activities that are performed by the classes and objects of ADO model.
6. Briefly describe the cursors. Discuss the steps a cursor type in programs. Give code to support you answers.
7. Describe the term Locking and its properties.

13.7 FURTHER READINGS

- Cornell, Gary. 1998. *Visual Basic 6 from the Ground Up*. New Delhi: Tata McGraw-Hill.
- Warner, Scott L. 1998. *Teach Yourself Visual Basic 6*. New Delhi: Tata McGraw-Hill.
- Jerke, Noel. 1999. *Visual Basic 6 - The Complete Reference*. New York: McGraw-Hill.
- Smith, Eric A., Valor Whisler and Hank Marquis. 1998. *Visual Basic 6 Programming Bible*. New Jersey: John Wiley & Sons, Inc.
- Azam, M. 2001. *Programming with Visual Basic*. New Delhi: Vikas Publishing House Pvt. Ltd.
- Manchanda, Mahesh. 2009. *Visual Programming*. New Delhi: Vikas Publishing House Pvt. Ltd.
- Balena, Francesco. 1999. *Programming Microsoft Visual Basic 6.0*. Bangalore: WP Publishers and Distributors (P) Ltd.
- Petroutsos, Evangelos. 1998. *Mastering Visual Basic 6*, 1st Edition. New Delhi: BPB Publications.
- Deitel, Harvey M., Paul J. Deitel and T. Tem R. Nieto. 1999. *Visual Basic 6: How to Program*. New Jersey: Prentice-Hall.
- Norton, Peter. 1998. *Peter Norton's Complete Guide to Visual Basic 6*. New Delhi: Techmedia.
- Reselman, Bob and Richard A. Peasley. 1998. *Using Visual Basic 6*. New Jersey: Pearson Education (Que Publishing).
- Donald, Bob and Oancea Gabriel. 1999. *Visual Basic 6 from Scratch*. New Delhi: Prentice-Hall of India.

UNIT 14 CONNECTING TO THE DATABASE

NOTES

- 14.0 Introduction
- 14.1 Objectives
- 14.2 Database Connectivity
- 14.3 Retrieving a Recordset
 - 14.3.1 What is Data Provider?
 - 14.3.2 What is OLE DB?
 - 14.3.3 Accessing Database using ADO Control
- 14.4 Working with Queries
 - 14.4.1 Parameterized Queries
 - 14.4.2 Action Query
- 14.5 Adding Records and Editing Records
 - 14.5.1 ADO: Adding a Record to a Record Set
 - 14.5.2 ADO: Editing a Record in a Record Set
- 14.6 Closing the Database Connection
- 14.7 Answers to Check Your Progress Questions
- 14.8 Summary
- 14.9 Key Words
- 14.10 Self-Assessment Questions and Exercises
- 14.11 Further Readings

14.0 INTRODUCTION

Applications communicate with a database for two basic functions or purposes. First function is to retrieve the data stored in the database and to present it in a user-friendly manner, and second function is to update the existing database by inserting the data, modifying the data and deleting the data that is not required.

Microsoft ActiveX Data Objects.NET (ADO.NET) is considered as a model, a part of the .NET framework that is specifically used by the .NET applications for retrieving, accessing and updating data. The data that resides or exists in a database is retrieved through the data provider. Various components of the data provider retrieve the data and also update the data.

ADO.NET is a large set of .NET class that enable us to retrieve data, manipulate data, and update data sources in various ways. As an integral part of the .NET framework, it shares many of its features, such as multi-language support, garbage collection, just-in-time compilation, object-oriented design, and dynamic caching. It has upgraded features in comparison to the previous versions of ADO. Consequently, the ADO.NET is considered as a core component of any data-driven .NET application or Web Services.

NOTES

Furthermore, the ADO.NET is a part of the .NET framework architecture. It is a model used by .NET applications to communicate with a database for retrieving, accessing, and updating data.

In this unit, you will study about the database connectivity, retrieving a recordset, creating a query dynamically, using a parameterized query, using action queries, saving, adding, editing and closing the database connection.

14.1 OBJECTIVES

After going through this unit, you will be able to:

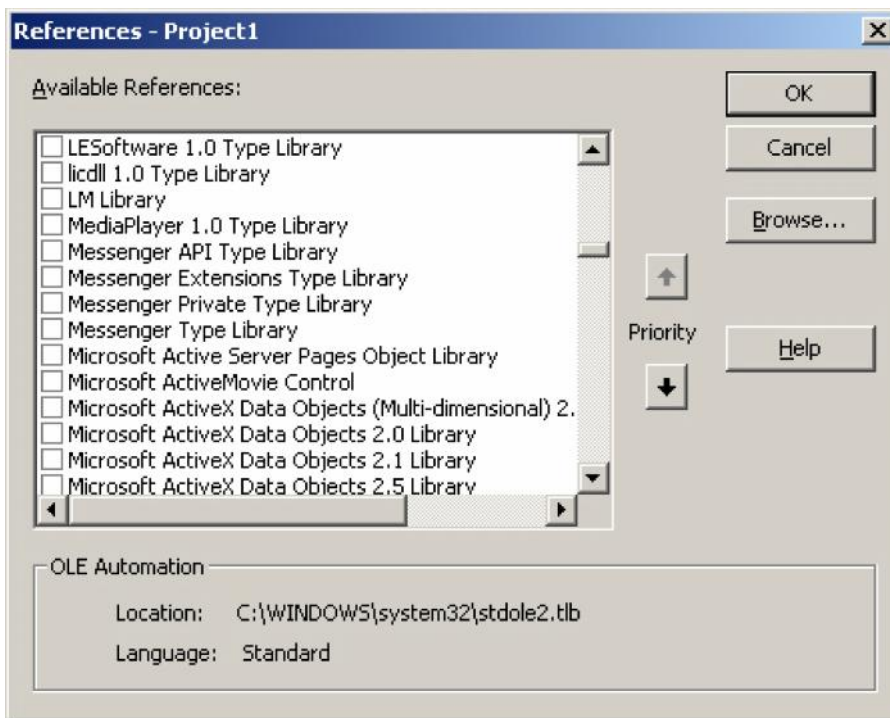
- Connect database to Visual Basic applications
- Elaborate on the concept of editing, saving, adding and deleting records
- Discuss the significance of retrieving RecordSet from the database
- Create a query dynamically
- Use the parameterized query and action queries
- Explain about the browsing of data from the data source
- Close the database connection

14.2 DATABASE CONNECTIVITY

Microsoft's ActiveX Data Objects (ADO) is the new data access technology developed by Microsoft. It comprises of a set of Component Object Model (COM) objects for accessing data sources. As a significant part of MDAC (Microsoft Data Access Components), ADO provides a middleware layer between programming languages and OLE DB (Object Linking and Embedding, DataBase), a means of accessing data stores, whether databases or not, in a uniform manner. ADO allows to write programs for accessing data from the data source without even knowing how the database is implemented, but the programmer must know about the database connection. Microsoft introduced ADO in October 1996, positioning the software as a successor to Microsoft's earlier object layers for accessing data sources, including RDO (Remote Data Objects) and DAO (Data Access Objects).

ADO is supported in any development language that supports binding to binary COM interfaces. These languages include ASP (Active Server Pages), Delphi, PowerBuilder, and Visual Basic for Applications (VBA). ADO.NET has replaced ADO as the primary mode for targeting Windows application development. ADO.NET follows the same design pattern as ADO, enabling an ADO developer an easy path forward when moving to the .NET framework.

