

*Zero to Mastery In*  
**VISUAL BASIC**



*Zero to Mastery In*  
**Visual Basic**

**Dr. RK Jain**

---

• Shadab Saifi (*Illustrator*) • Ayaz Uddin (*Editor*)

---



An ISO 9001:2008 Certified Company

**Vayu Education of India**

2/25, Ansari Road, Darya Ganj, New Delhi-110 002



Copyright © Vayu Education of India

**First Edition: 2022**

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of the copyright owners.

## **DISCLAIMER**

Errors, if any, are purely unintentional and readers are requested to communicate such errors to the publisher to avoid discrepancies in future.

***Published by:***

**AN ISO 9001:2008 CERTIFIED COMPANY**

**VAYU EDUCATION OF INDIA**

2/25, ANSARI ROAD, DARYA GANJ, NEW DELHI-110 002

PH.: 011-41564440, MOB. 09910115201











# CONTENTS











---

<b>1. INTRODUCTION TO VISUAL BASIC .....</b>	<b>1-9</b>
1.1 INTRODUCTION .....	1
1.2 FEATURES OF VB .....	1
1.3 VISUAL BASIC CONCEPT .....	2
1.3.1 Working of Windows, Events and Messages .....	2
1.3.2 Event Driven Model .....	2
1.3.3 Interactive Development .....	2
1.4 VISUAL BASIC ENVIRONMENT .....	3
1.4.1 VB Integrated Development Environment .....	3
1.4.2 VB Application Mode .....	7
1.5 PROPERTIES, METHODS AND EVENTS .....	7
1.6 DEBUGGING .....	7
1.7 DIFFERENCE BETWEEN .EXE AND .DLL FILE .....	8
Let Us Revise .....	9
<b>2. VB PROGRAMMING BASICS .....</b>	<b>10-25</b>
2.1 INTRODUCTION .....	10
2.2 KEYWORD .....	10
2.3 DATA TYPES .....	10
2.3.1 Numeric Data Type .....	11
2.3.2 Non Numeric Data Types .....	12
2.4 VARIABLES .....	13
2.4.1 Variable Naming Conventions .....	13
2.4.2 Declaring Variables .....	14
2.4.3 Type Declaration Characters .....	14
2.4.4 Fixed Length Vs. Variable Length String .....	15
2.4.5 Assigning Values To Variables .....	16
2.4.6 Variable Default Values .....	16
2.4.7 Implicit And Explicit Variable Declaration .....	17
2.4.8 Variable Scope And Lifetime .....	17
2.4.9 Static Variable .....	19
2.5 LITERALS .....	19
2.6 OPERATORS IN VB .....	20

2.6.1	Arithmetic Operator .....	20
2.6.2	Concatenation Operator .....	20
2.6.3	Comparison Operator .....	21
2.6.4	Logical Operator .....	21
2.6.5	Operators Precedence .....	22
2.7	SOME USEFUL FUNCTION .....	22
2.7.1	Rnd Function (Random Number Generator) .....	23
2.7.2	Format Function .....	23
2.7.3	Val Function .....	24
	Let Us Revise .....	25

### 3. WORKING WITH CONTROLS ..... 26-67

3.1	INTRODUCTION .....	26
3.2	INTRINSIC AND CONTAINER CONTROL .....	27
3.3	WORKING WITH CONTROLS .....	27
3.3.1	To Draw a Control On a Form .....	27
3.3.2	Object Naming Conventions .....	27
3.3.3	Setting Properties .....	28
3.4	CREATE EVENT PROCEDURE .....	29
3.5	MANIPULATING FORMS .....	30
3.5.1	Events of Form .....	30
3.5.2	Form Properties .....	31
3.6	BASIC CONTROLS .....	32
3.6.1	Pointer  .....	32
3.6.2	Label  .....	32
3.6.3	Text Box  .....	33
3.6.4	Command Button  .....	33
3.6.5	Check Box  .....	36
3.6.6	Option Button  .....	36
3.6.7	Frame Control  .....	37
3.6.8	ListBox Control  .....	37
3.6.9	ComboBox Control  .....	40
3.6.10	Image Box  .....	41

3.6.11	Picture Box 	43
3.6.12	Hscroll Bar  & Vscroll Bar 	45
3.6.13	Shape  & Line 	47
3.6.14	Timer 	49
3.6.15	File System Control : DriveList Box  , DirList Box  & FileList Box 	51
3.7	CONTROLARRAYS	53
3.8	SOME USEFUL EVENTS	54
3.9	ACTIVEX CONTROL	55
3.9.1	Adding and Removing ActiveX Controls	56
3.9.2	Common Dialog Boxes	57
3.9.3	Microsoft Windows Common Controls	60
3.10	OBJECT LINKING AND EMBEDDING (OLE  )	63
3.11	DIALOG BOX	63
3.11.1	Input Box	63
3.11.2	Message Box	64
	Let Us Revise	66
<b>4.</b>	<b>CONTROL STRUCTURE</b>	<b>68-98</b>
4.1	INTRODUCTION	68
4.2	CONTROL FLOW	68
4.3	DECISION STRUCTURES	69
4.3.1	If...Then Statement	69
4.3.2	Select...Case Statement	78
4.4	LOOPING STRUCTURE	84
4.4.1	For...Next	85
4.4.2	Do Loop Structure	87
4.4.3	While...Wend	91
4.4.4	Problems with loop	91
4.5	ARRAYS	91
4.5.1	Dimension of an Array	92
4.5.2	Declaring Arrays	92
4.5.3	Static and Dynamic Arrays	96
4.5.4	Arrays within UDTs	96
4.5.5	Array within Another Array	97
	Let Us Revise	97
<b>5.</b>	<b>PROCEDURES, FUNCTIONS AND MODULES</b>	<b>99-134</b>
5.1	INTRODUCTION	99
5.2	PROCEDURES	99

5.3	SUB PROCEDURES (SUB-ROUTINES)	100
5.3.1	<i>Declaring Syntax</i>	100
5.3.2	<i>Add Procedure Menu Option</i>	101
5.3.3	<i>Calling Sub-Procedures</i>	101
5.4	FUNCTION PROCEDURE	102
5.4.1	<i>Declaring Syntax</i>	103
5.4.2	<i>Calling Function</i>	103
5.5	PASSING PARAMETERS TO PROCEDURES	104
5.5.1	<i>Call by Value</i>	105
5.5.2	<i>Call By Reference</i>	105
5.6	PROPERTY PROCEDURE	109
5.7	CODE MODULE	109
5.7.1	<i>Form Module</i>	110
5.7.2	<i>Standard Module</i>	111
5.7.3	<i>Class Module</i>	117
5.8	LIBRARY FUNCTION	118
5.8.1	<i>String Function</i>	118
5.8.2	<i>Numeric Function</i>	128
5.8.3	<i>Date and Time Function</i>	130
	<i>Let Us Revise</i>	134
<b>6.</b>	<b>VB INTERFACE STYLE</b>	<b>135-149</b>
6.1	INTRODUCTION	135
6.2	INTERFACE STYLE	135
6.2.1	<i>Single Document Interface (SDI)</i>	135
6.2.2	<i>Multiple Document Interface (MDI)</i>	136
6.2.3	<i>Explorer Style Interface</i>	138
6.3	CREATING MENUS	138
6.3.1	<i>Menu Basics</i>	138
6.3.2	<i>Menu Title And Naming Guidelines</i>	139
6.3.3	<i>Access Keys And Shortcut Keys</i>	139
6.4	DESIGNING MENUS	140
6.4.1	<i>Steps To Create Menu</i>	141
6.4.2	<i>Menu Control Array</i>	148
6.5	POPUP MENUS	148
6.5.1	<i>Creating Popup Menu</i>	148
	<i>Let Us Revise</i>	149
<b>7.</b>	<b>ERROR HANDLING AND FILE HANDLING</b>	<b>150-160</b>
7.1	INTRODUCTION	150
7.2	TYPES OF ERROR	150
7.2.1	<i>Compile – Time Error</i>	150
7.2.2	<i>Run-Time Error</i>	151
7.2.3	<i>Logical Errors</i>	152



7.3	HANDLING ERRORS .....	152
7.4	TRAP THE ERROR .....	152
7.4.1	<i>On Error GoTo 0</i> .....	152
7.4.2	<i>On Error Resume Next</i> .....	153
7.4.3	<i>On Error GoTo Line</i> .....	153
7.5	HANDLE THE ERROR .....	154
7.5.1	<i>Leaving Error Handlers</i> .....	156
7.5.2	<i>Error Object</i> .....	156
7.6	FILE HANDLING .....	157
	<i>Let Us Revise</i> .....	159

## 8. DATABASE CONNECTIVITY AND VISUAL

	<b>DATABASE TOOLS .....</b>	<b>161-209</b>
8.1	INTRODUCTION .....	161
8.2	DB CONCEPTS .....	161
8.3	DATAACCESS MECHANISM .....	162
8.3.1	<i>Difference Between DAO, RDO and ADO</i> .....	163
8.4	DB ENGINE .....	163
8.4.1	<i>Microsoft Jet Database Engine</i> .....	163
8.4.2	<i>Open DB Connectivity (ODBC)</i> .....	164
8.4.3	<i>Object Link Embedding (OLE DB)</i> .....	165
8.5	VB DATA CONTROL .....	165
8.5.1	<i>Data Control</i> .....	165
8.5.2	<i>Data Bound Controls</i> .....	166
8.6	COMPANY DATABASE .....	166
8.7	DB CONNECTION BY DATA CONTROL .....	168
8.8	DB CONNECTIVITY BY ADO DC .....	174
8.9	DB CONNECTIVITY BY ADODB .....	187
8.9.1	<i>Objects Of ADO</i> .....	187
8.9.2	<i>Locktype</i> .....	188
8.9.3	<i>Steps To Access DB Through ADODB</i> .....	188
8.10	VISUAL DATABASE TOOLS .....	197
8.10.1	<i>Data Environment Designer</i> .....	197
8.10.2	<i>Data Report</i> .....	197
8.10.3	<i>Crystal Report</i> .....	198
8.11	CREATING DATA REPORT .....	198
8.11.1	<i>Steps To Create Data Environment For Emp_Dept</i> .....	198
8.11.2	<i>Steps To Create Data Report Emp_Dept</i> .....	203
8.12	TRANSACTIONS AND CONCURRENCY CONTROL .....	207
8.12.1	<i>Transaction</i> .....	208
8.12.2	<i>Concurrency Control</i> .....	208
	<i>Let Us Revise</i> .....	208

**9. HELP WRITING AND SOME OTHER FEATURES ..... 210-227**

9.1 INTRODUCTION ..... 210

9.2 HELP FILE ..... 210

    9.2.1 *Help Modes* ..... 210

    9.2.2 *Requirements* ..... 210

    9.2.3 *HTML Help File* ..... 211

    9.2.4 *Steps To Create Html Help File* ..... 211

9.3 CONTEXT-SENSITIVE HELP ..... 213

    9.3.1 *About The Helpprovider Class* ..... 214

    9.3.2 *Steps To Create Context-Sensitive Help* ..... 214

9.4 COMPONENT OBJECT MODEL (COM) ..... 216

9.5 DISTRIBUTED COMPONENT OBJECT MODEL (DCOM) ..... 216

9.6 WINDOWS APPLICATION PROGRAMMING INTERFACE (API) ..... 217

    9.6.1 *What Is The Windows API?* ..... 217

    9.6.2 *Exploring The API Function List* ..... 217

    9.6.3 *Calling API* ..... 218

9.7 MESSAGING APPLICATION PROGRAMMING INTERFACE (MAPI) ..... 219

9.8 MICROSOFT TRANSACTION SERVER ..... 221

    9.8.1 *Microsoft Transaction Server Run-Time Environment* ..... 221

    9.8.2 *Microsoft Transaction Server Explorer* ..... 222

    9.8.3 *Microsoft Transaction Server APIS* ..... 222

    9.8.4 *Microsoft Transaction Server Sample Applications* ..... 223

9.9 VISUAL SOURCESAFE ..... 223

    9.9.1 *Limitation* ..... 225

9.10 MICROSOFT'S VBSCRIPT ..... 225

    9.10.1 *Adding VbScript To Web Pages* ..... 225

    9.10.2 *Working With Variables* ..... 226

    9.10.3 *Objects And VbScript* ..... 226

    9.10.4 *Linking VbScript With Objects* ..... 227

    9.10.5 *Using VbScript With Forms* ..... 227

**10. ADVANCE FEATURES OF VB 2010 ..... 228-242**

10.1 INTRODUCTION ..... 228

10.2 VB COMPILER RUNTIME SWITCH ..... 228

10.3 AUTO-IMPLEMENTED PROPERTY ..... 229

    10.3.1 *Backing Field* ..... 231

    10.3.2 *Initializing an Auto Implemented Property* ..... 232

    10.3.3 *Property Definitions That Require Standard Syntax* ..... 232

    10.3.4 *Expanding an Auto-Implemented Property* ..... 233

10.4 COLLECTION INITIALIZERS ..... 233

10.5 IMPLICIT LINE CONTINUATION SUPPORT ..... 234

10.6 MULTILINE LAMBDA EX-PRESSIONS AND SUBROUTINES ..... 234

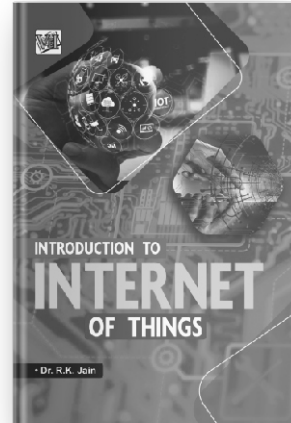
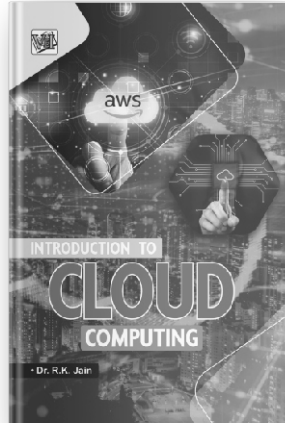
    10.6.1 *To Create a Single-line Lambda Expression Subroutine* ..... 235

10.6.2	<i>To Create a Multiline Lambda Expression Function</i> .....	236
10.6.3	<i>To Create a Multiline Lambda Expression Subroutine</i> .....	236
10.7	TYPE EQUIVALENCE SUPPORT .....	237
10.8	DYNAMIC SUPPORT .....	238
10.9	COVARIANCE AND CONTRAVARIANCE .....	239
10.10	NAVIGATE TO .....	240
10.11	NEW COMMAND-LINE OPTION .....	241
	<i>Let Us Revise</i> .....	241

<b>INDEX</b> .....	<b>243</b>
--------------------	------------

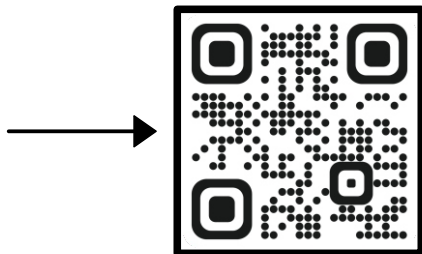
# SPECIAL BONUS!

Want These 3 Bonus Books for free?



Get FREE, unlimited access to these and all of our new books by joining our community!

SCAN w/ your camera TO JOIN!



OR Visit  
[freebie.kartbucket.com](http://freebie.kartbucket.com)

## Chapter-1

# Introduction to Visual Basic

### 1.1 INTRODUCTION

Visual Basic was developed in early 1990s by Microsoft Corporation. It is one of the powerful programming systems that help to develop sophisticated, graphical applications that can be run on Microsoft Windows environment. Visual Basic is actually BASIC language, which is visual in nature. It creates graphical entities rather than writing numerous lines of code to describe the appearance, functioning etc., of the application's interface.

A program can be developed in either modular or procedural programming style or object-oriented programming style. The modular programming style emphasizes on procedures and not on data. A bigger program is divided into smaller complete sub programs, known as module. The object-oriented programming style emphasizes upon objects. An object is an identifiable entity with some characteristic and associated behavior.

Visual Basic supports a new type of programming style i.e Event Driven Programming. The event driven programming style responds to the user events and is driven by the occurrence of user events. Visual Basic is a combination of object oriented and event driven programming, which supports visual effects.

In this chapter we, learn about visual basic feature, its environment and various tools that can be used in Visual Basic. In this chapter and in the entire forthcoming chapter Visual Basic is referred as VB.

### 1.2 FEATURES OF VB

VB programming environment provides all features that are required to develop a graphical user interface as ready to use components. To design user interface, programmer can directly use the components like button, text box, radio button etc instead of writing codes to create and display these components. VB provides many powerful features. Some prominent features are:

- It is successor of BASIC language.
- VB supports event driven programming.
- VB provides a common programming platform across all MS-Office applications.

- VB offers many tools that that provides a quick and easy way to develop applications.
- VB also provides many wizards that can automate tasks or coding.
- VB development environment provides tools for quick editing, testing and debugging.
- VB allows to migrate applications to an ActiveX documents. An ActiveX document enables the application to install and run from a web browser. Thus, even if the programmer does not know web language, he can develop applications that can run on web browser.
- A traditional client/server model has two layers i.e two tier of application – the client application which request for something and the server application which servers that request.

In two tier architecture, there are no middle layers of applications. However VB provides some layers between the client tier and server tier. Thus, VB is allows N-tier architecture.

### **1.3 VISUAL BASIC CONCEPT**

In order to develop an application, there are some key concepts upon which VB is built. VB is a window development language, which supports event driven programming. Also coding is done interactively in VB.

#### **1.3.1 Working of Windows, Events and Messages**

A window can be a rectangular region with its own boundaries. You must aware of several different type of window like document window, dialog window etc. There are many other type of windows. A command button, a text box, a radio button are also called windows.

The window operating system manages all of these windows by assigning a unique id to each window. The system continuously monitors each of these windows for signs of activities and events. Event can occur through user actions such as mouse click or key press etc.

An Event refers to the occurrence of an activity.

Each time an event occurs, it causes a message to be sent to the operating system. The system processes the message and broadcasts it to the other windows. Each window can then take the appropriate action based on its own instructions for dealing with that particular message.

A Message is the information / request sent to the application

#### **1.3.2 Event Driven Model**

In an event driven application, the code doesn't follow a predetermined path rather it executes different code sections in response to events. Events can be triggered by the user's actions, by messages from system or other applications or even from the application itself. The sequence of these events determines the sequence in which the code executes.

#### **1.3.3 Interactive Development**

The traditional application development process can be categorized into three steps- writing, compiling and testing code. VB uses an interactive approach to development that merges these three steps.

VB interprets your code as you entered it, catching and highlighting most syntax or spelling errors. In addition to catching errors, VB also partially compiles the code as it is entered. If the compiler finds an error, it is highlighted in your code. You can fix the error and continue compiling without having to start over.

## 1.4 VISUAL BASIC ENVIRONMENT

To start up VB, you need to follow following steps –

1. Click on **Start** button.
2. Click at **All Programs — Microsoft Visual studio — Microsoft Visual Basic**

This will open **New Project** dialog box where you find many different type of projects. Select the desires Project. Generally, we select **Standard EXE Project**.

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

VB provides many different types of project to create. These projects are listed in table 1.1.

The working environment in VB integrates many different functions such as design, editing, compiling and debugging within a common environment. This working environment of VB is known as *Integrated Development Environment (IDE)*.

### 1.4.1 VB Integrated Development Environment

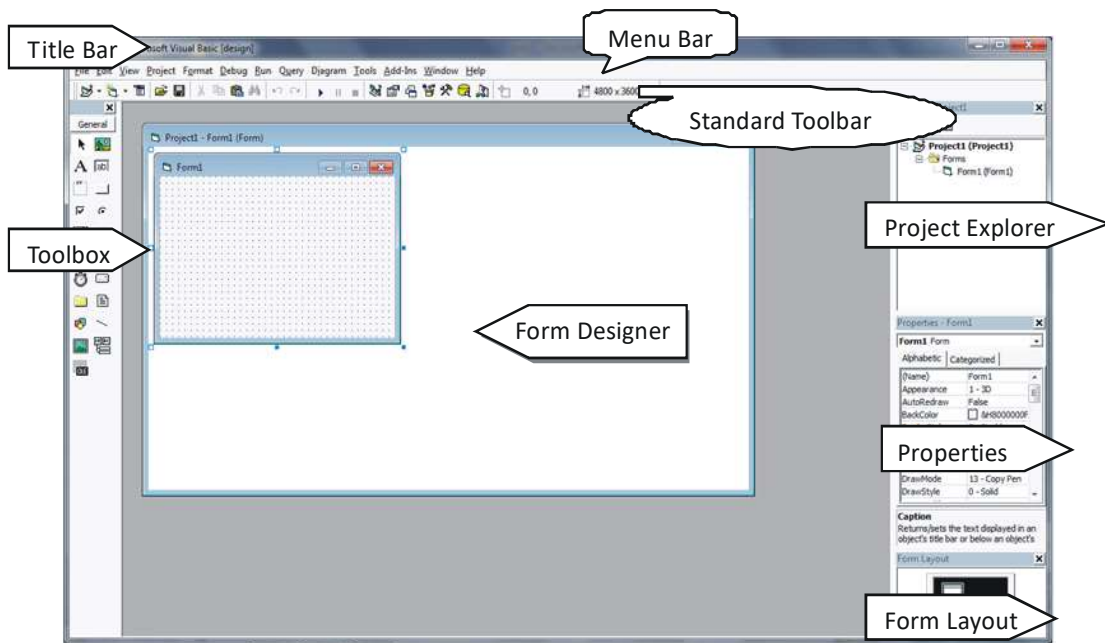
When we open a new project, the VB IDE shows the screen as shown in Fig 1.1. VB IDE consists of following elements:

**Table 1.1**

	<b>Project Type</b>	<b>Description</b>
1.	Standard EXE	For creating a typical 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. It is a basic element of user interface eg. Textbox, check box
5.	VB ApplicationWizard	For setting up the skeleton of a new application
6.	VB Wizard Manager	For building your own wizard. A wizard is a sequence of dialog windows that collect information from the user to carry out a specific task.
7.	Data Project	For creating data project which is combination of standard EXE and various data access controls.
8.	IIS Application	For creating an application that can run on a web server.
9.	Add in	For creating your own Add In .