Open a new project to connect it to a data source by using ADO. To use ADO in your project, you have to make a reference to it. For this, click Projects and from the menu select References.



NOTES

From the list displayed in the References dialog box, select the Microsoft ActiveX Data Objects 2.0 Library and the Microsoft ActiveX Data Objects Recordset 2.0 Library. Now you can use ADO in your project.

In order to achieve our objective of accessing a data source, extracting a set of records from it and manipulating or editing the RecordSet and finally updating the data source, we have to follow the steps given below:

- Make a connection to a data source.
- Create a command to specify the records to be extracted.
- Execute the command.
- Navigate and edit the data in the RecordSet (We assume that the data is returned as a RecordSet).
- Update the data source with changes made to the data in the RecordSet.

Please note that while using ADO, we do not have to strictly follow the hierarchy.

The Data Source

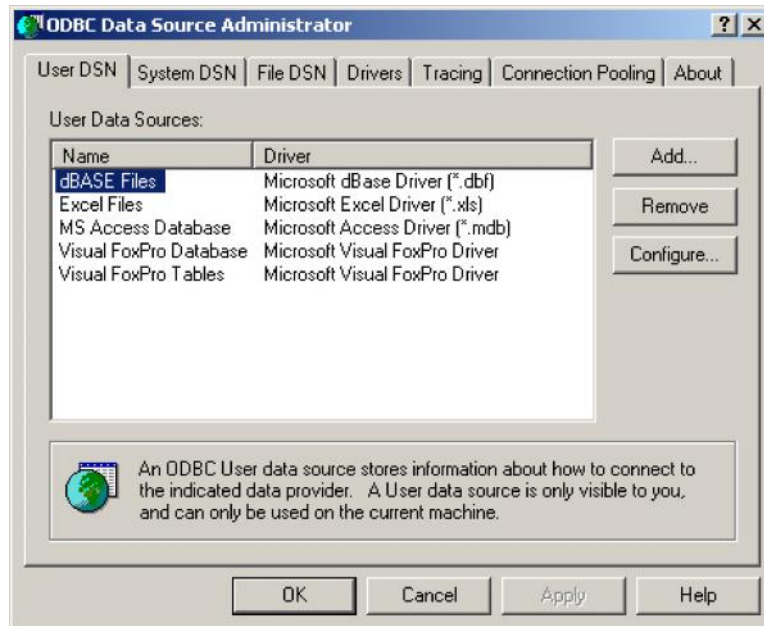
What we need now is a data source. Let us use the Invoice.mdb as our data source. After deciding that Invoice.mdb is our data source, we need to define the data source by using the Open Data Base Connectivity (ODBC) Data Source Administrator.

NOTES

The ODBC Data Source Administrator

ODBC is defined as “A standard protocol for database servers. ODBC has drivers for various databases that enable the applications to connect to the databases and access their data.” The condition is that these databases must have Structure Query Language (SQL) as the standard for data access.

From the Control Panel, double-click the Administrative Tools → ODBC icon.



Click the Add button to add a data source. Another dialog box will be displayed, asking you to select the driver.



Select Microsoft Access driver (*.mdb) since we are going to work on our Invoice.mdb. Click Finish. The next dialog box displayed will ask you to specify the name of the database.



NOTES

Click Select to choose the name of the .mdb file. After selecting the .mdb file, enter the name of the data source. You will be using this name as the DSN (Data Source Name). Click OK and exit from the ODBC administrator.

We will use the Data Source Name given here in the following example.

Using the Data Source Name in Our Project

In the General declaration, add the following lines of code:

```
Dim adocon As New ADODB.Connection
Dim rs As Recordset
Dim strconnect As String
```

The first line declares and sets 'adocon' as an ADODB connection object. You can declare the above as follows:

```
Dim adocon as ADODB.Connection
```

In the Form_Load event you can say

```
Set adocon = New ADODB.Connection
```

We have declared the connection object. Now to set the connection to a data source, add the following lines of code in the Form_Load event:

```
adocon.ConnectionString = "DSN=dsname" \
```

This line will set the connection string property of the 'adocon' object to the data source name. Remember that the name of the Data Source Name entered in the ODBC administrator must be given here. The next line opens the database.

```
adocon.Open
```

Add the line "MsgBox adocon.DefaultDatabase" in the above code and check if the database has been opened.

Run the program and checkout if you have managed to set up a connection with the data source using ADO.

NOTES

14.3 RETRIEVING A RECORDSET

To provide access to ADO `RecordSet` and `Record` objects from ADO.NET, the OLE DB .NET Data Provider overloads the `Fill` method of the `OleDbDataAdapter` to accept an ADO `RecordSet` object or an ADO `Record` object. Filling a `DataSet` with the contents of an ADO object is a one-way operation. That is, data can be imported from the ADO `RecordSet` or `Record` into the `DataSet`, but any updates to the data must be handled explicitly by either ADO.NET or ADO.

14.3.1 What is Data Provider?

A data provider is a control or object or mechanism that provides data for use by connecting to a source of data, for example a database or a text file. The data provider makes data connectivity much easier by hiding most of the implementation of data storage.

14.3.2 What is OLE DB?

OLE DB (Object Linking and Embedding, DataBase) is the underlying system service that a programmer using ADO actually uses. OLE DB is a set of interfaces that provides applications with uniform access to data stored in diverse information sources or data stores. OLE DB is suitable for relational and non-relational data sources. That is, with OLE DB, you can access all types of databases in the same manner.

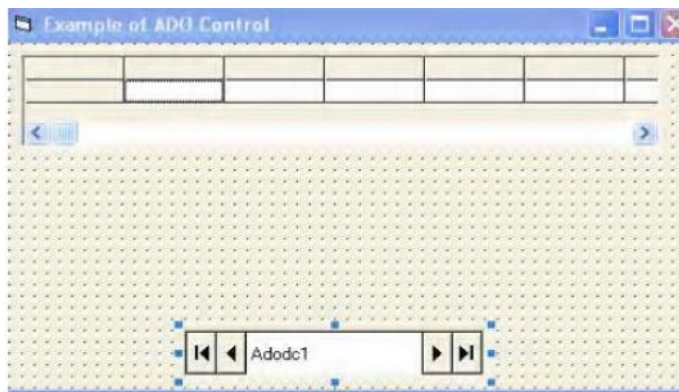
14.3.3 Accessing Database using ADO Control

We can access Microsoft Access database using ActiveX Data Object Data Control (ADO DC). You can create VB applications using ADO DC in two ways:

- Employing ADO DC interactively using VB connection Wizard Data Access Using ADO.
- Writing code, i.e., programming ADO (ADO DB).

Setting Up and using ADO Data Control

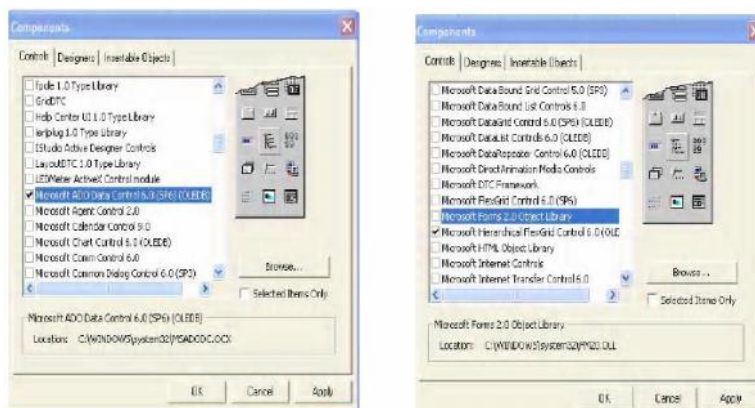
Design a Visual Basic (VB) application that displays the records of table Author using the Microsoft Access database option.



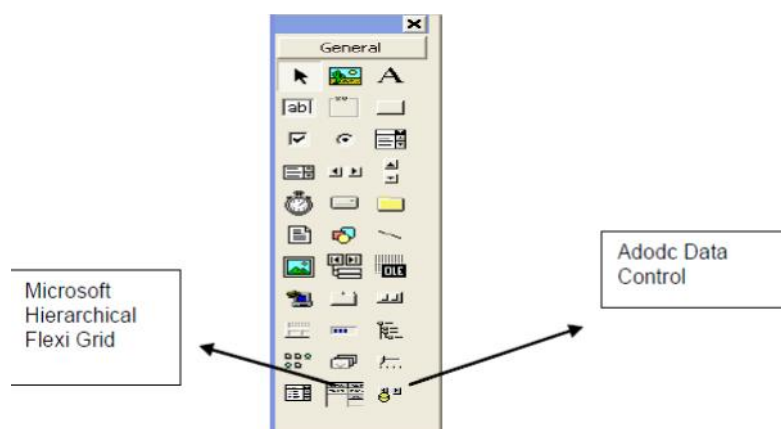
NOTES

1. Start up VB with a Standard EXE Project.
2. Go to the Project menu and select Component (shown in the screenshot given below). Now add two new controls to your project.

- The Microsoft ADO Data Control (OLE DB)
- The Microsoft Hierarchical Flex Grid Control 6.0



You will find that your toolbox is now populated with two more icons as shown in the screenshot given below.

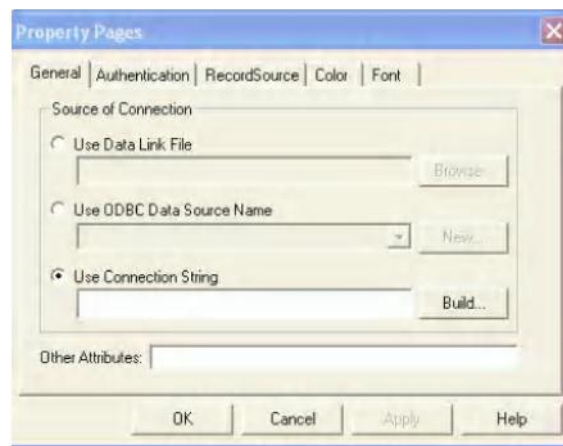


3. Now change the Caption of your main form to 'Example of ADO Control'.

NOTES

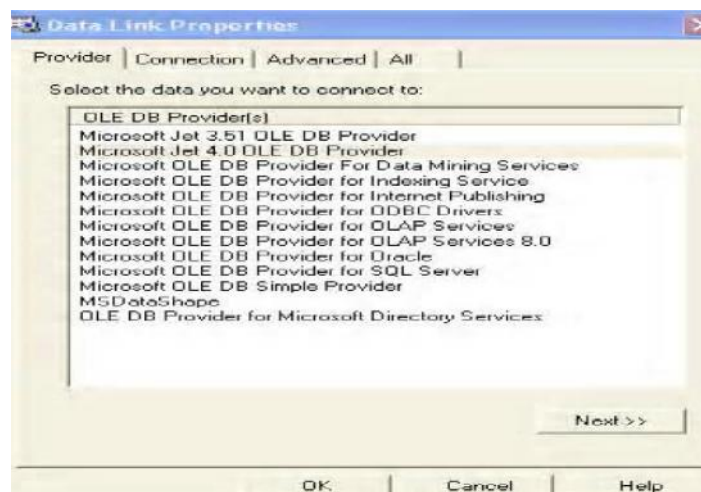
Setting Up ADO Control

4. Add an ADO data control to the form using the ADO DC tool and drawing it on the form. Name it as Adodc1 and change its Visible property to false.
5. Right-click this control and select the ADODC properties option from the shortcut menu that appears.
6. This will invoke the **Property Pages** (refer screenshot given below) for the ADO DC control that you added in your project. Now, under the **General** tab, select the source of connection as Use Connection String and click on the **Build** button. Alternatively, just select the ADO Data control and click the expression builder button in its **Connection** property or click in its (Custom) property.

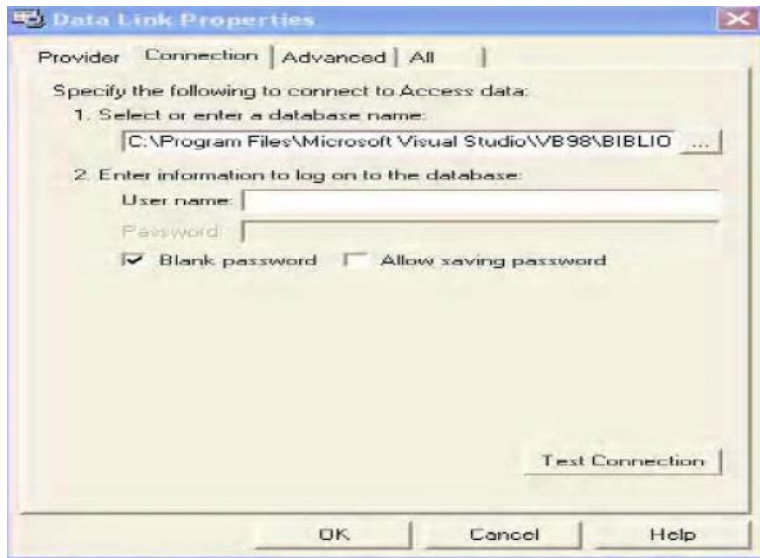


Connect to Database

7. Now go to another **Dialog Box** – **Data Link Properties** – **Pops Up**. Select Microsoft Jet 4.0 OLE DB provider for Microsoft Access from the list that appears and click next.



8. Since you do not have to choose from multiple servers, just type a valid user name and password entry in the dialog that appears as shown below in the screenshot. The new window will appear, click on the Test Connection button.



Now, this VB wizard will setup a connection to the Microsoft Access database. If the connection test is successful, the following Dialog Box will appear.



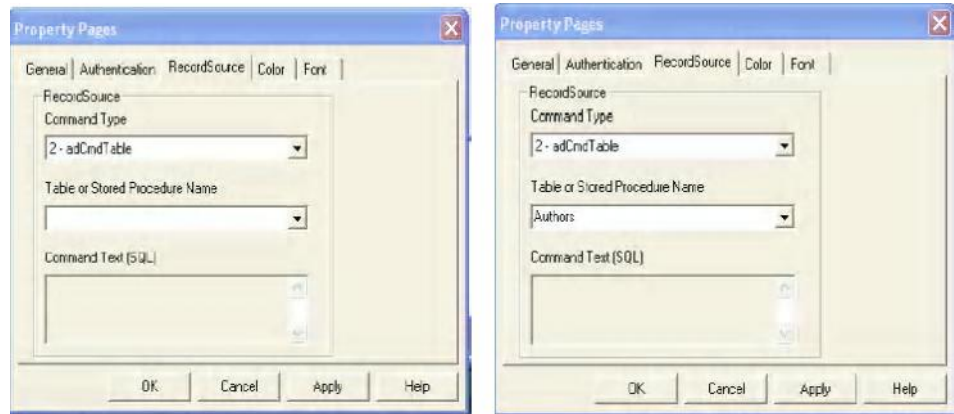
9. Click OK in the **Data Link Properties Dialog Box** and then again in the **Property Pages Dialog Box**.