10.	ActiveX DocumentDLL	For creating ActiveX documents in DLL form.
11.	DHTML Application	For building dynamic HTML pages.
12.	VB EnterpriseEdition Control	It creates a new standard EXE project and loads all the tools of the professional edition of VB.

- 1. Title Bar:** It is top most bar displaying the title of project. By default, VB will give name as Project1, Project2....to your project. It also displays the application mode which we discuss further.
- 2. Menu Bar:** The horizontal menu displayed below the title bar where each option on the menu bar has a drop-down list of items.
- 3. Toolbar:** A toolbar displays icon for the commonly used tasks. The standard toolbar of VB displays icons for the most frequently used commands in VB.
- 4. Form:** Forms are the main part of the project which is used to display various controls (textbox, listbox, buttons, etc) that form the user interface.



**Fig. 1.1**

- 5. Toolbox:** A toolbox is a window that displays a set of tools that may be used to place controls on the form. The buttons on the toolbox is called is termed as *control*.

In chapter – 3 , we discuss each control of toolbox in detail.



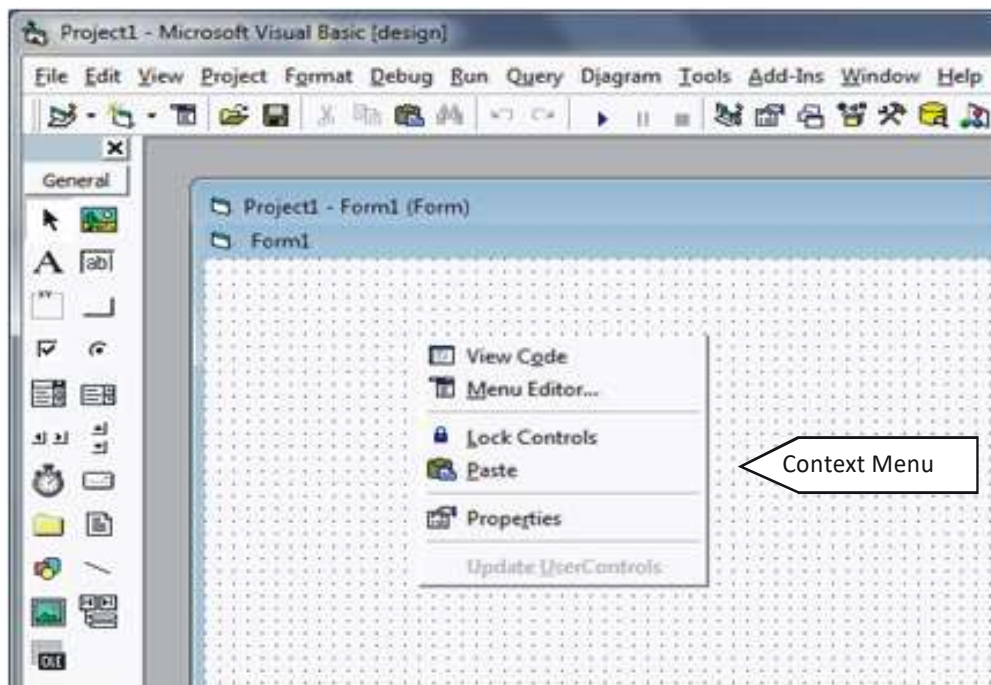
6. **Form Designer:** The form designer window is the form window in design form. In this window different controls are placed and grid of dots helps you to line up controls at the time of design. These dots will disappear at the time of execution.
7. **Project Explorer:** The project explorer window shows the list of forms and modules in a project. A VB project consists of a number of forms, modules and controls that make up an application.

If project explorer is not displayed, then click on view → Project explorer or press ctrl+R.

8. **Properties Window:** A form can consist of many controls on it. Every control and form has some properties associated with it. The properties window lists the properties of selected control. A property is a characteristic of a control such as its size, name, caption, color etc. It displays property of one control at a time.

If properties window is not displayed, then click on view → Properties Window or press F4.

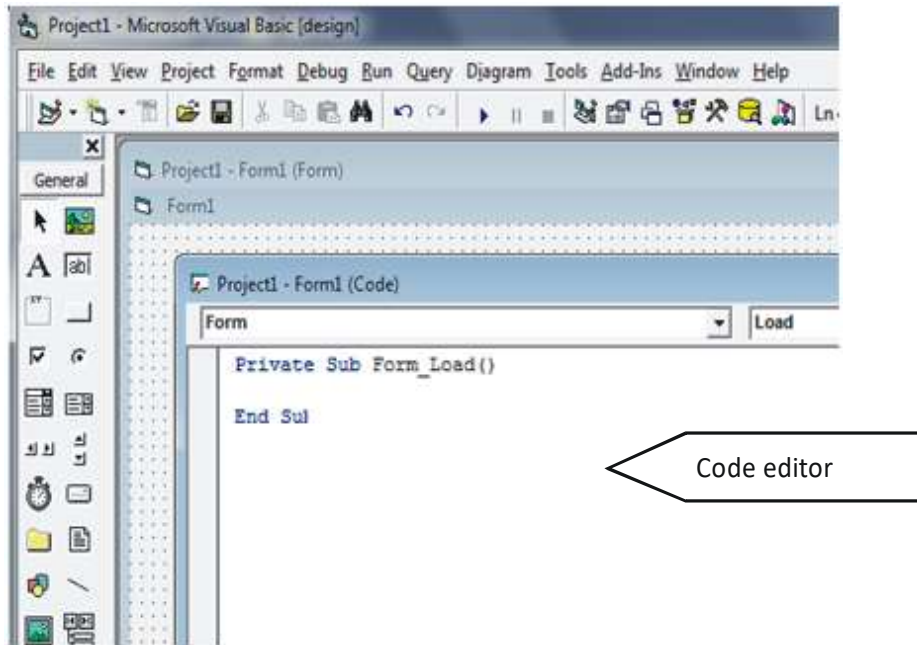
9. **Form Layout:** It shows how big a form is in relation to the screen. It also displays the position of the form where it will be displayed at run time.



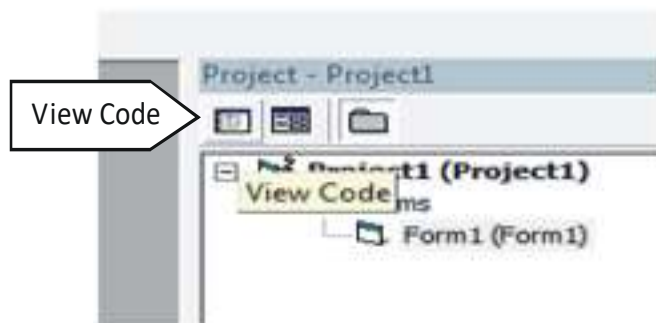
10. **Context menu:** These are the shortcut menus that contain shortcuts to frequently performed actions. To open a context menu, click the right mouse button on the object you are using.

The specific list of shortcuts available from context menu depends on the part of the environment where you click the right mouse button.

**11. Code Editor Window:** It is the window where you write code for your application.



To open code window either double click on the control or click on the view code button of project explorer.



### 1.4.2 VB Application Mode

A VB application works in the following three modes

1. Design Mode – There application is being created or designed.
2. Run Mode – When application is executing.
3. Break / Suspend Mode - When application is in the state of suspension.

## 1.5 PROPERTIES, METHODS AND EVENTS

An object or control has some characteristic attributes such as its name, color, appearance etc. These characteristic are called *properties*.

A *Method* causes an object to do some action and *Event* is the occurrence of an action i.e the activity that happens when an object does something. Methods are block of code that tells the control how to do things. Some common methods are move, drag and setfocus etc.

Table 1.2 shows some common events associated with controls

**Table 1.2**

<b>Events</b>	<b>Occurs When</b>
Change	The user modifies text in a combo box or text box.
Click	The user clicks the primary mouse button on an object.
Dbl Click	The user double clicks the primary mouse button on an object.
Got Focus	An object receives focus.
Key Down	The user presses a keyboard key while an object has focus.
Key Press	The user presses and releases a keyboard key while an object has focus.
Key Up	The user releases a keyboard key while an object has focus.
Lost Focus	An object lost focus.

Although properties, methods and events do different things but they are interrelated. For example – If you move a control with the move method, one or more control's position properties will change as a result. Because the control's size has changed, the resize event occurs.

Code in VB is divided into smaller blocks called procedures. A procedure containing code that is executed when an event occurs is called *Event procedure*.

## 1.6 DEBUGGING

Debugging is the process of locating and fixing or bypassing bugs (errors) in computer program code or hardware device. To *debug* a program or hardware device is to start with a problem, isolate the source of the problem, and then fix it. A user of a program that does not know how to fix

the problem may learn enough about the problem to be able to avoid it until it is permanently fixed.

Debugging is a necessary process in almost any new software or hardware development process, whether a commercial product or an enterprise or personal application program. Debugging tools (called *debuggers*) help identify coding errors at various development stages. VB gives us a lot of opportunities to find bugs before one put their software into the hands of the users. The primary debugging tools in the Visual Basic programming environment are the following three debugging windows:

- The Watch window
- The Locals window
- The Immediate window

**Immediate Window** is a great place for you to modify data or to test the function during development. The Immediate window is used at design time to debug and evaluate expressions, execute statements, print variable values, and so forth. It allows you to enter expressions to be evaluated or executed by the development language during debugging. To display the immediate window, open a project for editing, then choose Windows from the Debug menu and select Immediate.

**Local Window** enables you to see the value of every variable and each member of all the objects which are in current scope. To display the Locals window. From the Debug menu, choose Windows and click Locals. The default context is the function containing the current execution location. You can choose an alternate context to display in the Locals window.

**Watch Window** enables you to monitor the value for a certain state. You might want the program execution to pause on an instruction that sets a certain date. You might want to set watch expression that cause VB to break when a variable changes its value or when an expression's value us True.

## 1.7 DIFFERENCE BETWEEN .EXE AND .DLL FILE

.dll	.exe
<p>Full name of .dll is Dynamic Link Library</p> <p>.dll files requires address space to run</p> <p>.dll can be part of the .exe.</p> <p>.dll files can b reused in the application.</p> <p>.dll can not use by End User.</p> <p>We can not Run the .dll</p> <p>A .dll is a file that can be loaded and executed by programs dynamically.</p> <p>.dll is in-process component, both component and consumer will share same memory</p>	<p>Fullname of .exe is Extensible Execute File</p> <p>.exe files are executed in its own address space</p> <p>.exe can run independently</p> <p>.exe files cannot be reused</p> <p>.exe use by End User like-Client</p> <p>We can Run the .exe</p> <p>When you deployed your application at that time the .dll,.exe files will create</p> <p>.exe is out process component, it will run in its own memory.</p>

### **LET US REVISE**

- ✓ VB is a visual programming environment with BASIC language.
- ✓ VB supports event driven programming and ActiveX
- ✓ A property refers to the characteristic of a control.
- ✓ An event refers to an occurrence of an activity.
- ✓ A method causes an action to do something.
- ✓ A message is the request sent to an application.
- ✓ The working environment of VB is known as Integrated Development Environment.
- ✓ A project in VB is collection of several different files that make up your program.
- ✓ An ActiveX control is a basic element of user interface eg. Textbox , checkbox.
- ✓ A VB application works in three modes – design mode, run mode, break mode.
- ✓ Various windows can be opened through View menu.
- ✓ An event procedure is a procedure containing code that is executed when an event occurs.
- ✓ Process of locating or fixing errors in a program is called debugging.
- ✓ Debugging tools in VB are: Watch window, immediate window and local window.

## Chapter-2

# VB Programming Basics

### 2.1 INTRODUCTION

Through programming you can create many more useful and complex applications. But programming involves manipulation of many values and these values must be stored somewhere.

In this chapter we discuss how values are named, how they can store, what is the lifetime and scope of these values how these values are manipulated and many more.

### 2.2 KEYWORD

Keywords are the words whose meaning has already been explained in VB compiler. The keywords cannot be used as variable names because if we do so we are trying to assign a new meaning to the keyword, which is not allowed by the computer. However you may create variables names which exactly resembles the keyword but not exactly the same. It would be safer not to mix up the variable names and the keywords. These words are also called *Reserved words*.

The word whose meaning is already defined in VB is known as Keyword.

Following is the list of some keywords which is used by VB compiler–

Integer	String	Double	Date
Text	List	Image	Caption
Data	Recordset	If	For

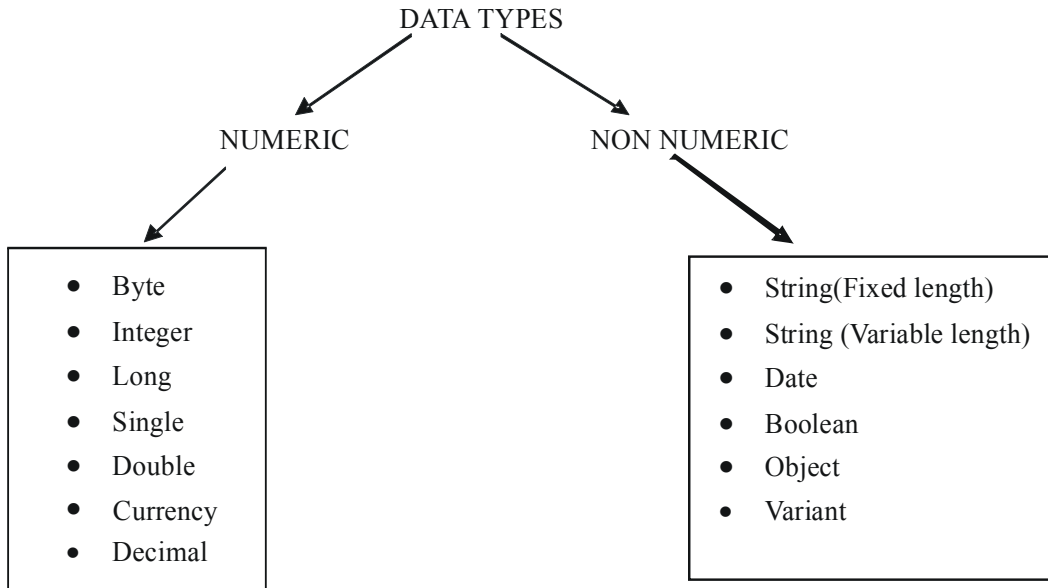
### 2.3 DATA TYPES

Like any other programming language, VB also provides facilities to work with different types of data. These different types of data types of data can be worked with through data types. Each type of data is handled in a different manner. A data type is to identify the type of data and associated operations with it.

Data Type is means to identify the type of data and associated operations with it.

## DATA TYPES

We can divide the data types in two major categories: Numeric and Non numeric data types.



### 2.3.1 Numeric Data Type

Numeric data are the data that consists of number, which can be used in calculations. These numeric data can be represented in different ways as mentioned in Table 2.1

**Table 2.1**

<b>Data Type</b>	<b>Description</b>
Byte	This data type allows positive integer in the range of 0-255.
Integer	This data type holds non fractional values within the range of -32,768 to +32,767.
Long	This data type holds non fractional values that are larger than integer.
Single	It is short for single precision values. It allows up to store 6 digits after decimal.
Double	It is short for double precision values. It can store values up to 14 digits after decimals.
Currency	This data type is used to hold currency data. It allows maximum 15 numbers of digits before decimal point and 4 numbers of digits after decimal point.

These data type have different size in terms of bytes and have different range of values that can be stored in them. This can be summarized in Table 2.2

Table 2.2

Type	Storage	Range Of Values
Byte	1 byte	0 to 255
Integer	2 bytes	-32,768 to +32,767
Long	4 bytes	-2,147,483,648 to +2,147,483,648
Single	4 bytes	-3.402823E+38 to -1.401298E-45 for negative values 1.401298E-45 to 3.402823E+38 for positive values
Double	8 bytes	-1.79769313486232E+308 to -4.94065645841247E-324 for negative values 4.94065645841247E-324 to 1.79769313486232E+308 for positive values
Currency	8 bytes	-922,337,203,685,477.5808 to -922,337,203,685,477.5807

### 2.3.2 Non Numeric Data Types

Non numeric values are those values which cannot participate in calculations like string, true or false. VB provides following different type of non numeric data type –

Table 2.3

Data Type	Description
Boolean	This data type can take either true or false.
Date	This data type holds date and time values.
Object	It is a special data type which is used to hold and refer to objects such as controls and forms.
String	This is used to store textual data. It can store data having digits, alphabets and other characters.
Variant	It can store any type of system defined data type. It is the default data type for values whose data type is not specified.

As numeric data type, these data types also have different size in terms of bytes and have different range of values that can be stored in them. This can be summarized in table 2.4.



Table 2.4

Data Type	Storage	Range
Boolean	1 Byte	True or false
Date	8 Bytes	1 January 100 to 31 December 9999
Object	4 Bytes	Any embedded object
String (Fixed)	1 byte for every character	1 to 65,400 characters
String (Variable)	Length of string + 10 bytes	0 to 2 billion characters
Variant	16 Bytes	Any value as large as double data type

## 2.4 VARIABLES

Sometimes you need to store some values while your program is running. For example – Consider a program handling transaction of a restaurant. To find the total profit of a day, you might need to store previous amount, which is added to new amount time by time. Thus a storage location is required to store the latest amount, which can be changed or varied. Such a storage location is called *Variable*.

Variable is a named storage location whose content can be varied.

A variable is temporary storage space for numbers, text, and objects. Variables are constantly being created and destroyed and will not hold any values after your program have ended.

### 2.4.1 Variable Naming Conventions

VB provides extremely liberal rules for naming variables. All variables names must conform to the following requirements:

- The name must begin with a letter of the alphabet.
- The name can be as long as 255 characters.
- The name must consist only of letters, digits, and the underscore character. (No punctuation marks are allowed).
- Variable names can't be duplicated with the same scope. This means that you can't have two variables of the same name within a procedure. However, you can have two variables with the same name in two different procedures.

Examples of some valid variable name in VB:

Rollno, acc\_number, rule1, data1a

Examples of some invalid variable name in VB:

3Rollno, acc.number, data 1a

### 2.4.2 Declaring Variables

Telling the program about a variable in advance is called declaring variable. A variable has two associated thing with it – a name and a data type. The variable name refers to the value stored in it and the data type tells what type of value can be stored in the variable. A variable can be declared as per following syntax:

#### Syntax:

```
Dim <varname> [As <datatype>] , <varname> [As <datatype>]
```

Where

- Dim is the keyword that tells VB that a variable is declared.
- <varname> is the variable name
- As is another keyword that tells VB the datatype of the variable.
- <datatype> is a legal data type that is defined in Vb
- [ ] brackets means the part is optional.

#### Example

1. **Dim rollno:** There variable name is ‘rollno’. As no data type is defined, thus VB assumes its data type as ‘variant’. But use of variant data type must be discouraged as it takes large memory as compared to other data types.
2. **Dim rollno As Byte:** There variable name is ‘rollno’ and its data type is ‘byte’. That means ‘rollno’ can store any value between its range i.e 0 – 255. This is the proper way of declaring a variable.
3. **Dim rollno, marks As Integer:** Here you want to define ‘rollno’ and ‘marks’ as integer data type. In above syntax only marks carry integer data type whereas data type of ‘rollno’ is variant as no data type is declared with it. Thus to declare ‘rollno’ and ‘marks’ both as integer data type you have to use following syntax :

```
Dim rollno as Integer, marks as Integer
```

### 2.4.3 Type Declaration Characters

An alternative way of specifying data type while declaring variables is the use of some special characters in place of data type. These special characters that specify the data type is called type declaration characters. Table 2.5 lists various type declarations

Symbols:

**Table 2.5**

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

The variables can be declared as follows by using type declaration character:

### Syntax:

```
Dim <varname><type declaration symbol>
```

‘As’ clause is not needed in this syntax. The type declaration symbol is not part of the variable name even though it is typed with variable name.

### Example

**Dim rollno%:** This defines ‘rollno’ as integer data type as % symbol is used for integer.

### 2.4.4 Fixed Length Vs. Variable Length String

A string is a data element which can store some information composed of characters. Number of characters that a string can store is called *string length*. As each character requires one byte of its storage, number of characters inside a string determines its string length in byte. There can be fixed length and variable length string.

#### Fixed Length String

Fixed length string occupies fixed number of bytes for data element they store. The numbers of bytes are determined by the maximum number of characters the string can store. To declare a string of fixed length you need to specify its size as shown below: Dim name As String \* 15

Above, you declare a string variable ‘name’ which can store maximum 15 characters. If you assign more characters than its length, then all extra characters are removed without any warning.

#### Variable Length String

Variable length string has varied string lengths which is determined separately for every data element inside a string. The number of characters in the data element becomes its string length. To declare a varied length string, you declare it as you declare other variables. Dim name As String

Above, you declare a varied length string. Its length changes as value of 'name' varies. If 'name' has data 'SHILPI' then its length is 6. If we change the data of 'name' to 'VISUAL BASIC' then its length is 12.

#### 2.4.5 Assigning Values To Variables

After declaring various variables using Dim statement, you need to assign values to these variables. To assign values to these variables use '=' as per following format:

##### Format

Variable = Expression

Where

- Variable is the variable name, which has been declared by using Dim statement.
- Expression can be literal or another variable or mathematical expression. We discuss literal in 2.5 of this chapter.