Specifying the Record Source

10. Now, once again in the **Property Pages** (bring it by right-clicking the ADO control and selecting properties), under the **RecordSource** tab, specify the RecordSource by first selecting the **Command Type** as **2-adCmdTable** and then by selecting the desired data table from the database, as shown below.

NOTES

NOTES

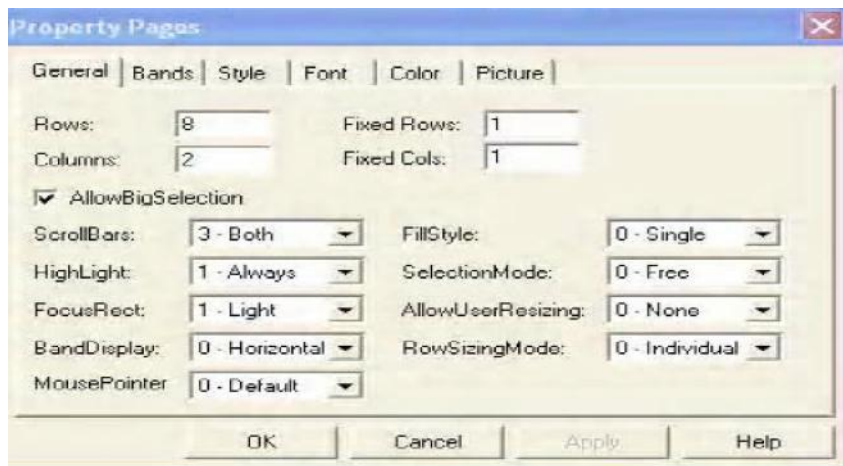


The **Command Type** basically specifies the type of data source to which the ADO control is attached. It can take any of the following values as shown in the Table 14.1.

Table 14.1 Values Taken by Command Type

Constant	Description
adCmdText	The Source contains a SQL command, such as a SELECT statement.
adCmdTable	The Source string contains the name of a table to be retrieved. (ADO itself creates a SQL query.) Using this option is not recommended because it can pull in too many records. But here, since we know that there are many records in Authors table, we have used it.
adCmdStoredProc	The Source string contains the name of a stored procedure (not applicable to Oracle).
adCmdUnknown	This constant indicates that the type of command in the Source argument is not known. Using this option is inefficient.

- After setting up the ADO control, add a Hierarchical Flex Grid to your form by picking the HflexGrid tool and drawing it on your main form. The HflexGrid helps to show data in the tabular form.
- Set the Name property of HflexGrid to HfGrid1.
- Select the HflexGrid control and in the Properties window, first of all, set its **DataSource** property to Adodc1, the name of ADO data control must be included in the project (refer screenshot given below).
- Now to set other properties, bring its **Property Pages** Dialog Box by Right-clicking on the HflexGrid control and selecting the **Property** option (as shown below). Further, under the General tab of Property Pages, set number of Columns to 8, fix Rows to 1, and fix Cols to 1 and click OK. You can adjust the height and width of the control on the form by dragging its handles.



NOTES

15. Save your project and run it.

Check Your Progress

1. Explain about Microsoft's ActiveX Data Objects (ADO).
2. List out the steps for accessing a data source, extracting records and updating the data source.
3. Write down the full form of ODBC.
4. Give definition of ODBC.
5. Explain about the data provider.
6. What do you mean by OLE control?
7. State the ways to create VB application using ActiveX Data Object Data Control (ADO DC).

14.4 WORKING WITH QUERIES

A query is a way of requesting information from the database. A database query can be either a select query or an action query. A select query is a query for retrieving data, while an action query requests additional actions to be performed on the data, like deletion, insertion, and updating. Visual Basic supports SQL (Structured Query Language) to declare and run required queries.

Dynamic SQL is a programming technique that enables to build SQL statements dynamically at runtime. More general purpose and flexible applications can be created using the Dynamic SQL because the full text of a SQL statement may be unknown at compilation. For example, Dynamic SQL helps in creating a procedure that operates on a table whose name may not be known till runtime.

SQL Server provides different methods for running a dynamically built SQL statement. The Dynamic SQL query can be written with parameters. The simple

NOTES

method is, within the declaration you can pass parameters into your WHERE clause of your SQL statement. Consider the following example for finding all records from the customers table where City = 'DELHI'. The example code is,

```
DECLARE @city varchar(75)
SET @city = 'DELHI'
SELECT * FROM Person.Address WHERE City = @city
```

14.4.1 Parameterized Queries

Parameterized queries are basically the same as an ordinary query, but they allow you to make use of parameters inside your queries. A parameter is additional information you can provide to an SQL query. This information usually works in conjunction with a condition inside the SQL query.

A simple example of an SQL Query containing a parameter looks like this:

```
"Select * From StudentInfo where StudentName = @Name"
```

In this case, Name is a parameter. To provide a value to the parameter, you can use the following VB.NET code:

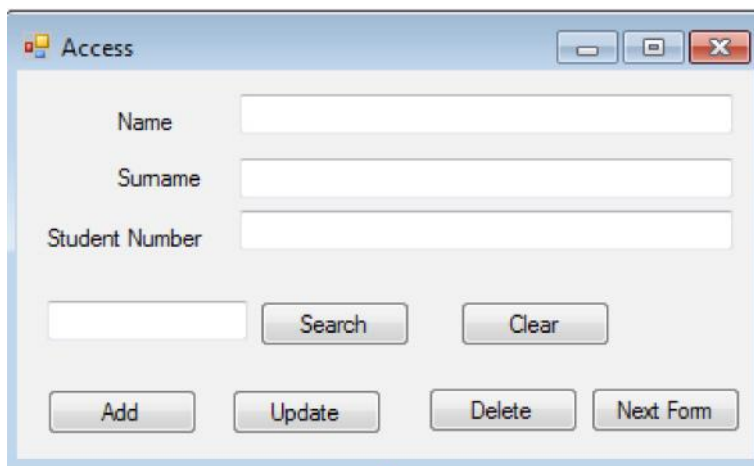
```
'Create Parameter Instead Of Hardcoding Values
'Name = whatever txtSearch Contains
oleAccCommand.Parameters.AddWithValue ("Name",
txtSearch.Text)
```

In the above code we have added a parameter to a command called Name and provide the value of whatever text has been entered into the txtSearch TextBox. The user can now type any value inside the txtSearch TextBox, and it will be added to the final query that will run on the database. Another way to add parameter in the code is as follows:

```
command.Parameters.Add ("@StudentNumber", SqlDbType.Int)
command.Parameters ("@StudentNumber").Value =
txtStudentNumber.Text
```

In the above code, in the first line the parameter is created to set its data type. In the next line, a value is added automatically when the code is executed. Following example will make the concept clear.

To get a better understanding of how parameters work, we will make use of **Visual Basic 6.0**. Start a new **Windows Forms Project**. Following figure will demonstrate how to use Parameters within your queries to a Microsoft Access DataBase.



NOTES

Create a table named **StudentInfo** inside a database named **Students**, and then create the following three fields:

1. StudentName, which has a Text Data Type.
2. StudentSurname, which has a Text Data Type.
3. StudentNumber, which has Numeric Data Type.

Navigate to your desired Form's code window and add the following Imports statement to import the Namespace equipped to handle Microsoft Access DataBase:

```
Imports System.Data.OleDb 'Import Access Db Handling Capabilities
```

Add the following modular variables, which can be used in the form(s):

```
'Access Database Connection String
Dim strAccConn As String =
"Provider=Microsoft.ACE.OLEDB.12.0; Data
Source=C:\Users\HannesTheGreat\Documents\Students.accdb;Persist
Security Info=False;"

'Object To Use As SQL Query
Dim strAccQuery As String

'Access Database Connection
Dim oleAccCon As OleDbConnection
```

A string object is created with **Microsoft Access Connection String Information**. Specify the exact location of the Microsoft Access Database as well as security information.

Add the **Form_Load** event:

```
Private Sub Form1_Load (sender As Object, e As EventArgs)
Handles MyBase.Load
```

NOTES

```
`Search SQL Query
`This Query Simply Retrieves All Information
strAccQuery = "Select * From StudentInfo"

`Instantiate Connection Object
oleAccCon = New OleDbConnection(strAccConn)

`Using Structure Simplifies Garbage Collection And
Ensures That The Object Will Be Disposed Of Correctly
Afterwards
Using oleAccCon

`Create Command Object, To Make Use Of SQL Query
Dim oleAccCommand As New
OleDbCommand(strAccQuery, oleAccCon)

`Open Connection To The Database
oleAccCon.Open()
`Reader Object To Traverse Through Found Records
Dim oleAccReader As OleDbDataReader =
oleAccCommand.ExecuteReader()

`If The Reader Finds Rows
If oleAccReader.HasRows Then

`Retrieve The Content, For Each Match
While oleAccReader.Read()

`GetString(1) Represents Column 2 Of
StudentsInfo table
txtName.Text = oleAccReader.GetString(1)

`GetString(2) Gets Information Stored In
Third Column
txtSurname.Text =
oleAccReader.GetString(2)

`Use GetValue or GetInt32 Here, Because
StudentNumber Is A Number Field
txtStudentNumber.Text =
oleAccReader.GetValue(0)
```

```

        End While

    Else

        'No Match Was Found
        MessageBox.Show("No rows found.")

    End If

    'Close Reader Object
    oleAccReader.Close()

End Using
End Sub

```

NOTES

14.4.2 Action Query

Action queries are queries that can make changes to many records at once. They are used to delete records, to update records, i.e., to change values in the records, to create new tables, to delete tables and to define queries that accept a user defined parameter.

```

Dim con As New ADODB.Connection
Dim rs As New ADODB.Recordset
Public Sub procedure1()
    con.Provider = "Microsoft.jet.oledb.4.0"
    con.Open(My.Application.Info.DirectoryPath &
"\Database1.mdb")
End Sub

Private Sub Button1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button2.Click

    Dim str As String = InputBox("enter first
name to search")
    Dim query As String = "Select * from ADODB
where firstname=" & str
    procedure1()
    rs.Open(query, con, 2, 3)
    If rs.EOF = True Or rs.BOF = True Then
        MsgBox("record not found")
    End If
End Sub

```

NOTES

```
Else
    Me.TextBox1.Text = rs.Fields(1).ToString
    Me.TextBox1.Text = rs.Fields(2).ToString
    Me.TextBox1.Text = rs.Fields(3).ToString
End If
rs.Close()
con.Close()
rs = Nothing
con = Nothing

End Sub
```

14.5 ADDING RECORDS AND EDITING RECORDS

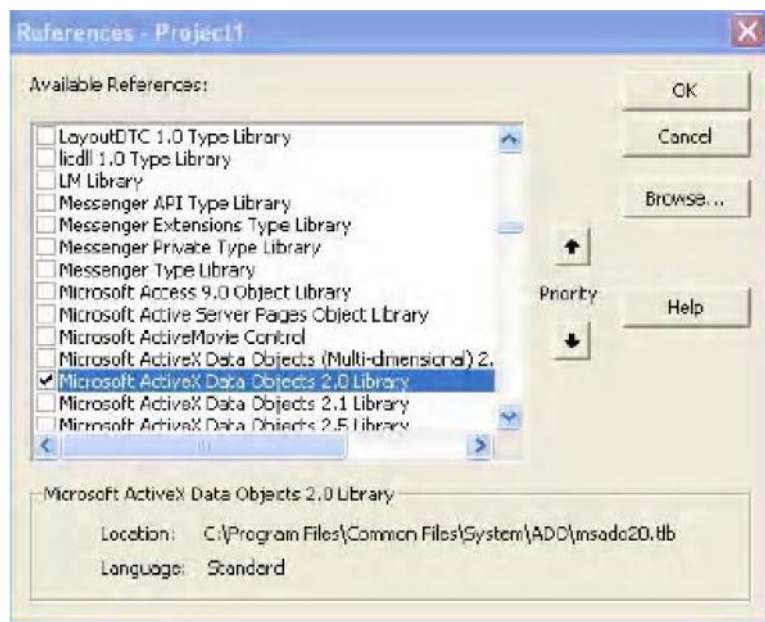
After using ActiveX Data Object (ADO) Data control, let us now learn to use ADO programmatically.

Earlier, when you were using ADO, programming was still being done, but not by you; it was being done by the Visual Basic (VB) connection wizard. That is, implicit programming was occurring. In this section, we shall be learning how to program explicitly.

Set Reference to Microsoft ActiveX Library

Before you can start writing code for ADO, make sure to set up a reference to

Microsoft ActiveX Library 2.x (x can be any number representing the version) by following command Project à References as shown in the following screenshot.



14.5.1 ADO: Adding a Record to a Record Set

To add a new record to an ADO recordset, you use the `AddNew` method. After you have updated the fields of the current record, you save that record to the database with the `Update` method. You can use `AddNew` method as shown below.

```
recordset.AddNew [ Fields [, Values]]
```

Here are the arguments for this method:

- **Fields** — A single name or an array of names or ordinal positions of the fields in the new record.
- **Values** — A single value or an array of values for the fields in the new record. If **Fields** is an array, **Values** must also be an array with the same number of members.

The order of field names must match the order of field values in each array.

Let us see an example. Now, we will add a new record in the recordset table using the `adoRecordSet`. In our ADO code example, the records are added when the user clicks the appropriate button. Following is the example code:

```
Private Sub cmdAdd_Click()  
On Error GoTo ErrLabel  
adoRecordset.AddNew  
Text1.Text = ""  
Text2.Text = ""  
Exit Sub  
ErrLabel:  
MsgBox Err.Description  
End Sub
```

Note that we also clear the two text boxes that display the field data, `Text1` and `Text2`, so that users can enter the data they want in the new record. When done, the `Update` button is pressed to update the data source.

14.5.2 ADO: Editing a Record in a Record Set

After changing the data in a record's fields or adding a new record, you update the data source to record the change, using the `Update` method:

Recordset. `Update Fields, Values` Here are the arguments for this method:

- **Fields**—A single name or an array of names or ordinal positions of the fields in the new record.
- **Values**—A single value or an array of values for the fields in the new record.

If **Fields** is an array, **Values** must also be an array with the same number of members. The order of field names must match the order of field values in each

NOTES

NOTES

array. Let us see an example. The users can update records in the ADO code example, by clicking the appropriate button. The data source can be updated using the following example code:

```
Private Sub cmdUpdate_Click ()  
On Error GoTo ErrLabel  
adoRecordset.Update  
Exit Sub  
ErrLabel:  
MsgBox Err.Description  
End Sub
```

The records can be updated in an ADO recordset.