##### Example

Rollno = 11

This stores value 11 in a variable 'Rollno', which is already declared by using Dim statement.

**Table 2.6**

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

#### 2.4.6 Variable Default Values

When you create a variable, VB automatically assigns a value to it. This value is known as *default value* of the variable. This value is depending upon the data type of the variable. Table 2.6 shows the default values of different data type.

### 2.4.7 Implicit And Explicit Variable Declaration

VB supports two types of variable declarations:

- Implicit Declaration
- Explicit Declaration

#### Implicit Declaration

Till now you learnt to declare variable. But even if we use a variable without declaring it, VB program will not produce any error message. As VB automatically creates a variable with Variant data type, if the variable being used has not been declared prior to it. These type of variable which don't declare before using it is called *Implicit Variable*.

#### Explicit Declaration

The variables which are properly declared by using keywords before using it is called Explicit Variable.

#### Example

```
Dim num As integer
num = val(Text1.Text)
sqrval = Sqr(num)
```

In above example 'num' is declared explicitly but another variable is implicitly declared.

First is 'sqrval' as it doesn't declared before so VB assign its data type as Variant.

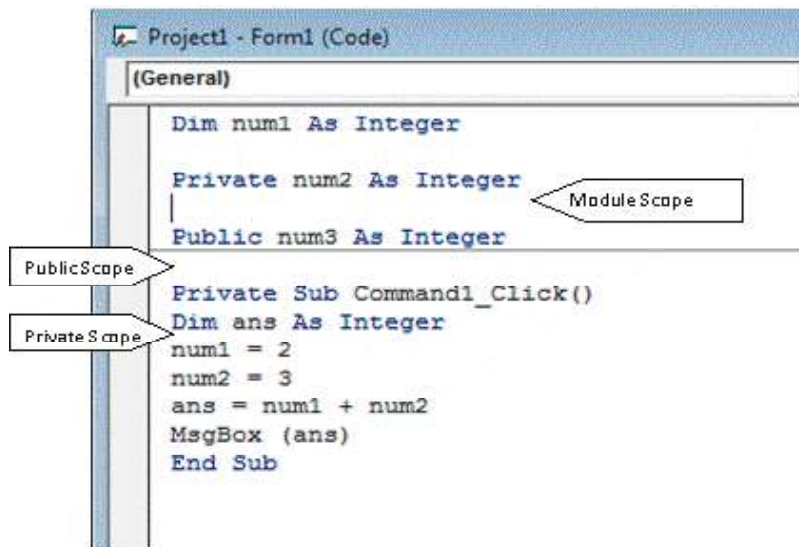
### 2.4.8 Variable Scope And Lifetime

When you declare a variable, it's not necessary that you can use them everywhere in your program. This is because a variable may not be accessible everywhere in the program. The area of a program in which a variable is accessible is called scope.

Part of program in which a variable is accessible is known as its Scope.

There are three different type of scope in VB :

- Private or Local Scope
- Module Scope
- Public Scope



*Fig. 2.1*

### Private or Local Scope

A variable that can be used only in one procedure, in which it is declared, is said to have private scope. If you give a Dim statement within a procedure, it means that this variable will be available and accessible only within this specific procedure and this variable is said to have private scope.

In fig 2.1, the variable 'ans' is declared within `command1_click()` procedure by Dim statement. This 'ans' variable can only be used within this procedure.

### Module Scope

Modules are place where you can put most commonly used routines, functions, constants, variables etc. These things may be used in many projects and anywhere inside a project. In VB there are three kinds of modules–

- (a) **Form Module:** It stores everything related to form. Eg- Event procedures for all controls placed in it, variables and constants being used in it etc.
- (b) **Standard Module:** It stores commonly used variables, constants and procedures etc.
- (c) **Class Module:** It stores code to create new objects.

A variable that is available inside a module i.e to all procedures in that module is said to have module scope.

In fig 2.1, variable 'num1' and 'num2' are having module scope.

## Public Scope

The variable available to all the modules and procedures in an application are said to have public scope. Since public variable are available to all procedures and modules of an application, this is also known as global variable.

In fig 2.1 , the variable ‘num3’ is public variable.

The time for which a variable lives in the memory is called its *lifetime*. A variable having private scope lives in memory as long as its parent procedure is being run. As soon as parent procedure ends, the variable is removed from the memory. The lifetime of module scope variable exists as long as their parent module is open. When the form is closed or removed from the memory, all its variable gets removed from the memory. The longest lifetime is occupied by public variables. These variables are available in the memory as soon as their parent application starts up and lives in the memory as long as application runs.

The time for which a variable lives in the memory is known as lifetime.

### 2.4.9 Static Variable

We discussed that private variable have lives in the memory as long as its parent procedure runs. But there is one exception here. If you declare a private variable with keyword static as follows :

```
Static <varname> As <data type>
```

The lifetime of the variable changes, now the variable lives in the memory even after its parent procedure is over. That means, static variable is not removed from the memory even after its procedure is over. Thus it retains its value and when next time its procedure gets executed, the static variable old value is accessible. However, the scope of static variable remains private to the procedure.

## 2.5 LITERALS

The value assigned to the variable is called literals. A literal whose data type is not specific is considered to be variant data type. Following rules must be followed to specify a literal:

- String literals must be declared within quotation marks. Eg. “shilpi”, “Visual Basic”, “21<sup>st</sup>”, “3214”. If quotation marks don’t have any space between them, “”, then it indicates a null or empty string.
- Date and time value must be declared in #. Eg - #17.05 pm#, #19 September 2011#.
- Boolean literals are either true or false.

The moment you add the suffix character to a literal, it becomes Constant. Constants are declares as follows:

### Format

```
Const <Constant name> = <value>
```

### Example

Const Pi= 3.14 This defines a constant Pi with value 3.14. User will not allow to change this value in entire module.

A constant is a fixed value i.e a value that doesn't change whereas a literal is a certain data value.

## 2.6 OPERATORS IN VB

In order to use variables efficiently, you need to use operators. Operators are basically some symbols that trigger an action and the data on which the operators work are called operands.

Operators are the symbols that trigger an action on some data. The data on which the operators operate are called Operands.

Operators in VB are divided into following categories:

- Mathematical / Arithmetic operators
- Concatenation or string operator
- Comparison operators
- Logical operators

### 2.6.1 Arithmetic Operator

VB supports a number of different math operators that can be used in program statement. Table 2.7 demonstrate the usage of different math operator by VB.

### 2.6.2 Concatenation Operator

The concatenation operator '&' is used to join two or more strings.

**Example:**

**Table 2.7**

Operator	Operation	Function	Example: Using values Val1 = 10 & val2 = 3
+	Addition	It adds two number	Val1 + val2 = 13
-	Subtraction	It subtracts second number from first number	Val1 - val2 = 7
*	Multiplication	It multiplies two number	Val1 * val2 = 30
/	Division	It divides two number	Val1 / val2 = 3.33
\	Integer Division	It divides two number add returns the integer quotient	Val1 \ val2 = 3
Mod	Modulus	It gives remainder of a division operation.	Val1 Mod val2 = 1
^	Exponentiation	It returns a number raised to the power of another number.	Val1 ^ val2 = 1000



```

Str1 = "Visual"
Str2 = "Basic"
Book = Str1 & " " & str2
Msgbox (Book)

```

The above code will display a message box with text ' Visual Basic '.



'+' sign is also used to concatenates two string.

### 2.6.3 Comparison Operator

Operators demonstrate in table 2.8 are used to compare the values of two or more variables or expressions.

### 2.6.4 Logical Operator

**Table 2.8**

<b>Operator</b>	<b>Operation</b>	<b>Function</b>	<b>Example: Using values Val1 = 10 &amp; val2 = 5</b>
=	Equal to	Returns true if both values are equal.	Val1=val2 is false
<>	Not equal to	Returns true if both values are not equal.	Val1<>val2 is true
>	Greater Than	Returns true if first value is greater than second	Val1>val2 is true
>=	Greater than or equal to	Returns true if first value is either greater than or equal to second	Val1>=val2 is true
<	Less than	Returns true if first value is lesser than second	Val1<val2 is false
<=	Less than or equal to	Returns true if first value is either lesser than or equal to second	Val1<=val2 is false

The logical operators are used to combine two or more checking conditions. Commonly used logical operators are described in table 2.9.

**Table 2.9**

<b>Operator</b>	<b>Function</b>	<b>Example : using variable Val1 = 200 &amp; val 2 = 300</b>
AND	If both operand conditions are true, it returns true.	(i) val1>=100 AND val2<=500, returns true (ii) val1>=100 AND val2<300, returns false
OR	If at least one operand conditions are true , it returns true.	(i) val1>=100 OR val2<=500, returns true (ii) val1>=100 OR val2<300, returns true
NOT	Negates the result of condition	(i) NOT(val1 >=200) returns false (ii) NOT(val1<200) returns true
XOR	Returns true if one operand condition is true and other is false. It returns false if both operands conditions are either true or false simultaneously.	(i) val1=200 XOR val2= 100, returns true. (ii) val1=200 XOR val2=300, returns false

### 2.6.5 Operators Precedence

A VB statement may have one or more operators in it. In such case, VB evaluates the expression depending upon the precedence of the operator. The precedence order is given below in table 2.10.

**Table 2.10**

<b>Operator</b>	<b>Operation</b>
^	Exponentiation
-	Unary Minus
*, /	Multiplication, Division
\	Integer Division
Mod	Remainder
+, -	Addition, Subtraction
&	Concatenations
=, <>, >, <, >=, <=	Comparison
NOT, AND, OR, XOR	Logical Operators

## 2.7 SOME USEFUL FUNCTION

Although you have learnt the basic things of programming which you must know before creating a program. There are two useful functions: Rnd() and Format(), which you will use in next chapter. So before moving you must aware of that function.

### 2.7.1 Rnd Function (Random Number Generator)

The Rnd() function is used to generate a random number. The Rnd() function returns a single value that contains a randomly generated number less than 1 but greater than or equal to 0. This is used as

#### Syntax:

```
Rnd [ ( number ) ]
```

The optional number argument can be used to determine how Rnd() generates the random number. It generates the number as explained in table 2.11.

#### Example:

```
Dim val
```

```
Val = Int ( ( 6 * Rnd ) + 1 )
```

The above statement generates a random value between 1 to 6.

**Table 2.11**

If number is	Rnd generates
Less than 0	The same number every time
Greater than 0	The next random number in the sequence
Equal to 0	The most recently generated number
Not supplied	The next random number in the sequence.

To produce random integers in a given range, use following formula:

$$\text{Int} ( ( \text{upperbound} - \text{lowerbound} + 1 ) * \text{Rnd} + \text{lowerbound} )$$

Where upperbound is the highest number of the range and lowerbound is the lowest number in the range.

### 2.7.2 Format Function

While working in an application, sometime you need to present data in a particular formatted way. VB provides a Format function to do this. The syntax of format function is :

#### Syntax

```
Format ( expression [, format [, firstdayoftheweek [, firstweekoftheyear ] ] ] )
```

Where

- Expression is required.
- Format is optional. It is a valid name or user defined format expression.
- First day of the week is optional. It is a constant that specifies the first day of the week.

↑ High  
Resolved left to right  
Low

- First week of the year is optional. It is a constant that specifies the first week of the year.

The expression argument specifies a number to convert, and the format argument is a string made up of symbols that shows how to format a number. The most commonly used symbols are listed in table 2.12.

### Example

```
Format(1997.9, "00000.00") results 01997.90
Format(1997.9, "#####.##") results 1997.9
Format(1997.9, "#,###.00") results 1,997.90
Format(now, "dd-mmm") results 19 – sep
Format(now, "dd-mm-yyyy hh:mm") results 19 september 2011 7:45
```

**Table 2.12**

Symbol	Description
0	Digit placeholder, prints a trailing or leading 0 in this position.
#	Digit placeholder, never prints leading or trailing 0.
.	Decimal placeholder
,	Thousands separator.
- + \$	Characters that display exactly as typed
dd/mmm/yy	Short date/month /year
mmm/yyy	Long month/year
hh/mm/ss	Hour/minute/second

### 2.7.3 val Function

In VB , we enter digits in the field of characters. So to make sure that digits are treated as number instead of string , we use val ( ) function. It converts a string value having digits into equivalent number. It has following syntax:

#### Syntax

Val ( String )

Where string is required.

#### Example

Val("19") results 19

**LET US REVISE**

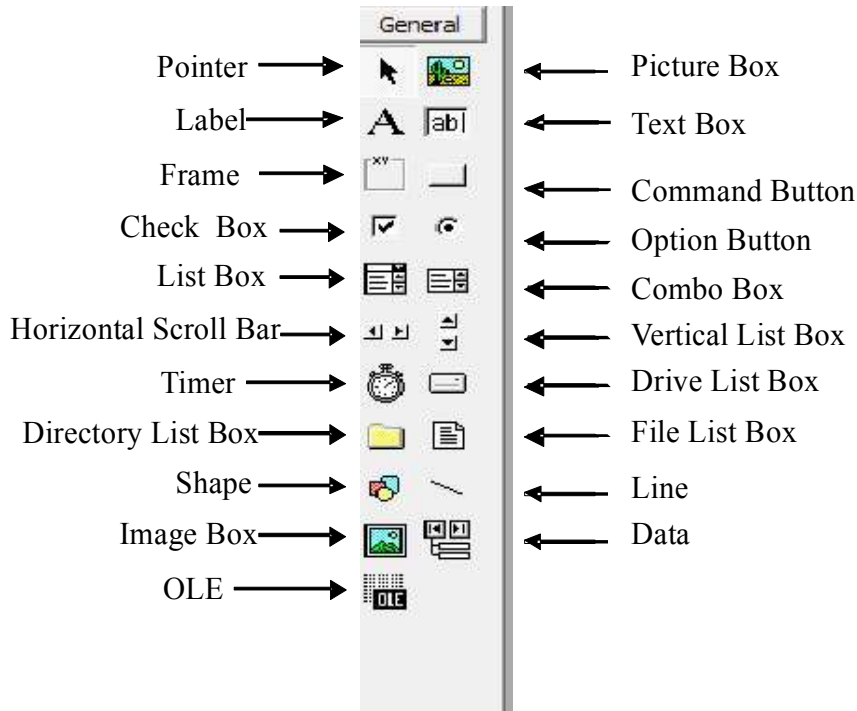
- ✓ Keyword are the word whose meaning is already defined in compiler.
- ✓ Data type are means to identify the type of data and associated operations with it.
- ✓ Data types supported by VB are Boolean, Byte, Currency, Date, Double, Integer, Long, Object, String, Variant.
- ✓ String can either be fixed length or variable length.
- ✓ A named storage location whose content can be varied is called variable.
- ✓ Variables can be declared by Dim statement.
- ✓ If data type of a variable is not specified, VB assumes it as variant.
- ✓ Variable which are not declared before using are called implicit variable
- ✓ A literal is a value that is assigned to a variable.
- ✓ val( ) converts a string value having digits into equivalent number.
- ✓ Str( ) converts a number into equivalent string.
- ✓ In VB a variable can have private, module or public scope.
- ✓ The time for which a variable lives in the memory is known as its lifetime.
- ✓ Symbols or word that triggers an action are known as operators.
- ✓ VB supports arithmetic, concatenation, comparison, logical operator.
- ✓ Rnd function is used to generate random function.
- ✓ Format function is used to present values in a formatted way.

# Chapter-3

## Working With Controls

### 3.1 INTRODUCTION

The VB toolbox contains many tools to design user interface quickly and easily. On screen you find twenty controls to work. These controls are command box, text box, label, list box, combo box etc. You can also put many more controls on window by accessing them from components. In this chapter you learn to work with controls, uses of different controls and their properties.



## 3.2 INTRINSIC AND CONTAINER CONTROL

The VB toolbox contains the tools that can use to draw controls on the form. The controls that are always included in toolbox and act as an integral part of Standard Exe project are known as Intrinsic Control.

A Container Control can hold other controls within it. The controls inside a container are called child controls. These child controls cannot move outside the container control. When you delete a container control, all its child automatically get deleted. Eg: Frame, PictureBox.

## 3.3 WORKING WITH CONTROLS

### 3.3.1 To Draw a Control On a Form

To draw a control on the form you have to follow the following steps –

1. Click on the desired control icon that is available on toolbox.
2. Now, move the mouse pointer over the form. When your pointer comes over the form it changes to crosshair (+) instead of arrow. Take it on the desired location and click and hold the left mouse button.
3. While holding the left mouse button, drag the mouse down till it comes to the desired size. Now release the mouse button.

You can also add a control on the form by double click on the control. This will automatically add this control on the form.

**To move** a control, click on the control and drag it to the new location. Now release the mouse button.

**To resize** a control, select it first and then use its sizing handle to resize it. When mouse pointer moves over the sizing handles, the pointer will change to double headed arrow. Now press the left mouse button, hold it and drag the sizing handle to desired new position.

To **delete** a control, select the control by clicking it and press DEL (Delete) key.

### 3.3.2 Object Naming Conventions

Every time when we put a control on the form, VB automatically gives a name to it. Name is one of the very important attribute as whenever you want to do something you have to call control by its name. If you add a Command Button VB name it Command1, similarly for textbox it named Text1.

When there are multiple controls on the form, naming like this can be very confusing as it does not mention anything characteristic of control. Thus, while naming control, we need to follow the following points –

- Name must begin with a letter.
- It must contain only letters, numbers and underscore.
- It must not be longer than 40 characters.

You can choose prefix as define in table 3.1 , to describe the class, followed by the descriptive name for the control. This naming convention makes the control more self descriptive.

Example – cmdadd

Above name describes that it is a command button which is performing addition of numbers.

**Table 3.1**

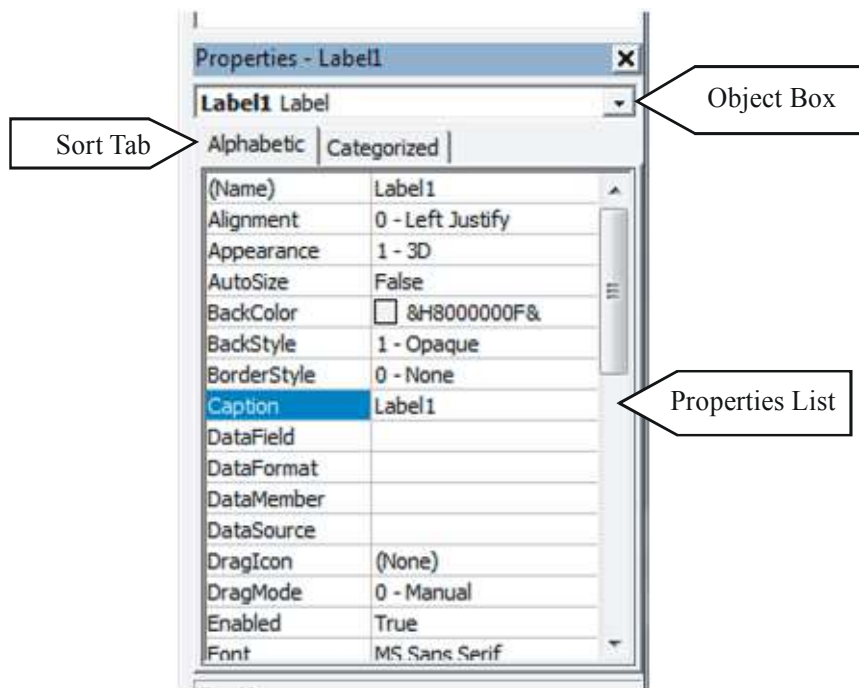
Control	Prefix	Control	Prefix
Label	Lbl	Picture Box	pic
Frame	fra	Text Box	txt
Check Box	Chk	Command Button	cmd
List box	Lst	Option Button	opt
Horizontal Scroll Bar	Hsb	Combo Box	cbo
Timer	Tmr	Vertical scroll bar	vsb
Directory List Box	Dir	Drive List Box	drv
Shape	Shp	File List Box	fil
Image Box	Img	Line	lin
OLE	Ole	Data	dat

### 3.3.3 Setting Properties

Every control has some properties which make it different from rest of the controls. The property window provides an easy way to set properties of the selected control. The property window consists of the following elements

- **Object Box:** It displays the name of the control for which you set properties.
- **Sort Tab:** It gives a choice between alphabetic list or hierarchical view of properties.
- **Properties List:** It displays all the properties of the selected object.





To set properties from the properties window, we do following –

1. From properties list, select the name of the property.
2. In right column, type or select the new property.

### 3.4 CREATE EVENT PROCEDURE

Code in VB is divided into smaller blocks. These blocks are called procedures. An event procedure is a procedure containing code that is executed when an event occurs. An event procedure for a control is comprised of the control's name, underscore and the event name. A procedure can be written between Sub and End Sub.

**Example:** If you want a command button named 'cmdadd' to invoke an event procedure when it is clicked, is written as

```
Sub cmdadd_click( )
```

```
End Sub
```

There are following three ways to open the code window

1. Double click on the object for which event procedure to be written. It will invoke code editor window for the selected object.

2. Double click on the form and then select the control from the object list box.
3. In Project Explorer window, click on the code view.

Now let's begin with the controls and discuss their important properties and events.

### 3.5 MANIPULATING FORMS

Form is the top level object in a VB application and every application must start with the form. Form acts as a container for all the controls that make up the user interface. At run time forms are called *windows*. There are two types of forms:

1. **Modal:** A modal form takes total control of the application. It prevents user interaction with any other window in the application besides the one that is shown. That is, it will not let the user perform any action while modal form is open.

Syntax:

```
<formname>.Show.[ Vbmodal ]
```

Where <formname> is the name of the form and Vbmodal is optional field.

2. **Modaless:** Modaless form can interact with the user and they allow the user to switch to any other form of the application. By default, forms are modaless.

Syntax:

```
<formname>.Show.[ Vbmodaless ]
```

Where <formname> is the name of the form and Vbmodaless is optional field.

#### 3.5.1 Events of Form

Certain events can fire every time when a user moves from one form to another. These events are:

1. **Initialize event:** This is the first event written which is used to initialize the variable used in the form.
2. **Load Event:** This event happens one time for each form when it is loaded into the memory and is ready to display. It includes code that is executed when this event occurs. A form can be loaded by giving the succeeding syntax:

```
Load <formname>
```

3. **Activate/Deactivate:** The activate event occurs whenever the form becomes the active window i.e it is receiving events from the user.

To show a form you may use the syntax:

```
<formname>.Show (e.g abc. show, where abc is the name of form and show is the method name)
```

To hide a form you use the syntax:

```
<formname>.Hide
```

4. **Unload:** This event is fired when unload is involved. In this all variables and objects should be cleared from the memory. This can also be done by using the statement Unload Me

**5. Terminate:** This is final event triggered from a form which indicates that VB has removed the form.

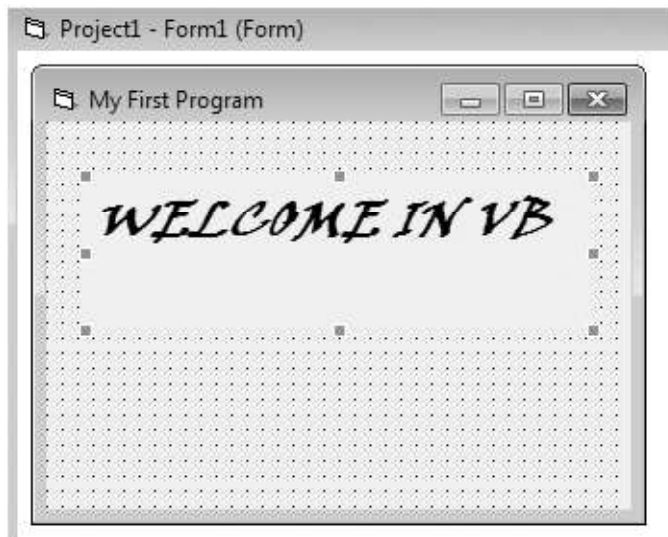
### 3.5.2 Form Properties

The appearance of the form can be customized by changing properties of form from properties window. Some commonly used properties are demonstrated in table 3.2.

**Table 3.2**

Property	Description
Name	Specifies the name of the form by which it is call at run time. By default VB call the first form as Form1
Caption	Holds the text that appears on the title bar of the form.
MinButton, Maxbutton	When these properties are true, it displays minimize and maximize button on the title bar.
BorderStyle	Determines the border style and appearance of the form.
BackColor	Specifies the form background color.
Enabled	Determines whether the form is active or not. By default its prop erty is true.
Font	Sets the text font name, style and size for all the controls that are placed on form.
Movable	Specifies whether the form can move at runtime or not.


**Example 1:** Design a form to welcome you in VB.



Open a form and placed a label box on it set their properties as follows:

Control	Property
Form	Name – Form1 Caption – My First Program Font – Viner Hand ITC, style –bold, size 20
Label	Name – Label1 Caption – WELCOME IN VB

First save this form with the extension *.frm* . Once you save the form immediately it asks you to save the project with the extension *.vbproj* .

When you click on the Run button  which is available on toolbar, following window will appear.



### 3.6 BASIC CONTROLS

Before writing an event procedure for the control to response to a user's input, you have to set certain properties for the control to determine its appearance and how it will work with the event procedure. Now let's start with controls that are available on toolbox.

#### 3.6.1 Pointer

This is the only control on the toolbox which cannot draw any thing. Its job is to select, resize or move a control that has already placed on a form.

#### 3.6.2 Label

It displays text that the user can not change directly. Only the person who designs the form can do it. Some major properties of control are:

Property	Description
Name	Used to name the label control
Caption	Used to display the text on the label control
Font	Used to specify the font, style and size of displayed text
Alignment	Sets alignment of the label's caption
WordWrap	Sets word wrap for the caption text. By default, if text exceeds the width of the label, it automatically wrap to the next line.
Autosize	Allows the control to horizontally expand according to the caption

### 3.6.3 Text Box

The text box is the standard control for accepting input from the user as well as to display the output. It can handle string (text) and numeric data but not images or pictures. String in a text box can be converted to a numeric data by using the function `Val(text)`. Major properties of text box are:

Property	Description
Name	Used to name the textbox control
Text	Specifies text to be displayed. This text can be edited at run time. Usually you have to clear the box at design time.
MaxLength	Sets the maximum number of characters allowed in the textbox.
Enabled	Determines whether the text box is active or not. By default its property is true.
Multiline	If it is true, it allows multiple lines of text in the text box.
Scrollbar	Enables to attach a built in scroll bar to the text box.
Password Character	It is important property if you want to make the field as password. If you specify a character in this property, then in place of text being entered, this character is displayed.

### 3.6.4 Command Button

The Command Button is used to initiate actions, usually by clicking on it. When user pushed and released the button, it carries out the appropriate action. Some common properties are:

Property	Description
Name	Used to name the command button
Caption	Used to display the text on the command button
BackColor	Specifies the background color.

Cancel	Determines whether the command button gets a click event if user presses Esc key.
Default	Determines whether the command button respond to an Enter key
Enabled	Determines whether the text box is active or not. By default its property is true.
Font	Used to specify the font, style and size of displayed text
MousePointer	Determines the shape of the mouse cursor when user moves the mouse over the command button.
Picture	Allows to display a selected graphical image on the command button when the <i>style</i> property is set to 1-Graphical.
Visible	Determines the command button appears or hidden from the user
Tooltip Text	Display the text that appears as a tooltip at runtime.

**Default Button** is that command button whose click event can be activated by pressing Enter key even though focus is not on the command button. **Cancel Button** is that command button whose click event gets activated by pressing Esc key. There is only one default and one cancel button on the form.

### Assigning Access key

To assign a keyboard access key to a command button, place an ampersand (&) in front of the letter that is to be used as access key while setting the caption property.

Assigning access key at design time	Access key appear at runtime
&Print	<u>P</u> rint

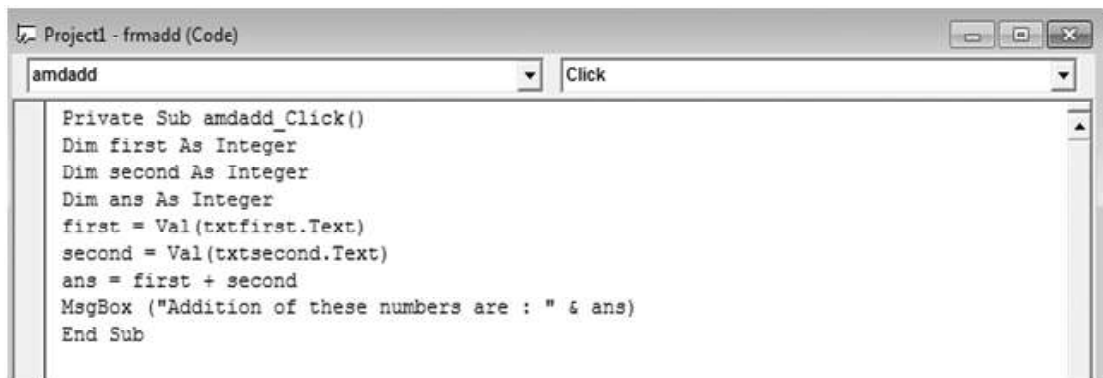
**Example 2:** Write a program to accept two numbers from users and display their sum in message box.

**Solution:** Design the form as:

The screenshot shows a Windows-style form titled "ADDITION OF TWO NUMBER". The form has a light gray background with a dotted grid pattern. It contains two text boxes for input, each preceded by a label: "Enter First Number" and "Enter Second Number". Below these is a button labeled "ADD".

Control	Property
Form	Name – frmadd
Label 1	Caption – ADDITION OF TWO NUMBER
Label 2	Name – lblfirst
Text box 1	Caption – Enter First Number
Text box 2	Name – lblsecond
Comm and Button	Caption – Enter Second Number
	Name – txtfirst
	Text – [Empty]
	Name – txtsecond
	Text – [Empty]
	Name – cmdadd
	Caption - ADD

Now write code on the click event of add button as

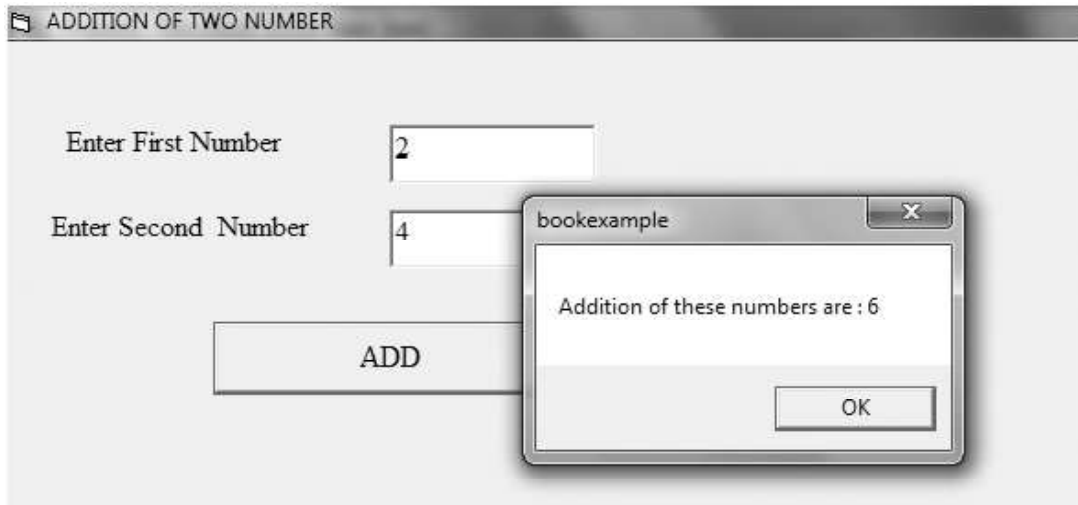


```

Project1 - frmadd (Code)
cmdadd Click
Private Sub cmdadd_Click()
    Dim first As Integer
    Dim second As Integer
    Dim ans As Integer
    first = Val(txtfirst.Text)
    second = Val(txtsecond.Text)
    ans = first + second
    MsgBox ("Addition of these numbers are : " & ans)
End Sub

```

Now save and run your program. At run time it appears as



Note : If you are adding a new form in the existing project. Then to run the second form go to Project — Project1 Properties and set form name as startup object.

### 3.6.5 Check Box

A check box control offers a small set of choices from which a user can choose one or more options. These check boxes works independently , thus user can select any number of boxes at runtime. Following are the main properties of check box:

Property	Description
Name	Used to name the check box
Caption	Used to display the text with the check box
Enabled	Determines whether the text box is active or not. By default its property is true.
Value	Used to specify the state of check box. If its value is true or 1, the check box is checked otherwise unchecked.

### 3.6.6 Option Button

An option button control offers a small set of choices from which a user can choose only one option. It always works in group. Selecting one option immediately clears the selected button from the group. Following are the main properties of check box:



Property	Description
Name	Used to name the option button
Caption	Used to display the text with the option button
Enabled	Determines whether the text box is active or not. By default its property is true.
Value	Used to specify the state of option button. If its value is true or 1, the option button is selected.

### 3.6.7 Frame Control

It is a container control which is used to group various other controls. It does not carry out any action and does not respond to any event by itself. Some major properties of frame are :

Property	Description
Name	Used to name the frame
Caption	Used to display the text in the frame control
BorderStyle	Used to specify the style of border. It can be 0 or 1
Appearance	Used to set the look of the frame. It can be either 0-flat or 1-3D

### 3.6.8 ListBox Control

It presents a list of choices to the user in a column. If the number of item exceeds in the list box, scroll bar automatically appear on the control. Major properties of listbox are:

Property	Description
Name	Used to name the listbox
Sorted	Determines whether the items in the list are sorted or not.
Style	Determines the appearance of the control. It can be either 0-standard or 1-checkbox style
Multi Select	Determines whether user can select multiple item from list or not. It can take following values: 0 – Multiple selection not allowed 1 – Simple multiple selection. Item can be selected or deselected through mouse. 2 – Extended multiple selection. Items can be selected by pressing shift and clicking the mouse.

### Common Methods of Listbox

With listbox you are able to do the following tasks:

- 1. Add items to the list:** To add items in the list, use **AddItem** method as follows:

Syntax:

```
List1.AddItem<item>,<index>
```

Where List1 is the name of the list control, item is the string to be added and index is its order. Index is the optional argument.

- 2. Remove items from the list:** To remove an item from the list you must know its index value. An item is removed from the list by using **RemoveItem** method with index value as follows:

Syntax:

```
List1.RemoveItem<index>
```

To count the total number of items in the list , use **ListCount** method

To remove first item from the list:

```
List1.RemoveItem 0
```

To remove last item of the list:

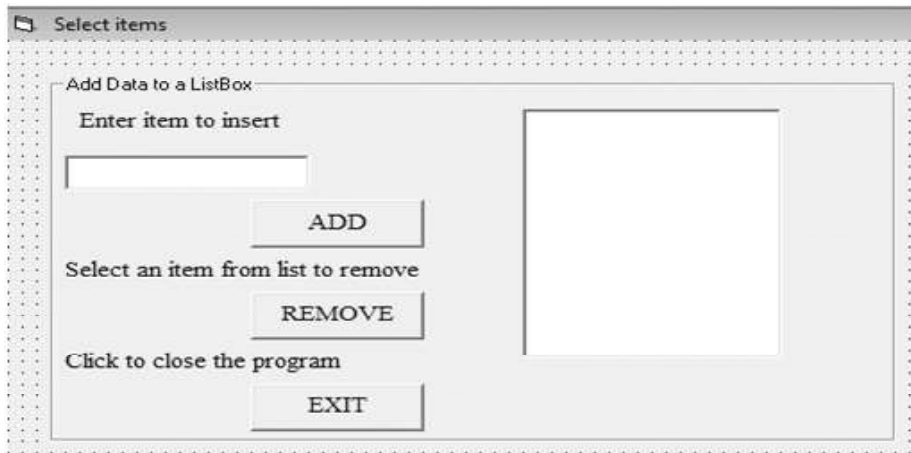
```
List1.RemoveItem Listcount-1
```

- 3. Access individual items in the list:** The items in a list box are sorted with List() property. List(0) holds first item, List(4) holds fifth item of the list. **ListIndex** property gives the index of selected item in the list. If multiple items are selected then listindex stores the index of most recently selected item. To remove the selected item from the list we use the following syntax:

```
List1.RemoveItem List1.Listindex
```

**Example 3:** Write a program to accept an item from user and add that item in the list with the privilege to delete any item from the list.

**Solution:** Design the form as follows:



Control	Property
Form	Name – frm1ist
Label 1	Caption – Select Item
Label 2	Name – lbltext
Label 3	Caption – Enter item to insert
Textbox 1	Name – lblremove Caption – Select an item from list to remove Name – lblexit Caption – Click to close the program Name – txtitem Text – [Empty]
List Box	Name – lstitem
Command 1	Name – cmdadd Caption – ADD
Command 2	Name – cmdremove Caption – REMOVE
Command 3	Name – cmdexit Caption – EXIT

Now write the code on click event of command button as:

```

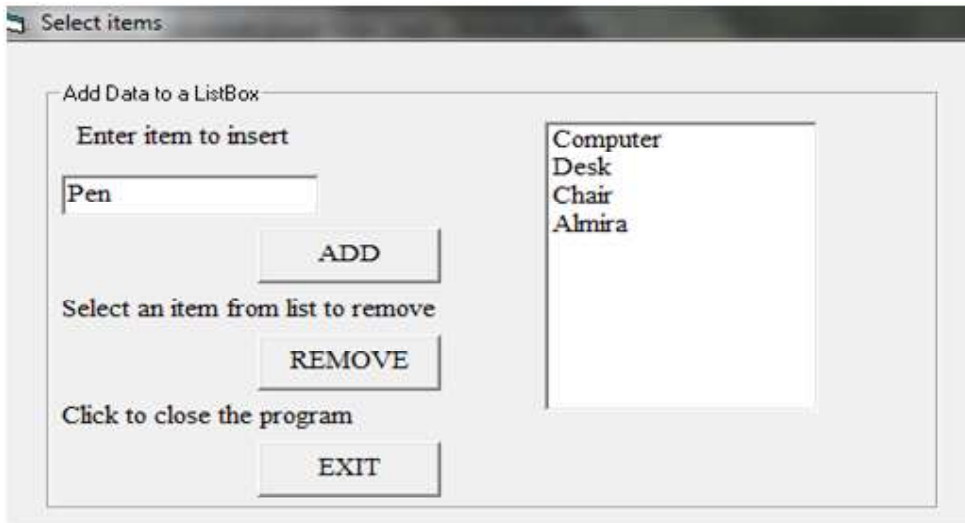
cmdremove
Click
Private Sub cmdadd_Click()
    Lstitem.AddItem txtitem.Text
    txtitem.Text = ""
    txtitem.SetFocus
End Sub

Private Sub cmdexit_Click()
End
End Sub

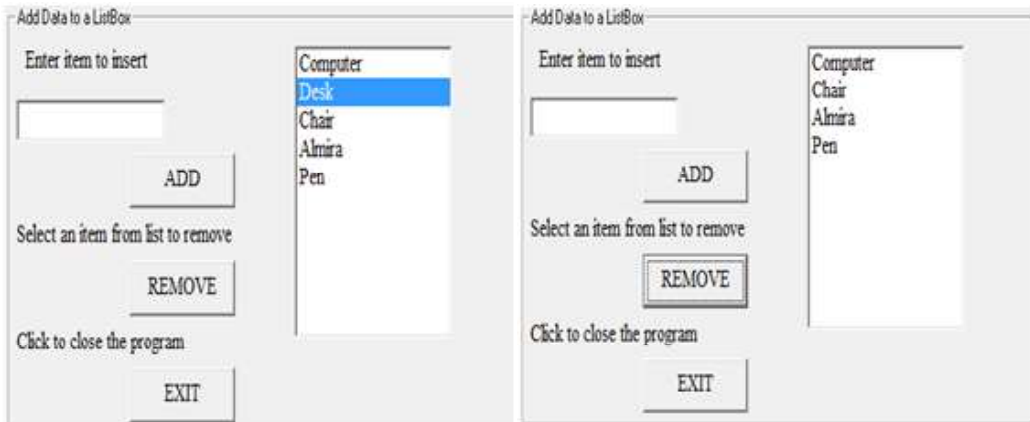
Private Sub cmdremove_Click()
    Lstitem.RemoveItem Lstitem.ListIndex
End Sub

```

At run time, at the time of adding items it appears as:



When we select an item from list, while clicking on remove button that item will be deleted



### 3.6.9 ComboBox Control

Combo box is a combination of text box and list box. It contains multiple item in its listbox. User can select any item from given list. If his desired item is not in the listbox, he can able to select that item by writing it in the textbox of combobox. It is practically an extendable listbox control, which can

grow when user wants to make a selection and retract after the selection is made. It has following common properties:

Property	Description
Name	Used to name the combobox
Sorted	Determines whether the items in the list are sorted or not.
Style	Determines the appearance of the control. There are following three type of combobox control 0 – Drop Down Combo. The control is made up of dropdown list and textbox. The user can select an item from list or type his own item. This is default combobox. 1-Simple Combo. Includes a textbox and a list that doesn't drop down. The user can select from the list or write his own item. 2- DropDown List. This is a drop down list, from which user can select an item but doesn't enter a new one.

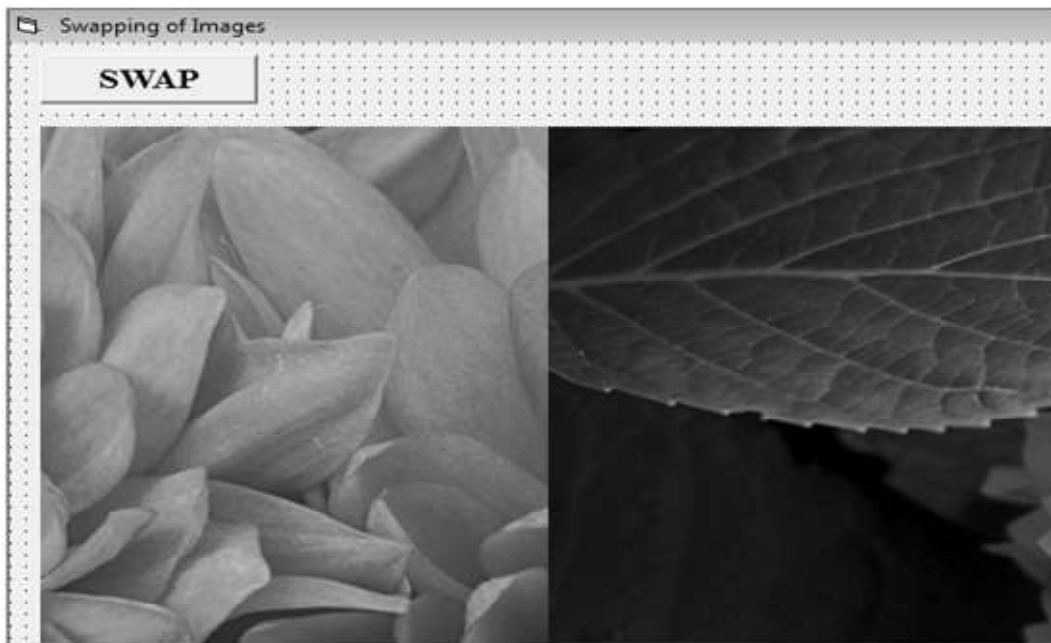
### 3.6.10 Image Box

The image control is used to display graphics in the format of : bitmap, icon, metafile, enhanced metafile, cursor, .jpeg or .gif files. Some common properties are:

Property	Description
Name	Used to name the Image box.
Appearance	Used to control the appearance. It can be set with or without 3D effect
BorderStyle	Determines border style of the control
Picture	Determines which picture will be displayed in the control. This can also be done with the help of <i>LoadPicture</i> function.
Stretch	Determines whether the image is resized or not.