14.6 CLOSING THE DATABASE CONNECTION

Using the **Close** method to close a **Connection** object also closes any active **Recordset** objects associated with the connection. A **Command** object associated with the **Connection** object you are closing will persist, but it will no longer be associated with a **Connection** object; that is, its **ActiveConnection** property will be set to **Nothing**. Also, the **Command** object's **Parameters** collection will be cleared of any provider-defined parameters.

You can later call the **Open** method to re-establish the connection to the same, or another, data source. While the **Connection** object is closed, calling any methods that require an open connection to the data source generates an error.

Closing a **Connection** object while there are open **Recordset** objects on the connection rolls back any pending changes in all of the **Recordset** objects. Explicitly closing a **Connection** object (calling the **Close** method) while a transaction is in progress generates an error. If a **Connection** object falls out of scope while a transaction is in progress, ADO automatically rolls back the transaction.

```
rs.close  
set rs=nothing  
cn.close  
set cn=nothing
```

Developing Applications Using Adodb

Design a small application that lets you Add, Modify, Delete and Cancel operations on two tables Emp and Dept of Oracle database. The application must also allow you to navigate the records. Also, it should provide options to view the complete

table in one go. There should be a main menu from where the other forms should get invoked.

Connecting to the Database

NOTES

The 'Employee Data' form is a data entry window with a dotted background. It features six input fields arranged in two columns: 'Emp No.', 'Name', 'Job', 'Dept No.', 'Salary', and 'Manager'. To the right of these fields are 'Cancel' and 'Exit' buttons. At the bottom, there is a row of five buttons: 'Save', 'New', 'Delete', 'Previous', and 'Next'.

Form frmEmp

The 'Department Data' form is a data entry window with a dotted background. It features three input fields: 'Dept No.', 'Dept Name', and 'Location'. To the right of these fields are 'Previous', 'Next', and 'Exit' buttons. At the bottom, there is a row of four buttons: 'Save', 'New', 'Delete', and 'Cancel'.

Form frmDep

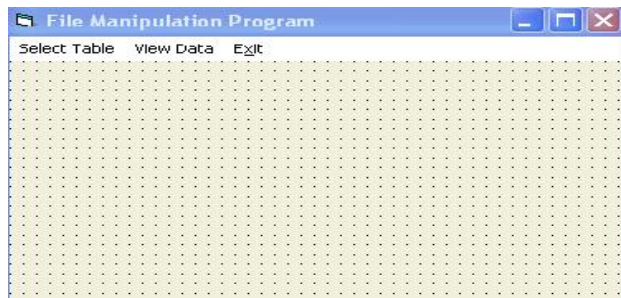
The 'Complete Dept Table' form displays a table with 10 rows and 5 columns. The table is currently empty. An 'Exit' button is located at the bottom right of the form.

Form frmViewEmp

The 'Complete Dept Table' form displays a table with 10 rows and 5 columns. The table is currently empty. An 'Exit' button is located at the bottom right of the form.

Form frmViewDep

NOTES



Form frmMain

1. Start VB and create a new project
2. Save the form and project as Employee.vbp
3. There are five forms in this project. This project could also be developed as an MDI application, but we have developed it as an SDI application.
4. In the first form, set the caption of the form as 'File Manipulation Program' and create the menus as shown above.
5. Add component *Microsoft Hierarchical FlexGrid Control 6.0* by following the command **Project à Component** (or Press **Ctrl + T**) and then selecting this component.
6. Now add four forms, draw the controls and place them according to the screenshot shown above.
7. Name the controls as you like. For both the HFlexGrids, set the **FixedRows** as 1 and **FixedCols** as 0.
8. Add reference to *Microsoft ActiveX Library 2.x* by following the command **Project à Reference** and then selecting the desired library option.
9. Add a standard module to your project by following the command **Project à Add Module**.

Declare ADO Objects

10. Now type the following code in the standard module to declare the required ADODB objects. We declared them as **Public** so that these are available everywhere in the project.

```
Public adoconn As ADODB.Connection    'the Connection object
Public adorsemp As ADODB.Recordset    'Recordset object for
emp table
Public adorsdep As ADODB.Recordset    'Recordset object for
dept table
```


Define Click Events for All Menu Options

11. Open the code window of frmMain and type the following code in it to define the click events of all menu options. If the user clicks on **Select Table à Emp**, then form frmDept should get displayed; on clicking **Select Table à Dept** option, form frmDept should get displayed. When the user clicks on **View Data à Employee Table** option, form frmViewEmp should get displayed and on clicking **View Data à Department Table** option, form frmViewDept should get displayed. On clicking at the Exit menu option, the application should end.

```
Private Sub mnuexit_Click()  
End  
End Sub  
  
Private Sub mnuoptdept_Click()  
frmdept.Show  
End Sub  
  
Private Sub mnuoptdepttable_Click()  
frmviewdept.Show  
End Sub  
  
Private Sub mnuoptemp_Click()  
frmemp.Show  
End Sub  
  
Private Sub mnuoptemptable_Click()  
frmviewemp.Show  
End Sub
```

Create the Form Load Event for Form FrmEmp

12. Double-click on form frmEmp and type the following in its Load event procedure :

```
Private Sub Form_Load()  
MsgBox " Connecting Access Database....."  
Set adoconn = New ADODB.Connection  
adoconn.ConnectionString = "Provider = MSDAORA; user id  
= scott; password = tiger; "  
  
adoconn.Open  
  
Set adorsemp = New ADODB.Recordset  
adorsemp.CursorType = adOpenDynamic  
adorsemp.LockType = adLockOptimistic  
adorsemp.Open "emp", adoconn, , , adCmdTable  
  
Set Text1.DataSource = adorsemp  
Text1.DataField = "empno"
```

NOTES

NOTES

```
Set Text2.DataSource = adorsemp
Text2.DataField = "job"

Set Text3.DataSource = adorsemp
Text3.DataField = "salary"

Set Text4.DataSource = adorsemp
Text4.DataField = "ename"

Set Text5.DataSource = adorsemp
Text5.DataField = "deptno"

Set Text6.DataSource = adorsemp
Text6.DataField = "manager"

End Sub
```

Create the form Load event for form frmDept

13. Double-click on form frmEmp and type the following in its Load event procedure :

```
Private Sub Form_Load()
MsgBox " Connecting Access Database....."
Set adoconn = New ADODB.Connection

adoconn.ConnectionString = "Provider = MSDAORA; user id
= scott; password = tiger; "
adoconn.Open

Set adorsdep = New ADODB.Recordset
adorsdep.CursorType = adOpenDynamic
adorsdep.LockType = adLockOptimistic
adorsdep.Open "dept", adoconn, , , adCmdTable

Set Text1.DataSource = adorsdep
Text1.DataField = "deptno"

Set Text2.DataSource = adorsemp
Text2.DataField = "deptname"

Set Text3.DataSource = adorsemp
Text3.DataField = "location"

End Sub
```

Create Click events for the command button in forms frmEmp and frmDept

14. The command button in both the forms carries out identical tasks. The only difference is that in form frmEmp, the commands button refer to recordset, namely adordep. Given below are the codes for click events of all buttons in the forms frmEmp and frmDept.