**Example 4:** WAP to swap two images.

Design the form with following properties:

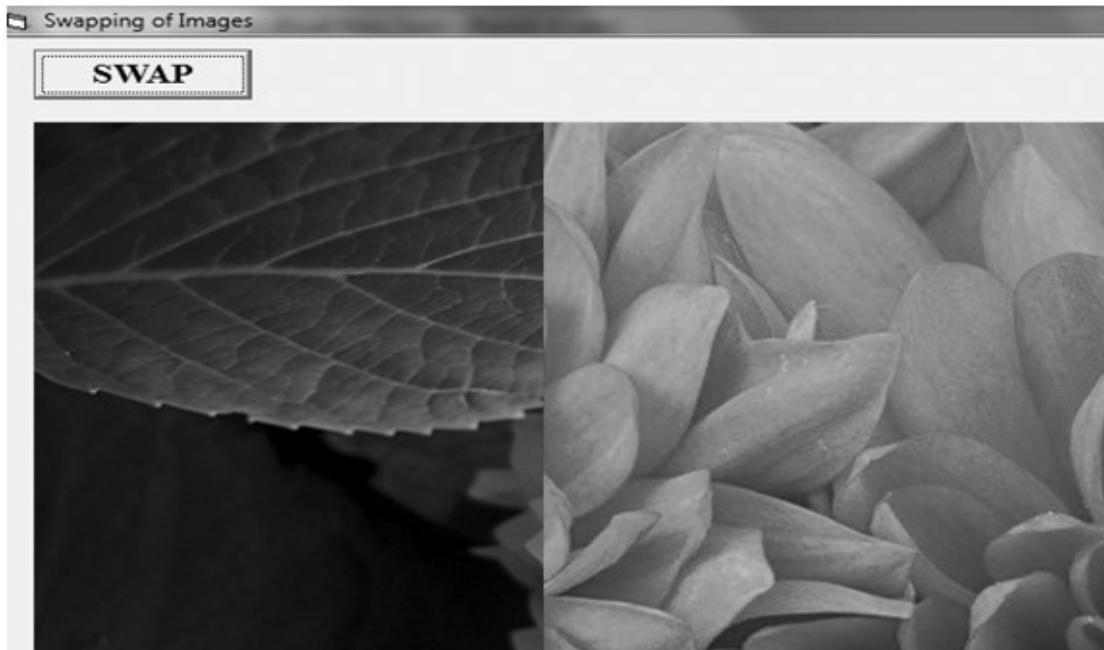


Control	Property
Form	Name – frmimage Caption – Swapping Of Images
Image 1	Name – image1 Picture – Select picture from dialog box Stretch – False
Image 2	Name – image2 Picture – Select picture from dialog box Stretch - False
Image 3	Name – image3 Stretch - False Visible – False
Command Button 1	Name – cmdswap Caption – SWAP

Write the following code

```
cmdswap
Click
Private Sub cmdswap_Click()
    Image3.Picture = Image1.Picture
    Image1.Picture = Image2.Picture
    Image2.Picture = Image3.Picture
End Sub
```

At runtime, after swapping it appear as



### 3.6.11 Picture Box

The picture box control is used to display graphics which act as a container for other controls. It is similar to the image box and supports its each graphic format . It is used to display output from graphics method or text using the Print method. It provides methods for drawing at runtime and is much more flexible than image box. Some common properties are:

Property	Description
Name	Used to name the Picture box.
BorderStyle	Determines border style of the control
Picture	Determines which picture will be displayed in the control. This can also be done with the help of <i>LoadPicture</i> function.
AutoSize	Determines whether picture box is automatically resized to display the entire image.

**Example 5:** WAP to load picture of penguins.

Design the form with following properties :



Control	Property
Form	Name – frmpicture Caption – Load Picture
Picture 1	Name – Picture1 AutoSize – True
Command Button 1	Name – cmdLoad Caption – Load Image

Write the following code

Path of picture



```
(General) (Declarations)
Private Sub cmdload_Click()
Picture1.Picture = LoadPicture("C:\Users\Public\Pictures\Sample Pictures\Penguins.jpg")
End Sub
```

At runtime it appears as:



### 3.6.12 Hscroll Bar & Vscroll Bar

Scroll bar is a graphical tool for quickly navigating a large list of items, by indicating the current position on the scale. There are two type of scroll bars :

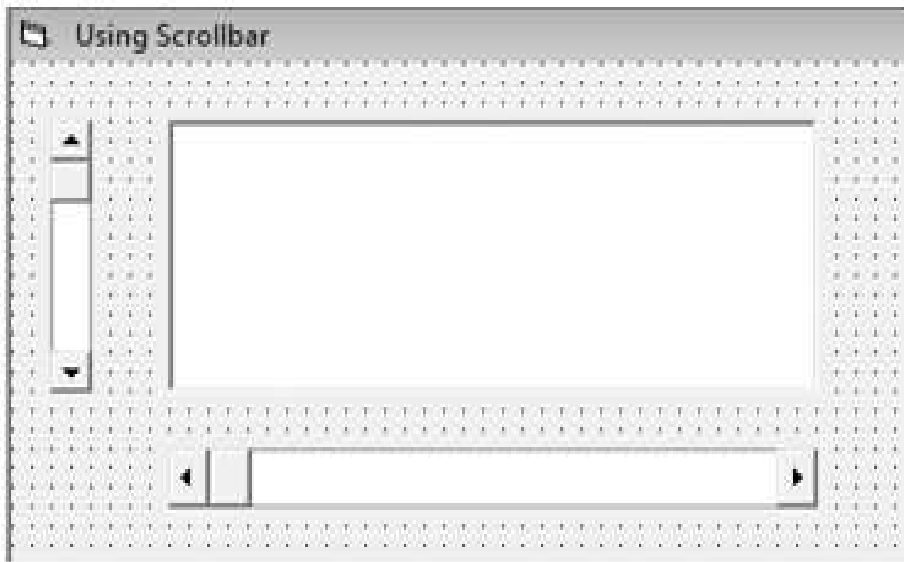
1. **Horizontal scroll bar (hscroll):** Moves left to right .
2. **Vertical scroll bar (vscroll):** Moves top to bottom.

The left end and top end corresponds to minimum value whereas other ends corresponds maximum value . Its value ranges between 0 to 32,767. Both have following common properties –

Property	Description
Name	Used to name the scrollbar.
Min	Returns scroll bar minimum value
Max	Returns scroll bar maximum value
Value	Returns current value of the scroll bar

**Example 6:** WAP to change the color of the textbox by using scrollbar.

Design the form with following properties :



Control	Property
Form	Name – frmscroll Caption – Using Scrollbar
Text 1	Name – txtcolor Locked – True Text – Nil
HScroll1	Name – Hscroll
VScroll1	Name – VScroll

Write the following code:

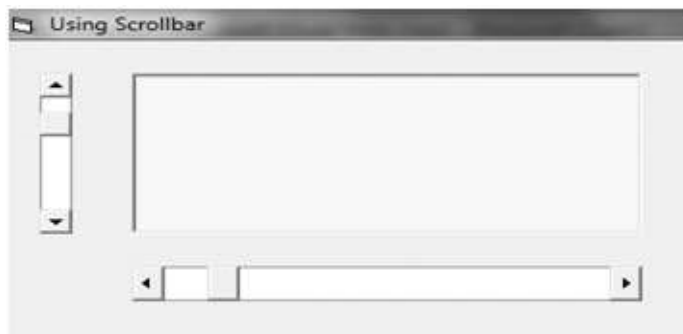
```

HScroll
Change
Private Sub HScroll_Change()
txtcolor.BackColor = RGB(VScroll.Value, HScroll.Value, 0)
End Sub

Private Sub VScroll_Change()
txtcolor.BackColor = RGB(VScroll.Value, HScroll.Value, 0)
End Sub

```

At runtime it appears as:

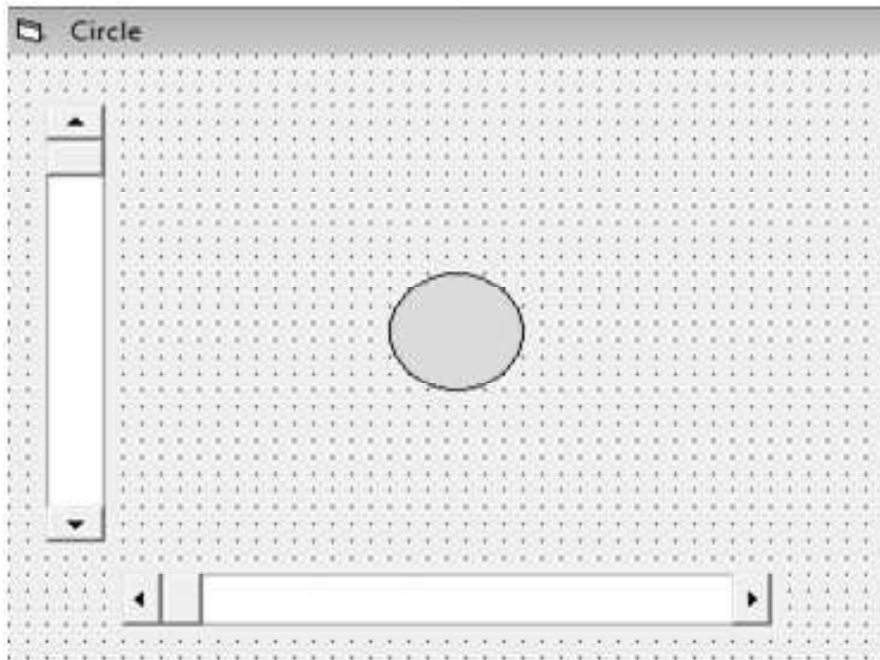


### 3.6.13 Shape & Line

These controls are used for drawing graphical elements on the form. They don't support any event, they only used for decorative purposes. Line is used to draw line only. They have following properties:

Property	Description
Name	Used to name the control.
Shape	Sets the shape of the control like rectangle, circle, oval etc
BorderColor	Sets the color of border of the shape
FillColor	Sets the color of the shape
FillStyle	Controls how shape is drawn
BorderStyle	Sets style of border. VB provides six border style
BorderWidth	Sets width of border line of shape

**Example 7:** WAP to change the size and shape of a circle by using scroll bar.  
Design the following form:

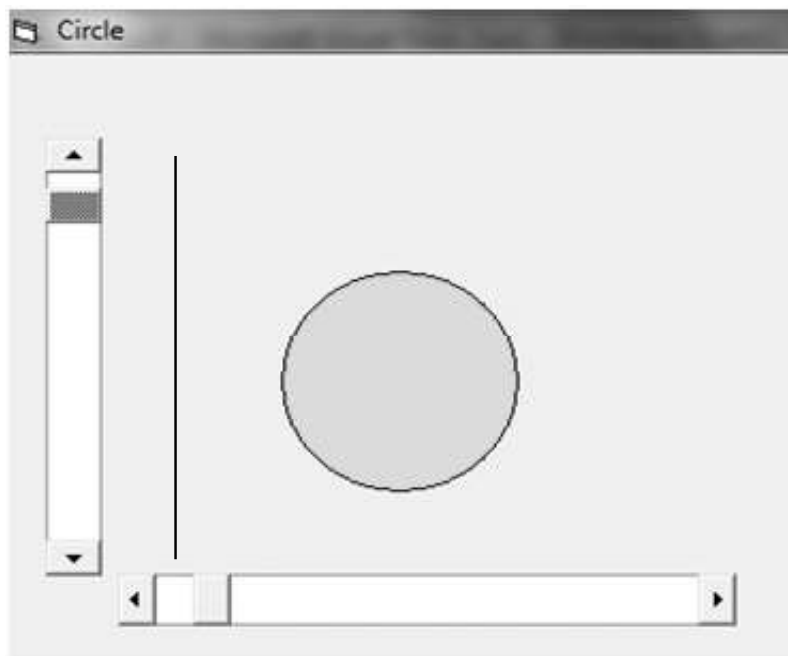


Control	Property
Form	Name – frmshape
Shape 1	Caption – Circle
	Name – shape
	Fill Color – Green Fill
	Style – 0 solid
	Style – Circle
HScroll1	Name – Hscroll
Vscroll1	Name – VScroll

Write the code as follows:

```
HScroll End Change
Private Sub HScroll_Change()
Shape.Width = HScroll.Value
End Sub
Private Sub VScroll_Change()
Shape.Height = VScroll.Value
End Sub
```

At runtime it appears as:



### 3.6.14 Timer

It is used to perform some activity at regular interval of time. This interval is specified by the programmer. This control is invisible at runtime. Instead of making a beep, the control executes code when the specified interval is complete. It has two properties:

Property	Description
Name	Used to name the timer.
Enable	It counts down repeatedly, as long as this property is true
Interval	It contains a value than lies between 0 – 65,535 milliseconds. Generally we set this to 1000 which indicates one second.

**Example 8:** WAP to display current date.

Design the form as:



Control	Property
Form	Name – frmtimer Caption – Date
Lable 1	Name – lbldate Caption – Nil
Timer1	Name – Timer Interval – 1000 Enable – true

At run time it appears as



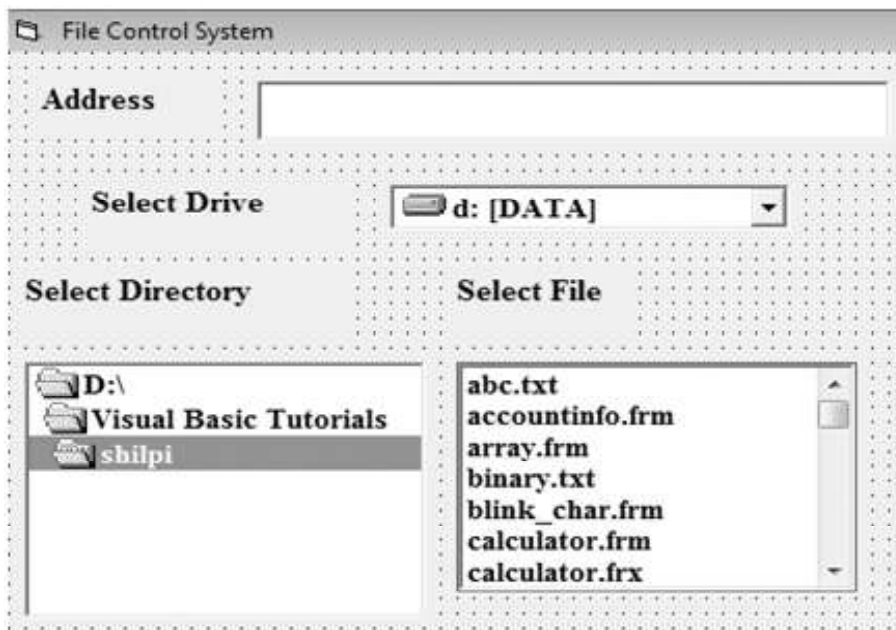
### 3.6.15 File System Control : DriveList Box , DirList Box & FileList Box

Using these controls user can access the host computer's file system, locate any folders or files on hard disk or on network drives. These controls are independent of one other but they always used together. These file controls has following function :

1. **Drive List Box:** Displays the name of drives within and connected to the computer. Its basic property is drive which sets the selected drive.
2. **Dir List Box:** Displays folder of the current drive. It displays all the folders or files stored in the drive which is selected by user through DriveListBox. Its basic property is Path which is name of the folder whose subfolders are displayed in the control.
3. **File List Box:** Displays the file of the current folder i.e folder displayed in DirList Box. Its basic properties are Path and Pattern. Path gives the path name of the folder whose files are displayed and pattern specifies which file will be displayed with a file matching string such as '\*.doc'.

**Example 9:** WAP to implement file system control which allows user to view the selected file address in the textbox.

Design the form with following properties



Control	Property
Form	Name – frmfile Caption – File Control System
Lable 1	Name – lbladd Caption – Address
Lable 2	Name – lbl dri Caption – Select Drive
Lable 3	Name – lbl dir Caption – Select Directory
Lable 4	Name – lbl file Caption – Select File
Textbox 1	Name – txtname Text – Nil Locked – True
Dir List Box	Name – Dir1
Drive List Box	Name – Drive1
File List Box	Name – File1

Write the following code:

```

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

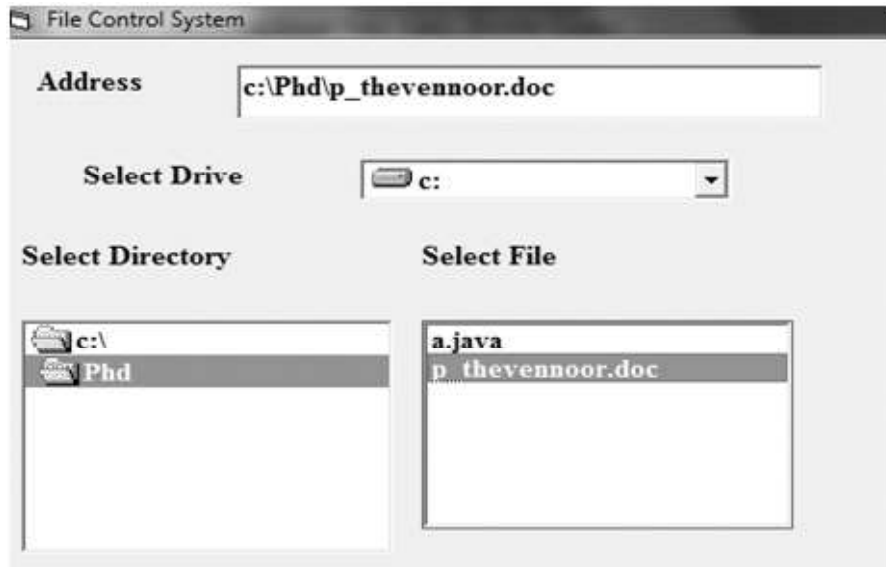
Private Sub Drive1_Change ()
Dir1.Path = Drive1.Drive
End Sub

Private Sub File1_Click ()
Dim s As String
s = Dir1.Path + "\" + File1.FileName
txtadd.Text = s
End Sub

```



At run time it appears as:



### 3.7 CONTROL ARRAYS

A control array is a group of controls that share common name, type and event procedures. Thus a single event procedure for an event will be applicable to all elements of a control array. A control array has atleast one element and can grow maximum upto 32767 elements. Elements of same control array have same name, have their own properties and they differentiate from each other through their index value.

Adding controls with control array uses fewer resources than simply adding multiple. If we want to create a new instance of a control at run time, that control must be a member of a control array. With a control array, each new element inherits the common event procedures of the array. Thus it is useful for mainly two reasons:

1. To create and add a new menu item to a menu at run time.
2. To simplify code because common blocks of code can be used for all menu item.

Steps to create control array:

1. Place the desired control on the form. Let it be label1.
2. Select the control (label1) and click on copy command.
3. When you click on paste command on the form a dialog box will ask you "You already have a control named label1. Do you want to create a control array?"

4. Click on yes button.

Now you have two controls on the form having same name 'label1' and different index value. These index value starts from 0 and goes upward. Similarly you may add any number of controls in a control array.

### 3.8 SOME USEFUL EVENTS

As event refers to the occurrence of the user activity. Here are some common events that can occur to the controls drawn on a form.

1. **Click Event:** A control's Click event fires when the control is enabled and the user both presses and releases a mouse button while the mouse pointer is over the control. If the mouse pointer is over a disabled control or if the mouse cursor is over a blank area of the form, then the form receives the Click event. The Click event is easy to understand, because it represents a common user action that occurs dozens of times during a single session in any Windows-based application.
2. **DbClick Event:** The DbClick event occurs on a form or a control when the object is enabled, the mouse pointer is directly over the form or control, and the user clicks the mouse twice in rapid succession. Windows determines whether the user's two clicks represent a double-click or two single clicks. The user can access the Windows Control Panel to set the maximum time interval between two clicks for these two clicks to count as a double-click.
3. **MouseDown and MouseUp:** The MouseDown event fires when the user presses a mouse button over a control or form. Similarly the MouseUp event occurs when the user releases the mouse button over a control or form. If the user moves the mouse between the time the button was pressed and released, the same control will receive the MouseUp event.

During the MouseDown and MouseUp event procedures, you might want to know whether the left or right mouse button was pressed. You might also want to know whether or not one of the auxiliary keys (Shift, Alt, or Ctrl) also was depressed during the mouse event.

All of the foregoing information is available in following four parameters that the MouseUp or MouseDown event procedure receives from the system:

- (a) **Button As Integer:** This is a value representing which mouse button fired the event. The value of this parameter is either vbLeftButton, vbRightButton, or vbMiddleButton.
- (b) **Shift As Integer:** This parameter represents an integer that indicates whether an auxiliary key is pressed during the Mouse event. It contains a value of 0 (none), 1 (Shift), 2 (Ctrl), 4 (Alt), or the sum of any combination of those keys.
- (c) **X As Single:** This is the horizontal position of the mouse pointer from the internal left edge of the control or form receiving the event.
- (d) **Y As Single:** This is the vertical position of the mouse pointer from the internal top edge of the control or form receiving the event.

4. **KeyUp and KeyDown Event:** The KeyDown and KeyUp events happen when the user respectively presses and releases a key on the keyboard. Their event procedures take the following two parameters:
1. **KeyCode:** contains an integer code for the physical key that the user pressed. You can check for a particular key by comparing KeyCode with one of the special VB internal constants for physical key codes. Each constant name begins with the string “vbKey” followed by an obvious name for the key. Examples of vbKey constants would be vbKeyA, vbKeyW, vbKeyF1, vbKeyPgUp, and so forth.
  2. **Shift:** indicates if any of the three shift keys (Alt, Ctrl, or Shift) is pressed at the moment. This parameter works in the same way as the Shift parameter for the MouseDown and MouseUp event procedures. That is, Shift is an integer representing a bit mask. You can extract information concerning the state of each of the three control keys by ANDing the Shift parameter with one of the three VB constants for the control keys.
5. **KeyPress Event:** The KeyPress event happens after the KeyDown event but before the KeyUp event. It detects the ASCII value of the character generated by the pressed key. The KeyPress event procedure’s single parameter is KeyAscii. KeyAscii is an integer representing the ASCII value of the character generated by the user’s key press. For instance, if the user keys an uppercase “A,” then the KeyPress event fires and the KeyAscii parameter will have a value of 65 (since 65 is the ASCII code for uppercase “A”).

If you change the value of KeyAscii to 0, then the system will not see a keystroke, and you have in effect discarded the keystroke. This implies the following general technique for handling user keyboard input in the KeyPress event procedure:

- (a) Use the Chr function to convert KeyAscii to a character value.
  - (b) Manipulate or evaluate the character.
  - (c) Use the Asc function to convert the changed character back to its corresponding integer value, or determine the desired ASCII value in some other way.
  - (d) Assign the new ASCII value to the KeyAscii parameter.
6. **GotFocus:** It occurs when an object receives the focus, either by user action, such as tabbing or clicking the object, or by changing focus in code using the SetFocus method. A form receives the focus only when all visible controls are disabled.
7. **LostFocus:** Occurs when an object loses the focus, either by user action, such as tabbing or clicking another object, or by changing focus in code using the SetFocus method.

### 3.9 ACTIVEX CONTROL

An active X control is an extension to the VB toolbox. It is an interactive object that can reside on any form that supports OCX controls. ActiveX control exists in a separate file with a .ocx file name extension.

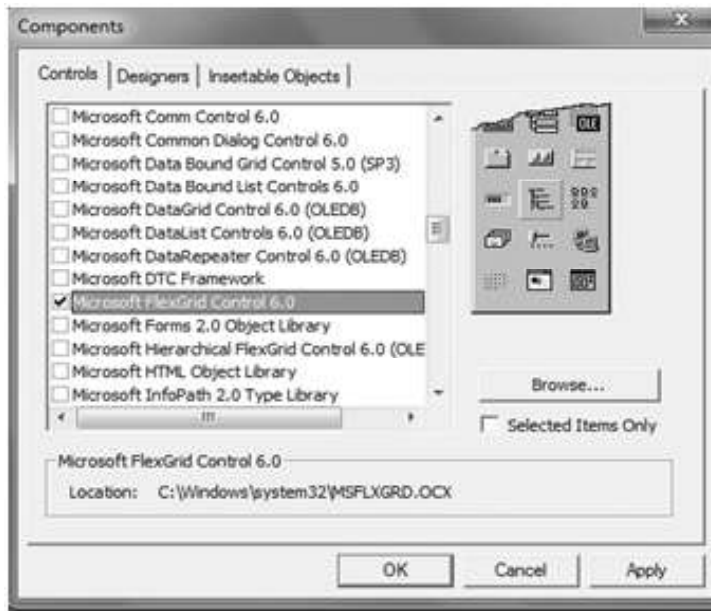
You can use these controls as any standard VB control. Programmers writing for ActiveX create components, self-sufficient programs that can be run by the Windows operating system. One of the main advantages of ActiveX components is that they can be re-used by many applications.

### 3.9.1 Adding and Removing ActiveX Controls

You can add or remove active controls to and from the toolbox using the following procedures.

#### Steps to add active control

1. Click on Project Menu — Components.
2. Select the checkbox next to the name of desired .ocx control and then click on OK.



Once a control is placed in the toolbox, you can use it as other intrinsic controls.

#### Steps to remove activeX control

1. Remove all instances of the control from the form. Delete any references to the control in the project's code.
2. From Project Menu — Components.  
Clear the checkbox next to the name of the .ocx control.  
Commonly used active control are:

### 3.9.2 Common Dialog Boxes

The Common Dialog Box Library contains a set of dialog boxes for performing common application tasks, such as opening files, choosing color values, and printing documents. The common dialog boxes allow you to implement a consistent approach to your application's user interface. It is stored in file COMDLG32.OCX.

You can use the common dialog control in your application by adding it to a form and setting its properties. The dialog displayed by the control is determined by the methods of the control. At run time, a dialog box is displayed or the Help engine is executed when the appropriate method is invoked. This control has the ability to display help by running the windows help file. At design time, the common dialog control is displayed as an icon on a form. This icon can't be sized.

The common dialog control allows you to display these commonly used dialog boxes:

- **Open:** The Open dialog box allows the user to specify a drive, a directory, a file name extension, and a file name.
- **Save As:** The Save As dialog box is identical to the Open dialog in appearance. It is used to save a file.
- **Color:** The Color dialog box allows the user to select a color from a palette or to create and select a custom color. At run time, when the user chooses a color and closes the dialog box, you use the Color property to get the selected color.
- **Font:** The Font dialog box allows the user to select a font by its size, color, and style.
- **Print:** The Print dialog box allows the user to specify how output should be printed. The user can specify a range of pages to be printed, a print quality, a number of copies, and so on. This dialog box also displays information about the currently installed printer and allows the user to configure or reinstall a new default printer.
- **Help:** The help dialog control allows you to display a Help file.

#### Steps to use the common dialog control

1. Add the common dialog control to the toolbox by selecting **Components** from the **Project** menu. Locate and select the control *Microsoft Common Dialog Control 6.0* in the **Controls** tabbed dialog, then click the **OK** button.

2. On the toolbox, click the **CommonDialog** control (  ) and draw it on a form.

When you draw a common dialog control on a form, it automatically resizes itself. Like the timer control, the common dialog control is invisible at run time.

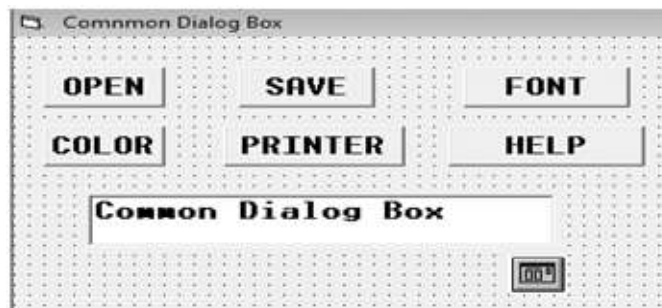
3. At run time, use the appropriate method, as listed in the following table, to display the desired dialog.

Method	Dialog displayed
ShowOpen	Open
ShowSave	Save As
ShowColor	Color
ShowFont	Font
ShowPrinter	Print
ShowHelp	Invokes Windows Help

When you call the equivalent method, the corresponding dialog box appears on the screen and execution of the program is suspended until the dialog box is closed.

**Example 10:** Design a form by using all the methods of commndialog box.

Design the form as follows



Control	Property
Form	Name – frmcdb Caption – Common Dialog box
CommandButton1	Name – cmdopen Caption – OPEN
CommandButton2	Name – cmdsave Caption – SAVE
CommandButton3	Name – cmdfont Caption – FONT
CommandButton4	Name – cmdcolor Caption – COLOR
CommandButton5	Name – cmdprint Caption – PRINT

CommandButton6	Name – cmdhelp Caption – HELP
Textbox1	Name – txtresult Text – [Empty]
CommonDialog1	Name – commondialog

Write the following code:

```
cmdprint
Private Sub cmdcolor_Click()
CommonDialog.ShowColor
txtresult.BackColor = CommonDialog.Color
End Sub

Private Sub cmdfont_Click()
CommonDialog.ShowFont
txtresult.Font = CommonDialog.FontName
End Sub

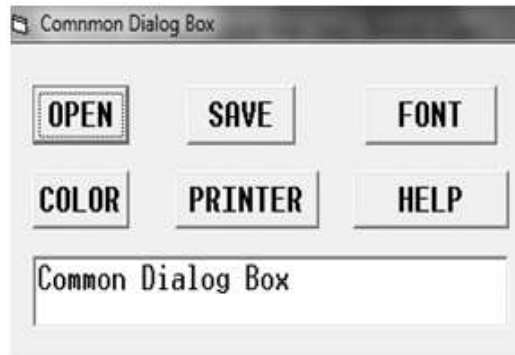
Private Sub cmdhelp_Click()
CommonDialog.ShowHelp
End Sub

Private Sub cmdopen_Click()
CommonDialog.Filter = "allfiles|*.*"
CommonDialog.ShowOpen
txtresult.Text = CommonDialog.FileName
End Sub

Private Sub cmdprint_Click()
CommonDialog.ShowPrinter
End Sub

Private Sub cmdsave_Click()
CommonDialog.ShowSave
End Sub
```

At run time when we click on the font button, it appear as



### 3.9.3 Microsoft Windows Common Controls

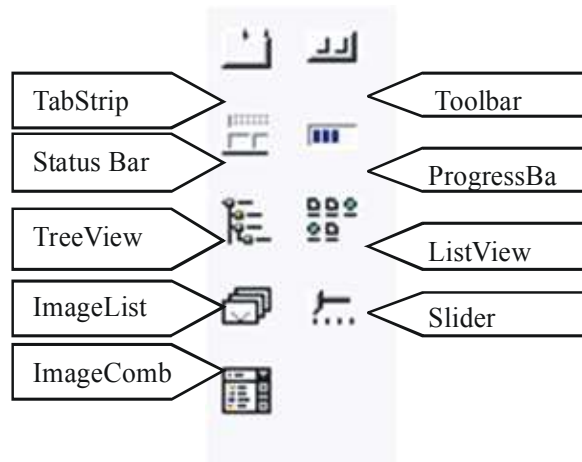
The Microsoft Windows Common Controls are a group of controls that let you add some of the same features to your program that windows uses – hence it calls common control. They are stored under file MSCOMCTL.OCX. It includes following control –



- TabStrip
- StatusBar
- TreeView
- ImageList
- ImageCombo
- Toolbar
- ProgressBar
- ListView
- Slider

### Steps to use the windows common control

1. Select **Components** from the **Project** menu. Locate and select the control *Microsoft Windows Common Control 6.0* in the **Controls** tabbed dialog, then click the **OK** button.
2. In the toolbox, you get following controls:



Select the desired control and place it on the form to use.

#### 3.9.3.1 Imagelist

The imagelist control contains a collection of images that can be used by other controls. Toolbars usually features icons that represent a function of the application. To display such images on the toolbar, you must associate an imagelist control with the toolbar control. This control is invisible at runtime.

Steps to add images to imagelist control

1. After adding windows common control in toolbox, click on **imagelist** and draw it on the form.
2. Select imagelist control. Right-click upon it and select **properties**. It will open **property page dialog box**.

3. In property page dialog box, select **image tab** and click on the **insert picture** button. This will open a dialog box to select the desired picture. Click on open, this will add image in image list. To add another image repeats the process.
4. Once you add all the required images in imagelist, click **ok**.

### 3.9.3.2 Toolbar

A toolbar control contains a collection of button objects used to create a toolbar that can associate with an application. Each button of toolbar corresponds to items in an application's menu, providing a graphic interface for user to access an application's most frequently used functions and commands.



Steps to create toolbar

1. After adding windows common control in toolbox, add **imagelist** control with desired number of pictures as described in imagelist section.
2. After this, add **toolbar** control on the form.
3. Right click on the toolbar and select **properties** from the shortcut menu. This will display the property page dialog box.
4. In **general** tab, select the **imagelist** property to the imagelist control name that is placed on the form. To give raising effect to the buttons of toolbar set **style** property to *tbrFlat*.
5. To create buttons, select **Button** tab.
6. Click at **insert button** to add a button in the toolbar. This will enable the button window.
7. After this add **index** number of image in the imagelist, that is to be inserted under image box.
8. Type a unique string under **key** box, a short description under **tooltip** box and to display text on the button adds **text** in caption box.
9. Repeat the process for adding more buttons. Then click on **ok**
10. To perform action by each button of the toolbar, add code for action in **button click** event of toolbar control. Key property is used to identify the button selected by the user.

### 3.9.3.3 Status Bar

A status bar control is a frame that can comprises several other panels which inform the user of the status of an application. The control can hold up to 16 frames.

**Steps to create status bar** with two panels. First panel will display a message and second panel will show time.

1. After adding windows common control in toolbox, add **Status Bar** control.

2. Right click on status bar and select **properties** from shortcut menu. This will open a **property window**.
3. Select **Panel** tab. To insert a panel click on **Insert Panel** button. Set panel **index** value to 1. Set **key** property to msg, **style** property to 0-sbrText and to display default text in this panel type some text in **text** property box.
4. To add another panel, again click on Insert Panel button. Set panel index to 2, key property to time, style property to 5-sbrTime. This will add time panel.
5. Once complete, click on **apply** and then **ok**.

### 3.10 OBJECT LINKING AND EMBEDDING (OLE )

OLE is a compound document standard developed by Microsoft Corporation. It enables you to create objects with one application and then link or embed them in a second application. Embedded objects retain their original format and links to the application that created them. For developers, it brought **OLE Control eXtension (OCX)**, a way to develop and use custom user interface elements. On a technical level, an OLE object is any object that implements the IOleObject interface, possibly along with a wide range of other interfaces, depending on the object's needs.

Visual Basic provides us the facility to link / open any different application from him by using object linking and embedding tool box,

**Steps to add PowerPoint/ Excel / word file** in your VB project

1. Open VB project.
2. Place OLE Tool on form and arrange its dimensions.
3. Click on SourceDoc property.
4. Select option create new object and then select desired object (.ppt,.exl, .doc)
5. To open a specific file select from file and click on browser. Select file and check on link.
6. At run time double click on the object.

### 3.11 DIALOG BOX

A dialog box is a window that is used to display or accept information. Dialog boxes are either modal or modeless. A *modal dialog box* does not allow the user to perform to work on other application until it is closed. A *modeless dialog box* allows the user to switch between the open dialog box and other application window. There are two type of dialog boxes- Input box and Message box.

#### 3.11.1 Input Box

Inputbox function displays a command prompt in a dialog box. It is a modal dialog box and ask the user to enter some data for processing. It is used by the following syntax:

**Syntax:**

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

**Where**

Prompt

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

**Title**

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

**DefaultResponse**

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

**XPos**

Optional. Numeric expression that specifies, in pixels, the distance of the left edge of the dialog box from the left edge of the screen. If you omit XPos and YPos, the dialog box is centered on the screen.

**YPos**

Optional. Numeric expression that specifies, in pixels, the distance of the upper edge of the dialog box from the top of the screen. If you omit XPos and YPos, the dialog box is centered on the screen

**Example:** The following coding displays as:

```
a = InputBox("Enter your name ", "Inputbox", "xxx")
```



### 3.11.2 MessageBox

The MsgBox function displays a message in a dialog box, waits for the user to click a button, and returns an Integer indicating which button the user clicked. It is used to get yes or no response from users, and to display brief messages, such as errors, warning or alerts in a dialog box. After reading the message user chooses a button to close.

**Syntax:**

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

**Where****Prompt**

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

**Buttons**

Optional. Numeric expression that is the sum of values specifying the number and type of buttons to display, the icon style to use, the identity of the default button, and the modality of the message box. If omitted, the default value for buttons is 0

**Title**

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

**Helpfile and context**

Both optional. These arguments are only applicable when a Help file has been set up to work with the application.

Usually the button argument can take any of the following value:

Constant	Value	Description
vbOKOnly	0	Display OK button only.
vbOKCancel	1	Display OK and Cancel buttons.
vbAbortRetryIgnore	2	Display Abort, Retry, and Ignore buttons.
vbYesNoCancel	3	Display Yes, No, and Cancel buttons.
vbYesNo	4	Display Yes and No buttons.
vbRetryCancel	5	Display Retry and Cancel buttons.



### LET US REVISE

- ✓ The VB toolbox contain tools that can used to draw controls on the form.
- ✓ A control can be drawn on the form by clicking at its icon on the toolbox and then dragging on the form.
- ✓ The control array found on the toolbox are called intrinsic control.
- ✓ A container control holds other control within it.
- ✓ The control name must begin with a letter and can contain digits, letters and underscore.
- ✓ An event procedure is a procedure containing code that is executed when an event occurs.
- ✓ The form act as a container for all the controls that make up the interface.
- ✓ A form can be loaded and unloaded through Load and Unload statement.
- ✓ In a form module, a form can be referred by keyword Me.
- ✓ A form can be shown by Show method and hide by Hide method.
- ✓ By setting default property to true, a command button can be designed as Default button.
- ✓ By setting cancel property to true, a command button can be designed as Cancel button.
- ✓ The label and textbox are text manipulation control.
- ✓ By setting multiline property to true, a textbox can hold multiple lines.
- ✓ The Maxlength property of a textbox specifies the maximum number of allowed characters.
- ✓ The Password property of a textbox used to hide the entered text.
- ✓ The controls available for choice selections are checkbox, option button, listbox, combobox and scroll bar.
- ✓ Additem and Removeitem method used to add or remove element from list.
- ✓ A listbox can be cleared through clear method.
- ✓ The Listcount property tells total number of items in the list.
- ✓ The shape and line are used for drawing graphical elements on the surface of a form.
- ✓ Timer is an invisible control used for taking actions at regular intervals.
- ✓ The three file system control – drivelistbox, dirlistbox and filelistbox, are used to explore the file system.
- ✓ A control array is a group of control that share common name and code.
- ✓ Some common events are click, dblclick, keypress, keydown & keyup, mouseup & mousedown, gotfocus and lostfocus.
- ✓ The OLE container control supports object linking and embedding.
- ✓ An active control is an interactive object that can reside on any form that supports OCX control.

- ✓ A common dialog box provides commonly used dialog box i.e open,save,font,color and help dialog box.
- ✓ The MS windows common control provides a set of 9 controls i.e tabstrip, toolbar, status bar, progressbar, treeview, listview, imagelist, slider, imagecombo.
- ✓ The toolbar control is associated with an imagelist control.
- ✓ A sttusbar is a control that can consists several panels.
- ✓ A dialog box is a window that is used to display information. There are two types of dialog box- message box and input box.
- ✓ Message box is used to display some information.
- ✓ Inputbox is used to accept some value from the user.

## Chapter-4

# Control Structure

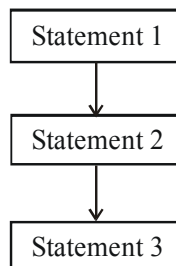
### 4.1 INTRODUCTION

Generally a program executes all its statement from beginning to end. But this is not true in every case. Most programs decide what to do in response to changing circumstances. These programs not only store data but also manipulate data. To perform these manipulations, programs need tools for performing repetitive actions and for making decisions. This can be done by providing statements in the program. Such statements are called program control statements. In this chapter we discuss such statements.

### 4.2 CONTROL FLOW

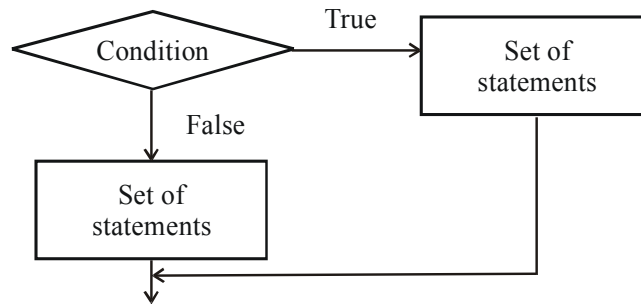
In a program, statements can be executed sequentially, selectively or iteratively. Thus, every program constructs sequence, selection and iteration.

- 1. Sequence:** The sequence construct means that the statements are executed sequentially. This represents the default flow of statements i.e the computer executes one line of code after other. Programming languages provide tools to control the flow of a program by skipping or repeating certain lines or groups of lines of code.

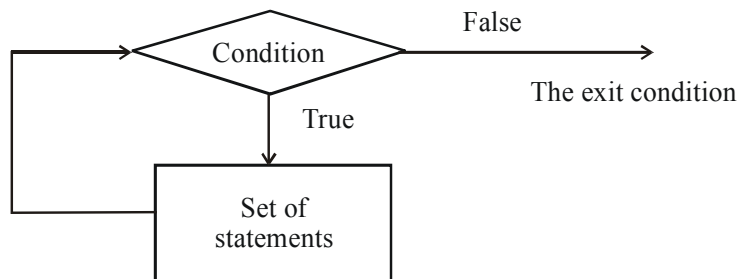


- 2. Selection:** The selection construct means the execution of statements depends upon a condition test. If a condition evaluates to true, a set of statements is followed otherwise another set of statements will follow. This construct is called decision construct as it helps in making decision which set of statements is to be executed.





- 3. Iteration:** The iteration construct means repetition of a set of statements depending upon a test condition. Till the time a condition is true, a set of statements are repeated again and again. As soon as the condition becomes false, the repetition stops. This is also called looping construct. The set of statements that are repeated again and again is called body of loop and the condition on which execution stop is called exit/test condition.



### 4.3 DECISION STRUCTURES

The decision structure allows to choose the set of statements for execution depending upon the test condition's. VB provides two type of selection construct –

1. If Statement
2. Select Case Statement

#### 4.3.1 If...Then Statement

The If...Then statement tests a particular condition and executes the followed set of statements.