Click Events of Button in Form frmEmp

Connecting to
the Database

```
Private Sub Command1_Click()  
Set adorsemp = Nothing  
Set adorsconn = Nothing  
Me.Hide  
End Sub  
  
Private Sub Command2_Click()  
If (MsgBox("are you sure you want to save the record?",  
vbYesNo) = vbYes) Then  
adorsemp.Update  
MsgBox " Record Saved!!"  
End If  
End Sub  
  
Private Sub Command3_Click()  
adorsemp.AddNew  
  
End Sub  
  
Private Sub Command4_Click()  
If (MsgBox("are you sure you want to delete this record?",  
vbYesNo) = vbYes) Then  
adorsemp.Delete  
adorsemp.Update  
MsgBox ("Record successfully deleted in table")  
End If  
  
End Sub  
  
Private Sub Command5_Click()  
adorsemp.CancelUpdate  
End Sub  
  
Private Sub Command6_Click()  
If (adorsemp.BOF = True) Then  
MsgBox "beginning of File"  
adorsemp.MoveLast  
Else  
adorsemp.MovePrevious  
End If  
  
End Sub  
  
Private Sub Command7_Click()  
If (adorsemp.EOF = True) Then  
MsgBox "End of File"  
adorsemp.MoveFirst  
Else  
adorsemp.MoveNext  
End If  
  
End Sub
```

NOTES

Click Events of Button in Form frmDept

NOTES

```
Private Sub Command3_Click()  
Me.Hide  
  
End Sub
```

The remaining click events are the same as frmEmp

15. Create events of form frmViewEmp

```
Private Sub Command1_Click()  
Set adorsemp = Nothing  
Set adoconn = Nothing  
Me.Hide  
End Sub  
  
Private Sub Form_Load()  
Set adoconn = New ADODB.Connection  
    adoconn.ConnectionString = "Provider = MSDAORA; user  
id = scott; password = tiger; "  
adoconn.Open  
Set adorsemp = New ADODB.Recordset  
adorsemp.CursorType = adOpenDynamic  
adorsemp.LockType = adLockOptimistic  
adorsemp.Open "emp", adoconn, , , adCmdTable  
Set MSHFlexGrid1.DataSource = adorsemp  
  
End Sub
```

16. Create events of form frmViewDept

```
Private Sub Command1_Click()  
Set adorsdep = Nothing  
Set adoconn = Nothing  
Me.Hide  
End Sub  
  
Private Sub Form_Load()  
Set adoconn = New ADODB.Connection  
    adoconn.ConnectionString = "Provider = MSDAORA; user  
id = scott;          password = tiger; "  
adoconn.Open  
  
Set adorsdep = New ADODB.Recordset  
adorsdep.CursorType = adOpenDynamic  
adorsdep.LockType = adLockOptimistic  
adorsdep.Open "dept", adoconn, , , adCmdTable  
Set MSHFlexGrid1.DataSource = adorsdep  
  
End Sub
```

17. Save your project and run it.

Check Your Progress

8. What is a query?
9. How will you add a record to a `RecordSet`?
10. Define arguments of `AddNew` method.
11. Explain about the argument which are used in `update` method.
12. What is the syntax for closing the connection?

NOTES

14.7 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Microsoft's ActiveX Data Objects (ADO) is the new data access technology developed by Microsoft. It comprises of a set of Component Object Model (COM) objects for accessing data sources. As a significant part of MDAC (Microsoft Data Access Components), ADO provides a middleware layer between programming languages and OLE DB (Object Linking and Embedding, DataBase), a means of accessing data stores, whether databases or not, in a uniform manner.
2. In order to achieve our objective of accessing a data source, extracting a set of records from it and manipulating or editing the `RecordSet` and finally updating the data source, we have to follow the steps given below:
 - Make a connection to a data source.
 - Create a command to specify the records to be extracted.
 - Execute the command.
 - Navigate and edit the data in the `RecordSet` (We assume that the data is returned as a `RecordSet`).
 - Update the data source with changes made to the data in the `RecordSet`.
3. Full form of ODBC is Open DataBase Connectivity.
4. ODBC is defined as "A standard protocol for database servers. ODBC has drivers for various databases that enable the applications to connect to the databases and access their data." The condition is that these databases must have Structure Query Language (SQL) as the standard for data access.
5. A data provider is a control or object or mechanism that provides data for use by connecting to a source of data.
6. OLE DB is the underlying system service that a programmer using ADO actually uses. OLE DB is a set of interfaces that provides applications with uniform access to data stored in diverse information sources or data stores. OLE DB is suitable for relational and non-relational data sources.

NOTES

7. You can create VB applications using ActiveX Data Object Data Control (ADO DC) in following two ways:
 - Employing ADO DC interactively using VB connection Wizard Data Access Using ADO
 - Writing code, i.e., programming ADO (ADO DB)
8. A query is a way of requesting information from the database. A database query can be either a select query or an action query. A select query is a query for retrieving data, while an action query requests additional actions to be performed on the data, like deletion, insertion, and updating. Visual Basic supports SQL (Structured Query Language) to declare and run required queries.
9. To add a new record to an ADO `RecordSet`, the `AddNew` method is used.
10. The arguments for `AddNew` method are as follows:
 - Fields—A single name or an array of names or ordinal positions of the fields in the new record.
 - Values—A single value or an array of values for the fields in the new record. If Fields is an array, Values must also be an array with the same number of members.
11. `Recordset`, `Update Fields`, `Values` Here are the arguments for update method:
 - Fields—A single name or an array of names or ordinal positions of the fields in the new record.
 - Values—A single value or an array of values for the fields in the new record.
12. Following is the syntax for closing the connection:

```
rs.close
set rs=nothing
cn.close
set cn=nothing
```

14.8 SUMMARY

- Microsoft's ActiveX Data Objects (ADO) is the new data access technology developed by Microsoft. It comprises of a set of Component Object Model (COM) objects for accessing data sources.
- As a significant part of MDAC (Microsoft Data Access Components), ADO provides a middleware layer between programming languages and

OLE DB (Object Linking and Embedding, DataBase), a means of accessing data stores, whether databases or not, in a uniform manner.

- Microsoft introduced ADO in October 1996, positioning the software as a successor to Microsoft's earlier object layers for accessing data sources, including RDO (Remote Data Objects) and DAO (Data Access Objects).
- ADO is supported in any development language that supports binding to binary COM interfaces. These languages include ASP (Active Server Pages), Delphi, PowerBuilder, and Visual Basic for Applications (VBA).
- ADO.NET has replaced ADO as the primary mode for targeting Windows application development. ADO.NET follows the same design pattern as ADO, enabling an ADO developer an easy path forward when moving to the .NET framework.
- Open DataBase Connectivity (ODBC) is defined as "A standard protocol for database servers. ODBC has drivers for various databases that enable the applications to connect to the databases and access their data."
- The condition is that these databases must have Structure Query Language (SQL) as the standard for data access.
- A query is a way of requesting information from the database. A database query can be either a select query or an action query. A select query is a query for retrieving data, while an action query requests additional actions to be performed on the data, like deletion, insertion, and updating. Visual Basic supports SQL (Structured Query Language) to declare and run required queries.
- Dynamic SQL is a programming technique that enables to build SQL statements dynamically at runtime. More general purpose and flexible applications can be created using the Dynamic SQL because the full text of a SQL statement may be unknown at compilation.
- Dynamic SQL helps in creating a procedure that operates on a table whose name may not be known till runtime.
- SQL Server provides different methods for running a dynamically built SQL statement.
- The Dynamic SQL query can be written with parameters. The simple method is, within the declaration you can pass parameters into your WHERE clause of your SQL statement.
- Parameterized queries are basically the same as an ordinary query, but they allow you to make use of parameters inside your queries. A parameter is additional information you can provide to an SQL query. This information usually works in conjunction with a condition inside the SQL query.