##### Syntax

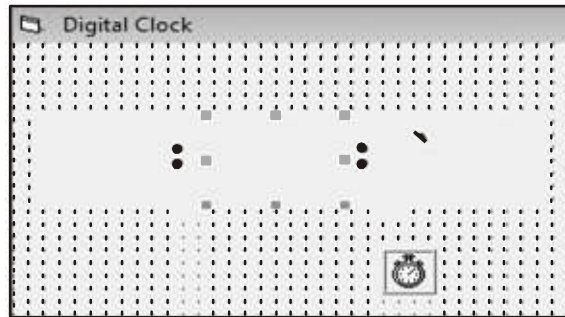
```

If condition
Then statements
End If
  
```

If the condition evaluates to true, the statement executes otherwise ignored.

**Example 1:** Design a digital clock.

Design the following form



Control	Property
Form	Name – frmclock
Lable 1	Caption – Digital Clock
Lable 2	Name – lblhr
Lable 3	Caption – 00
Lable 4	Name – lable2
Lable 5	Caption – :
Timer 1	Name – lblmin
	Caption – 00
	Name – lable4
	Caption – :
	Name – lblsec
	Caption – 00
	Locked – True
	Name – Timer
	Interval – 1000

Write the following code:

```

Timer
Private Sub Timer_Timer()
Dim hr As Byte, min As Byte, sec As Byte
hr = Int(lblhr.Caption)
min = Int(lblmin.Caption)
sec = Int(lblsec.Caption)
sec = sec + 1
If sec = 60 Then
sec = 0
min = min + 1
End If
If min = 60 Then
min = 0
hr = hr + 1
End If
If hr = 24 Then
hr = 0
End If
lblhr.Caption = Format(hr, "00")
lblmin.Caption = Format(min, "00")
lblsec.Caption = Format(sec, "00")
End Sub

```

At run time it appear as



If...Then...Else Statement

This form of if allows for either – or kind of conditions. Its syntax is

**Syntax**

**If** condition **Then**

Statements 1

**Else**

Statements 2

**End If**

If the condition evaluates to true the statements 1 are executed otherwise statements 2 are executed.

**Example 2:** WAP that offers various food items and a mode of payment to select. It then displays the total payable amount in a message box.

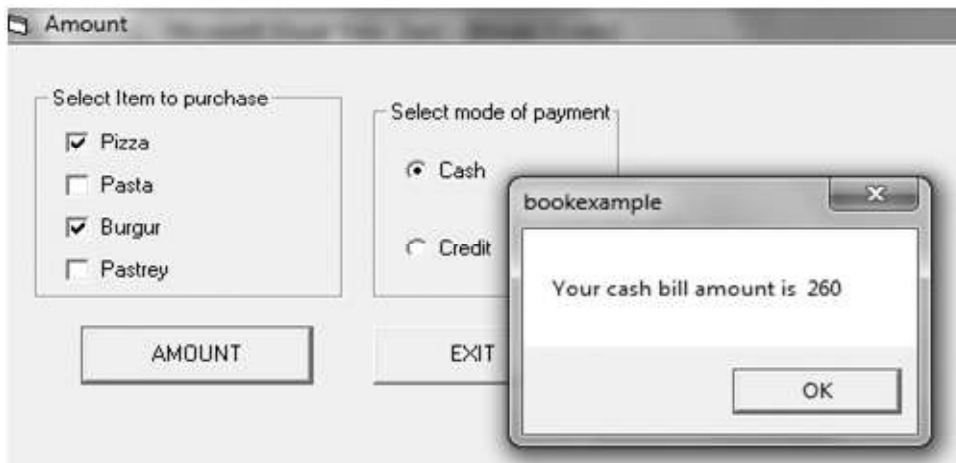
Design the following form

Control	Property
Form	Name – frmamount
Frame 1	Caption – Amoount
Check 1	Name – Frame
Check 2	Caption – Select item to purchase
Check 2	Name – chkpizza
Check 3	Caption – Pizza
Check 4	Name – chkpasta
Frame 2	Caption – Pasta
Option 1	Name – chkburgur
Option 2	Caption – Burgur
Command 1	Name – chkpastrey
Command 2	Caption – Pastrey
	Name – frame 2
	Caption – Select mode of payment
	Name – optcash
	Caption – Cash
	Name – optcredit
	Caption – Credit
	Name – cmdamt
	Caption – Amount
	Name – cmdexit
	Caption – Exit

Write the following code

```
cmdamt Click  
  
Private Sub cmdamt_Click()  
    Dim amt As Integer  
    amt = 0  
    If chkpizza.Value = 1 Then  
        amt = amt + 200  
    End If  
    If chkpasta.Value = 1 Then  
        amt = amt + 150  
    End If  
    If chkburgur.Value = 1 Then  
        amt = amt + 60  
    End If  
    If chkpastrey.Value = 1 Then  
        amt = amt + 50  
    End If  
    If optcash.Value = True Then  
        MsgBox (" Your cash bill amount is " + Str(amt))  
    Else  
        MsgBox (" Your credit bill amount is " + Str(amt))  
    End If  
End Sub  
  
Private Sub cmdexit_Click()  
End  
End Sub
```

At run time it appear as



#### If.....Then.....Else If Statement

This is last statement in if family of statements. This statement allows you to test a number of exclusive cases and only executes one set of lines of code for the case that is true first. The syntax for using this statement is:

#### Syntax:

```

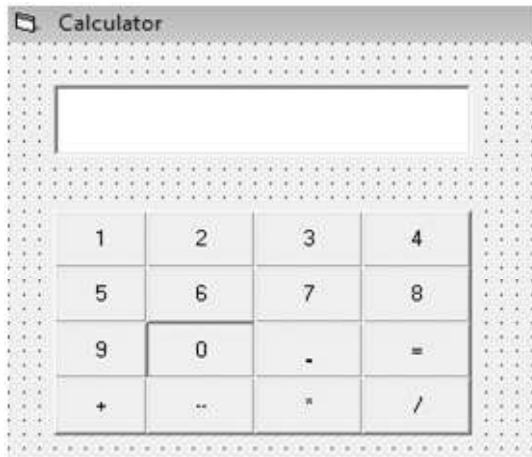
If condition1 Then
    statements 1
Elseif condition2 then
    statements 2
Elseif condition3 then
    Statements 3
    .
    .
    .
End If

```

If first condition is true then statement 1 is executed and then flow jumps to the endif statement. If condition1 is false then next condition is evaluated. If it is true the statement2 is executed and flow jumps to the endif. This keeps going till an else statement or endif statement are executed.

**Example 3:** WAP to design a basic mathematical calculator by using control array.

Design the following form



Control	Property
Form	Name – frmcal
	Caption – Calculator
Command 1	Create control array of command 1 to create numberpad of calculator Name – Digit Caption – 0,1,2.....9 Index – 0,1,2.....9
Command 2	Name – cmddot Caption – .
Command 3	Name – cmdequal Caption – =
Command 4	Name – cmdadd Caption – +
Command 5	Name – cmdsub Caption – —
Command 6	Name – cmdmul Caption – *
Command 7	Name – cmddiv Caption – /
Text1	Name – txtdisplay Text – nil Locked - True

Write the following code

(General)	(Declarations)
<pre>Dim operand1 As Integer Dim operand2 As Integer Dim operator As String Dim cleardisplay As Boolean</pre>	
<pre>Private Sub cmdadd_Click() operand1 = Val(txtdisplay.Text) operator = "+" txtdisplay.Text = "" End Sub</pre>	
<pre>Private Sub cmddiv_Click() operand1 = Val(txtdisplay.Text) operator = "/" txtdisplay.Text = "" End Sub</pre>	
<pre>Private Sub cmddot_Click() If InStr(txtdisplay.Text, ".") Then Exit Sub Else txtdisplay.Text = txtdisplay.Text + "." End Sub</pre>	
<pre>Private Sub cmdequal_Click() Dim ans As Double operand2 = Val(txtdisplay.Text) If operator = "+" Then ans = operand1 + operand2 ElseIf operator = "-" Then ans = operand1 - operand2 ElseIf operator = "*" Then ans = operand1 * operand2 Else ans = operand1 / operand2 End If txtdisplay.Text = ans End Sub</pre>	

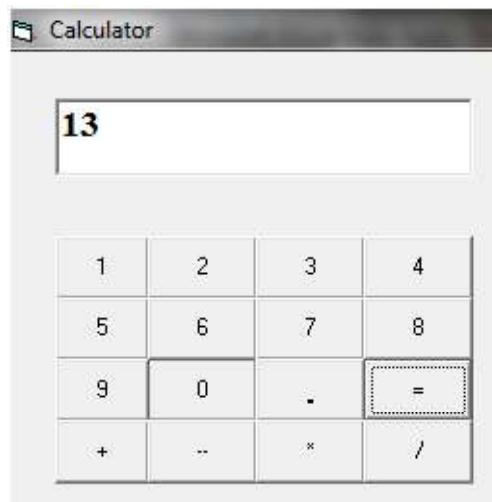


```
Private Sub cmdmul_Click()  
operand1 = Val(txtdisplay.Text)  
operator = "*"   
txtdisplay.Text = ""  
End Sub
```

```
Private Sub cmdsub_Click()  
operand1 = Val(txtdisplay.Text)  
operator = "-"   
txtdisplay.Text = ""  
End Sub
```

```
Private Sub digit_Click(Index As Integer)  
If cleardisplay Then  
    txtdisplay.Text = " "  
    cleardisplay = False  
End If  
txtdisplay.Text = txtdisplay.Text + digit(Index).Caption  
End Sub
```

At runtime to calculate  $5+8$  , it appear as



### 4.3.2 Select...Case Statement

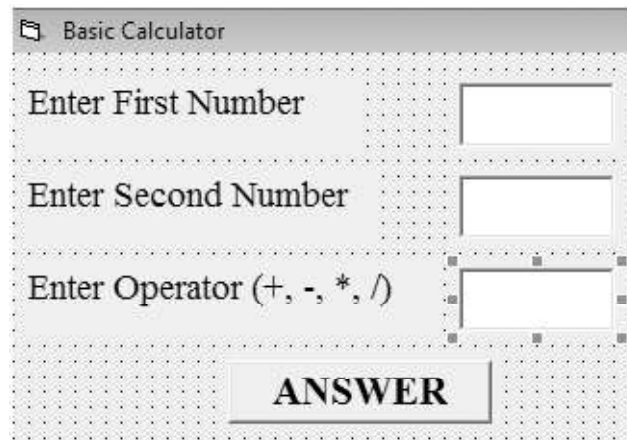
The if statement is good for data comparisons in case where two or more relational tests must be made. But when you compare too many conditions, then nesting becomes difficult to read and understand. VB supports a statement called select...case that handle such multiple choice conditions better than if-else. The select case can be used when multiple if statements become messy and difficult to read. It comes in three format.

**Format 1:**

```
Select Case expression
  Case value
    Statement
  Case value
    Statement
  .
  .
  .
  Case else
    Statement
End select
```

**Example 4:** WAP to accept two numbers and a basic mathematical operator to perform the user's choice action.

Design the following form



The image shows a screenshot of a Windows-style application window titled "Basic Calculator". The window has a dotted background and contains three input fields on the left and one output field on the right. The input fields are labeled "Enter First Number", "Enter Second Number", and "Enter Operator (+, -, \*, /)". The output field is labeled "ANSWER".

Control	Property
Form	Name – frmmath
Lable 1	Caption – Basic Calculator
Lable 2	Name – lblfirst
Lable 3	Caption – Enter First Number
Text 1	Name – lblsec
Text 2	Caption – Enter Second Number
Text 3	Name – lblop
Command 1	Caption – Enter Operator (+, -, *, /)
	Name – txtfirst
	Text – nil
	Name – txtsec
	Text – nil
	Name – txtop
	Text – nil
	Name – cmdans
	Caption – ANSWER

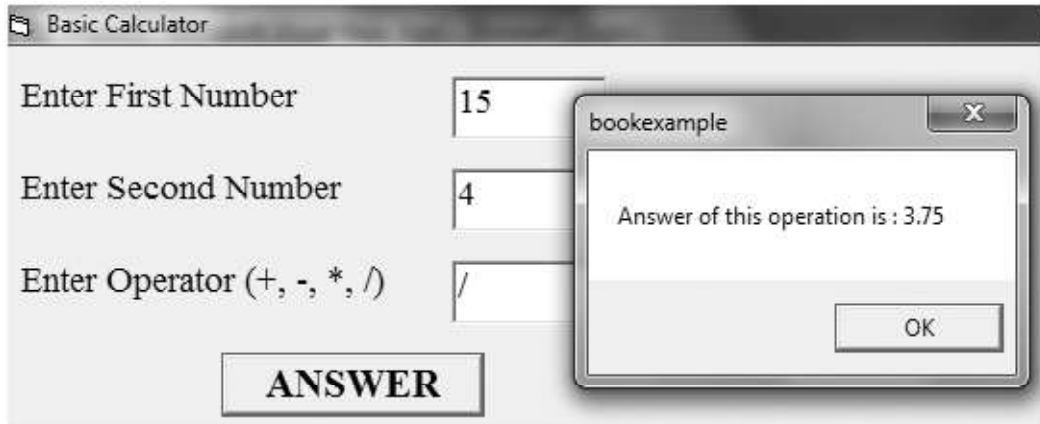
Write the following code

```

cmdans Click
Private Sub cmdans_Click()
Dim first As Integer, second As Integer
Dim ans As Double
Dim operator As String
first = Val(txtfirst.Text)
second = Val(txtsec.Text)
operator = txttop.Text
Select Case operator
Case "+"
ans = first + second
Case "-"
ans = first - second
Case "*"
ans = first * second
Case "/"
ans = first / second
Case Else
MsgBox (" Check the operator")
End Select
MsgBox (" Answer of this operation is : " & ans)
End Sub

```

At runtime it appear as



**Format 2:** This format specify the relation of values with the case expression.

**Select Case** expression

**Case** Is relation

Statement

**Case** Is relation

Statement

•

•

•

Case else

Statement

End select

Here relation can be tested to perform against expression.

**Example 5:** WAP to find overall percentage and division of a student depends upon his marks obtained from 100 in three subjects .

Design the following form

The image shows a Windows form titled "Result". It has a standard Windows window border. Inside the form, there are five labels on the left side: "Student's Name", "Marks obtained in", "Science", "Maths", and "English". To the right of each label is a text input field. At the bottom center of the form, there is a button labeled "RESULT".

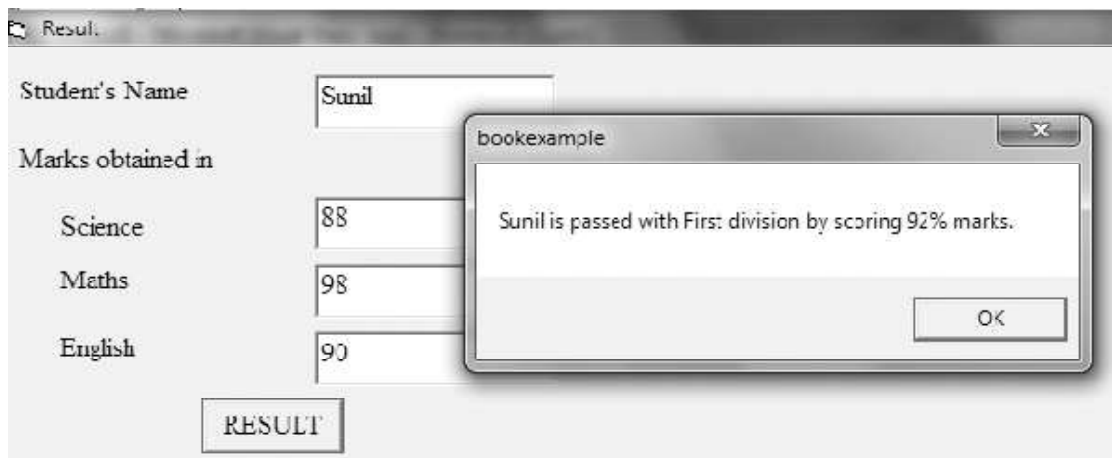
Control	Property
Form	Name – frmresult
Lable 1	Caption – Result
Lable 2	Name – lblname
Lable 3	Caption – Student's Name
Lable 4	Name – lblscience
Text 1	Caption – Science
Text 2	Name – lblmath
Text 3	Caption – Maths
Text 4	Name – lbleng
Command 1	Caption – English
	Name – txtname
	Text – nil
	Name – txtscience
	Text – nil
	Name – txtmath
	Text – nil
	Name – txteng
	Text – nil
	Name – cmdans
	Caption – ANSWER

Write the following code

```
cmdresult
Click

Private Sub cmdresult_Click()
    Dim sci As Integer, math As Integer, eng As Integer
    Dim percentage As Double, division As Double
    sci = Val(txtscience.Text)
    math = Val(txtmaths.Text)
    eng = Val(txteng.Text)
    percentage = ((sci + math + eng) * 100) / 300
    Select Case percentage
    Case Is > 59
        MsgBox (txtname.Text & " is passed with First division by scoring " & percentage & "% marks.")
    Case Is > 49
        MsgBox (txtname.Text & " is passed with Second division by scoring " & percentage & "% marks.")
    Case Is > 33
        MsgBox (txtname.Text & " is passed with First division by scoring " & percentage & "% marks.")
    Case Else
        MsgBox (txtname.Text & " is failed.")
    End Select
End Sub
```

At runtime it appear as



**Format 3:** This format specifies the ranges in the case value.

**Select** Case expression

**Case** exp1 to exp2

Statement

**Case** exp1 to exp2

Statement

•

•

•

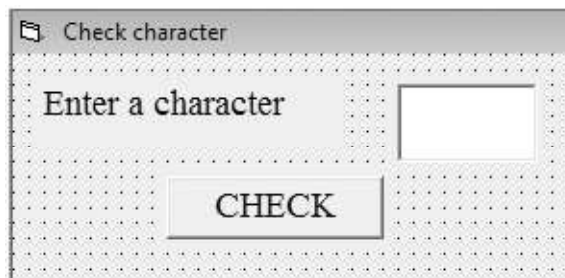
Case else

Statement

End select

**Example 6:** WAP to obtain a character from user and check whether it is in upper case, lower case, digit or symbol.

Design the following form:

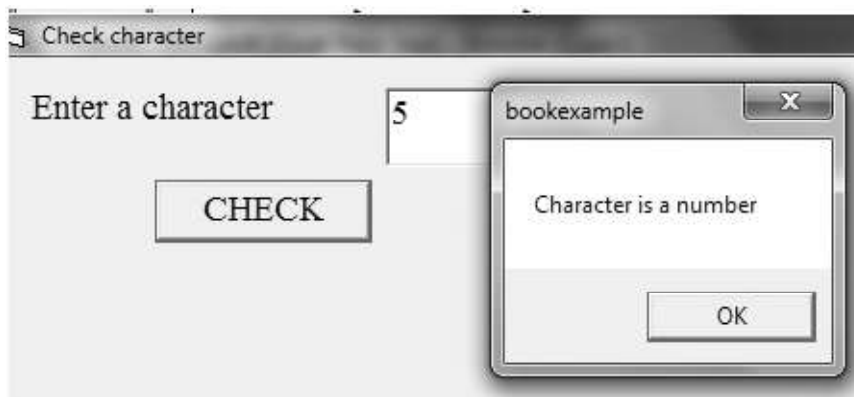


Control	Property
Form	Name – frmchar
Label 1	Caption – Check Character
Text 1	Name – lblchar
Command 1	Caption – Enter a character
	Name – txtchar
	Text – nil
	Name – cmdcheck
	Caption – CHECK

Write the following code

```
cmdcheck
Private Sub cmdcheck_Click Object
Dim char As String
char = txtchar.Text
Select Case Asc(char)
Case Asc("A") To Asc("Z")
MsgBox (" Character is in upper case")
Case Asc("a") To Asc("z")
MsgBox (" Character is in lower case")
Case Asc("0") To Asc("9")
MsgBox (" Character is a number")
Case Else
MsgBox (" Character is a symbol")
End Select
End Sub
```

At runtime it appear as



#### 4.4 LOOPING STRUCTURE

Looping structures are the statements that execute instructions repeatedly. VB provides two looping structures:

1. **Sentinel controlled loop structure:** It iterates the statements until a sentinel / terminating value is obtained.



**2. Counter controlled loop structure:** It requires a counter variable which get incremented in every iteration. Loop terminates when the counter reaches a particular value.

There are three types of looping structure

1. For....Next
2. Do loop
3. While ....wend

#### 4.4.1 For....Next

This loop is used when you know exactly how many times the code must be repeated. The general syntax is

**Syntax:**

**For** variable = <start\_value> **To** <End\_value> **Step** [<increment\_value>]

Statements

**Next** variable

The number of iterations is determined by start\_value and end\_value, where both are integers. Initially the variable has start\_value, the variable is get incremented by the increment\_value after every iteration. When the variable value exceeds the end value the loop terminates. The step clause is optional and by default, the increment value is equal to 1 .