NOTES

NOTES

- Action queries are queries that can make changes to many records at once. They are used to delete records, to update records, i.e., to change values in the records, to create new tables, to delete tables and to define queries that accept a user defined parameter.
- To provide access to ADO `RecordSet` and `Record` objects from ADO.NET, the OLE DB .NET Data Provider overloads the `Fill` method of the `OleDbDataAdapter` to accept an ADO `Recordset` object or an ADO `Record` object.
- Filling a `DataSet` with the contents of an ADO object is a one-way operation. That is, data can be imported from the ADO `Recordset` or `Record` into the `DataSet`, but any updates to the data must be handled explicitly by either ADO.NET or ADO.
- A data provider is a control or object or mechanism that provides data for use by connecting to a source of data.
- The data provider makes data connectivity much easier by hiding most of the implementation of data storage.
- OLE DB (Object Linking and Embedding, DataBase) is the underlying system service that a programmer using ADO actually uses.
- OLE DB is a set of interfaces that provides applications with uniform access to data stored in diverse information sources or data stores.
- OLE DB is suitable for relational and non-relational data sources. That is, with OLE DB, you can access all types of databases in the same manner.
- We can access Microsoft Access database using ActiveX Data Object Data Control (ADO DC).
- Visual Basic (VB) wizard will setup a connection to the Access database.
- The Source contains a Structure Query Language (SQL) command, such as a `SELECT` statement.
- The Source string contains the name of a table to be retrieved. (ADO itself creates a SQL query.) Using this option is not recommended because it can pull in too many records.
- The Source string contains the name of a stored procedure (not applicable to Oracle).
- This constant indicates that the type of command in the Source argument is not known. Using this option is inefficient.
- When you were using ADO, programming was still being done, but not by you; it was being done by the Visual Basic (VB) connection wizard. That is, implicit programming was occurring.

- To add a new record to an ADO recordset, you use the `AddNew` method.
- After you have updated the fields of the current record, you save that record to the database with the `Update` method.
- A single name or an array of names or ordinal positions of the fields in the new record.
- A single value or an array of values for the fields in the new record. If `Fields` is an array, `Values` must also be an array with the same number of members.
- The order of field names must match the order of field values in each array.
- Using the **Close** method to close a **Connection** object also closes any active **RecordSet** objects associated with the connection.
- A **Command** object associated with the **Connection** object you are closing will persist, but it will no longer be associated with a **Connection** object; that is, its `ActiveConnection` property will be set to **Nothing**. Also, the **Command** object's `Parameters` collection will be cleared of any provider-defined parameters.
- Closing a **Connection** object while there are open **RecordSet** objects on the connection rolls back any pending changes in all of the **RecordSet** objects.
- Explicitly closing a **Connection** object (calling the **Close** method) while a transaction is in progress generates an error. If a **Connection** object falls out of scope while a transaction is in progress, ADO automatically rolls back the transaction.

NOTES

14.9 KEY WORDS

Microsoft's ActiveX Data Objects (ADO): Microsoft's ActiveX Data Objects (ADO) is the new data access technology developed by Microsoft. It comprises of a set of Component Object Model (COM) objects for accessing data sources.

Open DataBase Connectivity (ODBC): Open DataBase Connectivity (ODBC) is defined as, 'A standard protocol for database servers. ODBC has drivers for various databases that enable the applications to connect to the databases and access their data.'

Query: A query is a way of requesting information from the database. A database query can be either a select query or an action query. A select query is a query for retrieving data, while an action query requests additional actions to be performed on the data, like deletion, insertion, and updating.

NOTES

Dynamic SQL query: Dynamic SQL query can be written with parameters. The simple method is, within the declaration you can pass parameters into your WHERE clause of your SQL statement.

Parameterized queries: Parameterized queries are basically the same as an ordinary query, but they allow you to make use of parameters inside your queries. A parameter is additional information you can provide to an SQL query.

Object Linking and Embedding (OLE): OLE DB is the underlying system service that a programmer using ADO actually uses. OLE DB is a set of interfaces that provides applications with uniform access to data stored in diverse information sources or data stores. OLE DB is suitable for relational and non-relational data sources.

Data provider: A data provider is a control or object or mechanism that provides data for use by connecting to a source of data.

RecordSet: A RecordSet is a data structure that consists of a group of database records, and can either come from a base table or as the result of a query to the table.

14.10 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What is Microsoft's ActiveX Data Objects (ADO)?
2. Define the term Open DataBase Connectivity (ODBC).
3. Write the sequence of steps you need to perform for retrieving data from an already created data source.
4. What are queries?
5. How the query is created dynamically?
6. Differentiate between parameterized query and action query.
7. State about the data source.
8. Explain about the RecordSet.
9. What is the open method for a RecordSet?
10. Write down the syntax for closing the database connection.

Long-Answer Questions

1. Briefly discuss about the significance and characteristic properties of Microsoft's ActiveX Data Objects (ADO).

2. Explain the concept and methodology used for Open DataBase Connectivity (ODBC).
3. Discuss briefly about database connectivity and the process involved for accessing data, retrieving data and updating data from an already created data source in ADO.
4. Briefly discuss the concept and significance of data source giving appropriate examples.
5. Elaborate on the Open DataBase Connectivity (ODBC) data source administrator.
6. Discuss briefly about creating a query dynamically, using a parameterized query and action queries giving appropriate example codes.
7. Discuss briefly about the open method for an ADO connection, giving appropriate code in Visual Basic.
8. Describe the following methods giving appropriate programing codes:
 - AddNew
 - Update
9. Briefly discuss the concept of adding and editing recodes in the database source giving examples program codes.
10. Explain the steps and commands required to close a database connection.

NOTES

14.11 FURTHER READINGS

- Cornell, Gary. 1998. *Visual Basic 6 from the Ground Up*. New Delhi: Tata McGraw-Hill.
- Warner, Scott L. 1998. *Teach Yourself Visual Basic 6*. New Delhi: Tata McGraw-Hill.
- Jerke, Noel. 1999. *Visual Basic 6 - The Complete Reference*. New York: McGraw-Hill.
- Smith, Eric A., Valor Whisler and Hank Marquis. 1998. *Visual Basic 6 Programming Bible*. New Jersey: John Wiley & Sons, Inc.
- Azam, M. 2001. *Programming with Visual Basic*. New Delhi: Vikas Publishing House Pvt. Ltd.
- Manchanda, Mahesh. 2009. *Visual Programming*. New Delhi: Vikas Publishing House Pvt. Ltd.
- Balena, Francesco. 1999. *Programming Microsoft Visual Basic 6.0*. Bangalore: WP Publishers and Distributors (P) Ltd.

NOTES

Petroutsos, Evangelos. 1998. Mastering Visual Basic 6, 1st Edition. New Delhi: BPB Publications.

Deitel, Harvey M., Paul J. Deitel and T. Tem R. Nieto. 1999. Visual Basic 6: How to Program. New Jersey: Prentice-Hall.

Norton, Peter. 1998. Peter Norton's Complete Guide to Visual Basic 6. New Delhi: Techmedia.

Reselman, Bob and Richard A. Peasley. 1998. Using Visual Basic 6. New Jersey: Pearson Education (Que Publishing).

Donald, Bob and Oancea Gabriel. 1999. Visual Basic 6 from Scratch. New Delhi: Prentice-Hall of India.