**Example 7:** WAP to print Fibonacci series.

Design the following form

Control	Property
Form	Name – frmseries
Lable 1	Caption – Fibonacci Series
Text 1	Name – lblno
Command 1	Caption – Enter number of terms to be displayed
	Name – txtno
	Text – nil
	Name – cmdprint
	Caption – Print Series

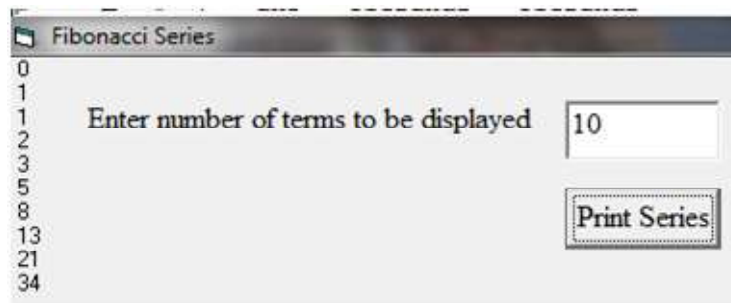
Write the following code

```

cmdprint
Click
Private Sub cmdprint_Click()
Dim first As Integer, second As Integer, number As Integer
If txtno.Text = "" Then
MsgBox ("Enter number of terms")
Else
first = 0
second = 1
Print first
Print second
For i = 1 To (Val(txtno.Text) - 2)
number = first + second
Print number
first = second
second = number
Next i
End If
End Sub

```

At runtime it appear as



### Nested for loop

A loop structure placed inside another loop structure is called nested loop. Here the inner loop must end before the outer loop. The nested for loop is used to display data in the form of rows and columns. It can be write as follows

```
For i = 1 to 10
```

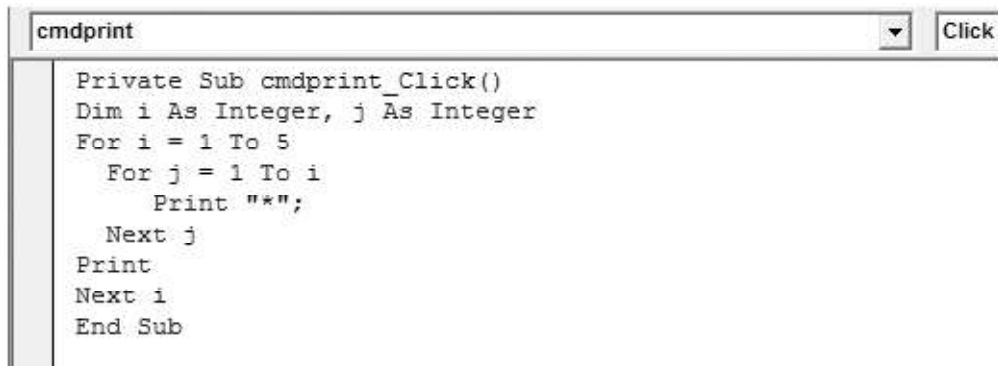
```
For j = 1 to 5
```

```
Print i,j
Next j
Next i
```

**Example 8:** WAP to print the following pattern

```
*
**
***
****
*****
```

Write the following code on click event of Print command button.



```
cmdprint
Private Sub cmdprint_Click()
Dim i As Integer, j As Integer
For i = 1 To 5
  For j = 1 To i
    Print "*";
  Next j
  Print
Next i
End Sub
```

“;“ ensures that successive \* is placed next to previous \*

Sometimes, you may need to exit from a loop even though its terminating condition is not reached. In such cases use **Exit Do** statement at the place from where you want to exit.

#### 4.4.2 Do Loop Structure

A Do loop repeats statements in its body as long as the condition evaluates to true. And a Do ...Until loop repeats itself as long as the given condition evaluates to false. There are four ways to use do...loop statement:

##### Do While.....Loop

Do While...Loop is an entry controlled loop i.e there is control over the entry into the loop. You should use this loop when you want to test a condition first and if the condition evaluates to true then only you want to repeat the statements. The general form of Do While ...Loop is

**Syntax:****Do While** Condition

Statement

Loop

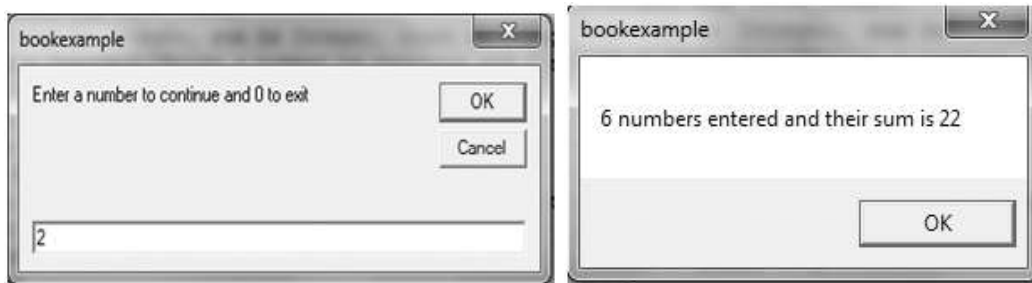
The condition is evaluated first if it is false, the body will never be executed. If it is true, the body will execute and again check the condition before executing next time.

**Example 9:** WAP to count the numbers entered and calculate their sum. The loop should terminate when 0 is entered.

Write the following code at Form\_Load event

```
Form
Load
Private Sub Form_Load()
Dim num As Integer, sum As Integer, count As Integer
num = InputBox("Enter a number ")
count = 0
sum = 0
sum = sum + num
Do While num <> 0
num = InputBox("Enter a number to continue and 0 to exit")
sum = sum + num
count = count + 1
Loop
MsgBox (count & " numbers entered and their sum is " & sum)
End Sub
```

At run time it appear as



## Do...Loop While

In Do..Loop while, first the body of loop is executed then condition is evaluated. If condition is true then it repeats again otherwise terminated. In this loop the body is executed at least one time. This is an exit controlled loop.

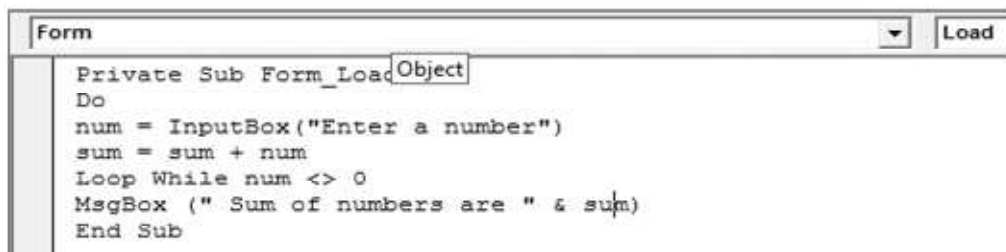
### Syntax:

```
Do
    Statement
```

```
Loop While Condition
```

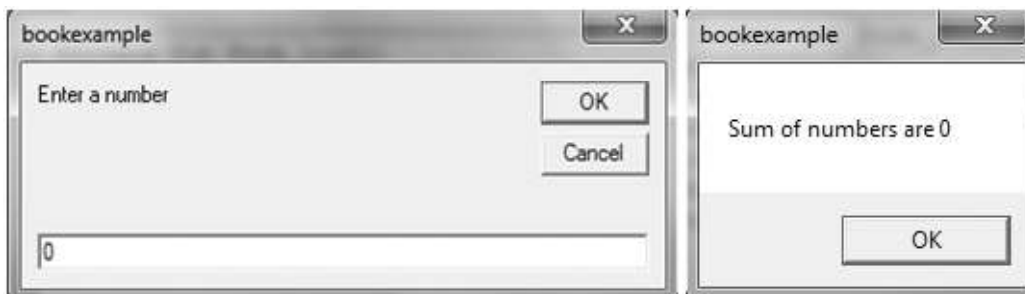
**Example 10:** WAP to calculate the sum of numbers entered by user. The loop should terminate when 0 is entered.

Write the following code at Form\_Load event



```
Private Sub Form_Load(Object)
Do
num = InputBox("Enter a number")
sum = sum + num
Loop While num <> 0
MsgBox (" Sum of numbers are " & sum)
End Sub
```

The above code will execute at least once even if condition is not evaluated. At runtime even if we write 0 as first number it displays the result as follows



## Do Until....Loop

This is similar to Do While loop, but here the loop repeats as long as test condition is false. As soon as the condition becomes true, the loop will terminate.

**Syntax:****Do Until** condition

Statement

Loop

The following code repeats as long as number is not 0.

Do Until num=0

Num = inputbox(" Enter a number?")

Sum = sum+num

Loop

**Do....Loop Until**

This loop is similar to above loop but only difference is that it evaluates the condition at the end of loop. The general form of this loop is

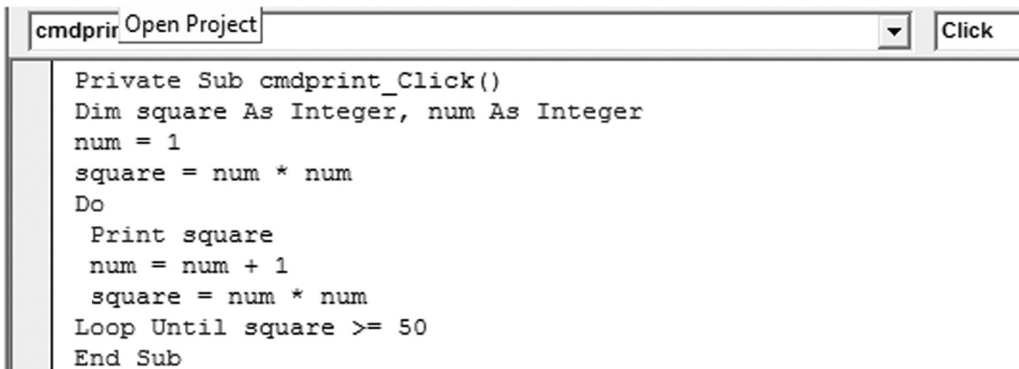
**Syntax:****Do**

Statement

**Loop Until** condition

**Example 11:** WAP to print first n squares on number where square is less than 50.

Write the following code at click event of Print command button



```
cmdprint_Open Project [v] Click
Private Sub cmdprint_Click()
Dim square As Integer, num As Integer
num = 1
square = num * num
Do
Print square
num = num + 1
square = num * num
Loop Until square >= 50
End Sub
```

At runtime it gives the following result



Sometimes, you may need to exit from a loop even though its terminating condition is not reached. In such cases use **Exit Do** statement at the place from where you want to exit.

#### 4.4.3 While ...Wend

The while...wend loop is functionally equivalent to the Do While ...Loop. It uses the following

**syntax:**

```
While condition
  Statements
Wend
```

The following code repeats as long as number is not 0.

```
While num<>0
  Num=inputbox(" Enter a number")
  Sum = sum+num
Wend
```

#### 4.4.4 Problems with loop

While using loop two types of problems are often encountered:

- 1. Infinite loop:** These loops are written in such a way that it never reach at their terminating condition. It keeps repeating itself endlessly that is why it is also called endless loop. It is a common programming error which is caused by neglecting increment or decrement counter. You can use **Break** or **Ctrl+Break** button to suspend execution and correct the code.
- 2. Invalid initial and terminal conditions:** A program also not able to perform well if it doesn't get proper initial and terminal value. It mostly done with arrays.

#### 4.5 ARRAYS

When you work with a single item, you only need to use one variable. However, if you have a list of items which are of similar type to deal with, you need to declare an array of variables instead of using a variable for each item. For example, if we need to enter one hundred names, you might have difficulty in declaring 100 different names, this is a waste of time and efforts. So, instead of declar-

ing one hundred different variables, you need to declare only one array. Thus, *an array is a collection of variables, all with the same data type and name.*

An array stores in a consecutive group of memory locations having the same name and the same data type. To refer to a particular location or element in the array, you specify the array name and the array element position number. The Individual elements of an array are identified using an index value. The index value of first element of an array is always be 0. 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.

#### 4.5.1 Dimension of an Array

An array can be one dimensional or multidimensional. One dimensional array is like a list of items or a table that consists of one row of items or one column of items. The format for a one dimensional array is **ArrayName(x)**.

Student	Name(0)	Name(1)	Name(2)	Name(3)	Name(4)	Name(5)
Name						

A two-dimensional array will be a table of items that make up of rows and columns. To identify a particular table element, we must specify two indexes: The first (by convention) identifies the element's row and the second (by convention) identifies the element's column. Normally it is sufficient to use one dimensional and two dimensional arrays, you only need to use higher dimensional arrays if you need with engineering problems or even some accounting problems. VB supports maximum 60 dimensional arrays. The format for a two dimensional array is **ArrayName(x,y)**.

Name(0,0)	Name(0,1)	Name(0,2)	Name(0,3)
Name(1,0)	Name(1,1)	Name(1,2)	Name(1,3)
Name(2,0)	Name(2,2)	Name(2,2)	Name(2,3)

#### 4.5.2 Declaring Arrays

Arrays occupy space in memory. The programmer specifies the array type and the number of elements required by the array so that the compiler may reserve the appropriate amount of memory. Arrays may be declared as Public or Private by using keywords Dim or Static. Array must be declared explicitly with keyword "As".

There are two types of arrays in Visual Basic namely:

1. **Fixed-size array:** The size of array always remains the same as declared. It doesn't change its size during the program execution.
2. **Dynamic array:** The size of the array can be changed at the run time i.e size changes during the program execution.



### Fixed-sized Arrays

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

### Declaring a fixed-array

```
Dim numbers(6) As Integer
```

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

If we want to specify the lower limit, then the parentheses should include both the lower and upper limit along with the To keyword as

```
Dim numbers (1 To 6 ) As Integer
```

In the above statement, an array of 6 elements is declared but with indexes running from 1 to 6 instead of 0 to 5.

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

```
Public numbers(5) As Integer
```

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

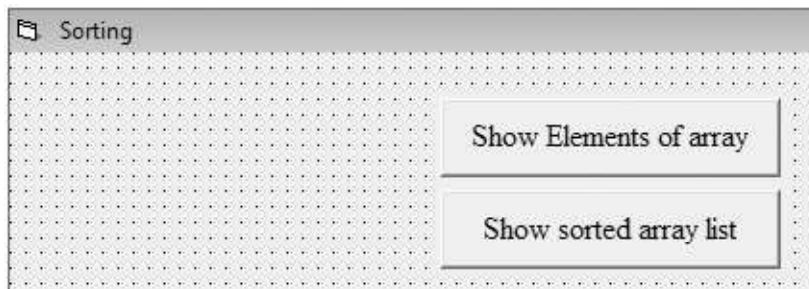
```
Dim AvgMarks ( 50, 50)
```

It is also possible to define the lower limits for one or both the dimensions as for fixed size arrays.

```
Dim Marks ( 101 To 200, 1 To 100)
```

**Example 12:** WAP to sort numbers in ascending order.

Design the following form:



Control	Property
Form	Name – frm\$array
	Caption – Soring
Command 1	Name – cmdlist
	Caption – Show Elements of array
Command 2	Name – cmdsort
	Caption – Show sorted array list

Write the following code

```

cmdlist Click
Dim list(20) As Integer
Dim num As Integer, arr As Integer, temp As Integer
Dim i As Integer, j As Integer, k As Integer

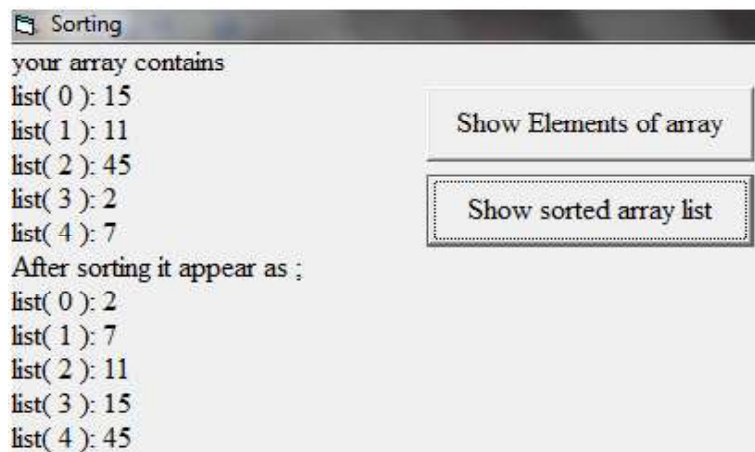
Private Sub cmdlist_Click()
arr = 0
Print "your array contains"
For k = 0 To num - 1
Print "list"; "("; arr; "):"; list(k)
arr = arr + 1
Next k
End Sub

Private Sub cmdsort_Click()
For i = 0 To num - 1
For j = i + 1 To num - 1
If list(i) > list(j) Then
temp = list(i)
list(i) = list(j)
list(j) = temp
End If
Next j
Next i
arr = 0
Print "After sorting it appear as ;"
For k = 0 To num - 1
Print "list"; "("; arr; "):"; list(k)
arr = arr + 1
Next k
End Sub

```

```
Private Sub Form_Load()  
num = InputBox("Initialize the size of array")  
For i = 0 To num - 1  
list(i) = InputBox("Enter element in your array")  
Next i  
End Sub
```

At runtime it appear as



### 4.5.3 Static and Dynamic Arrays

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

```
Dim Names(100) As String           'This is a static array.
```

Most programs don't use static arrays because programmers rarely know at compile time how many items you need and also because static arrays can't be resized during execution. Both these issues are solved by dynamic arrays. You declare and create dynamic arrays in two distinct steps. Initially you declare the array to account for its visibility using a Dim command with an empty pair of brackets.

```
Dim Customers() As String
```

Then you create the array when you actually need it, using a ReDim statement:

```
ReDim Customer(1000) As String
```

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

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

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

Finally, you can destroy an array using the Erase statement. If the array is dynamic, Visual Basic releases the memory allocated for its elements (and you can't read or write them any longer); if the array is static, its elements are set to 0 or to empty strings.

### 4.5.4 Arrays within UDTs

UDT(User Defined Type) structures can include both static and dynamic arrays. Here's a sample structure that contains both types:

```
Type MyUDT
    StaticArr(100) As Long
    DynamicArr() As Long
End Type

...

Dim udt As MyUDT
ReDim udt.DynamicArr(100) As Long
udt.StaticArr(1) = 1234
```

The memory needed by a static array is allocated within the UDT structure; Dynamic arrays are advantageous when each individual UDT variable might host a different number of array items.

#### 4.5.5 Array within another array

It is possible to create a variant array and populate it with other arrays of different data types. The following code creates two arrays, one containing integers and other strings. It then declares a third variant array and populates it with the integer and string array.

```
Private Sub command1_Click()  
    Dim counterA(5) As Integer  
    Dim intx As Integer  
    For intx = 0 To 4  
        counterA(intx) = 5  
    Next intx  
    Dim counterB(5) As String  
    For intx = 0 To 4  
        counterB(intx) = "hello"  
    Next intx  
    Dim arrx(2) As Variant  
    arrx(1) = counterA()  
    arrx(2) = counterB()  
    MsgBox arrx(1)(2)  
    MsgBox arrx(2)(3)  
End Sub
```

### LET US REVISE

- ✓ Every programming language provides constructor to support sequence, selection or iteration.
- ✓ Sequence construct means the statements are being executed sequentially.
- ✓ Selection construct means the execution depends upon a condition test.
- ✓ Iteration means repetition of a set of statements depending upon a test condition.
- ✓ If statement and select case are two selections construct.
- ✓ An if statement can have another if statement inside it, called nested if.
- ✓ The select case handles multiple choice conditions better than if else.
- ✓ Looping structure are statements that execute repeatedly.
- ✓ For..Next, Do loop and while..wend provides loop structure.

- ✓ Exit do, Exit for can be used to exit from loop.
- ✓ Array is a group of homogeneous element.
- ✓ Arrays can either be fixed size i.e static or dynamic.
- ✓ Redim statement is used to specify number of elements in dynamic array.
- ✓ To preserve array element, Preserve keyword is used with Redim statement.

## Chapter-5

# Procedures, Functions and Modules

### 5.1 INTRODUCTION

In most programming languages, large programs are generally avoided because it is difficult to manage a single list of instructions. Thus, a large program is broken down into various small units which perform a complete task. Such units are called procedures. There is various types of procedures. We will discuss those in this chapter. Along with procedures, we also discuss modules and their type.

### 5.2 PROCEDURES

A procedure is a block of Visual Basic statements enclosed by a declaration statement (**Function, Sub, Operator, Get, Set**) and a matching **End** declaration which performs a well defined task.. All executable statements in Visual Basic must be within some procedure.

Benefits of procedures:

1. They allow you to break up large or complex tasks into understandable pieces. Thus it makes program easier to handle.
2. Procedures are useful for condensing repeated operations such as the frequently used calculations, text and control manipulation etc.
3. They save you programming time by allowing you to reuse the same code over and over in your program.
4. They allow you to modularize your program so that if the specification of your program changes, you can minimize the number of lines that you must modify.
5. It is easier to debug a program a program with procedures, which breaks a program into discrete logical limits.
6. After you develop procedures for use in one program, you can use them in other programs, often with little or no modification. This helps you avoid code duplication.

Visual Basic uses following types of procedures:

- (i) Sub Procedures perform actions but do not return a value to the calling code.

- (ii) Function Procedures return a value to the calling code. They can perform other actions before returning.
- (iii) Property Procedures return and assign values of properties on objects or modules.

### 5.3 SUB PROCEDURES (SUB-ROUTINES)

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.

You can define a Sub procedure 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.

A Sub procedure can take arguments, such as constants, variables, or expressions, which are passed to it by the calling code.

There are two type of sub procedures in VB-

- (i) **General Procedures:** A general procedure is one that you create for your own specific purpose. It tells application how to perform a specific task and it must be specifically invoked by the application. They are Public by default, which means you can call them from anywhere in your application.
- (ii) **Event Procedures:** An event procedure is a procedure block that contains the control's actual name, an underscore(\_), and the event name. An event procedure is procedures associated with a specific event of an object and are named in a way that indicates the object and the event clearly. When an object in VB recognizes that an event has occurred, it automatically invokes the event procedure using the name corresponding to the event. Event Procedures acquire the declarations as Private by default.

#### 5.3.1 Declaring Syntax

The syntax for declaring a Sub procedure is as follows:

```
[ modifiers ] Sub Procedurename [( parameterlist )]  
    'Statements of the Sub procedure.  
End Sub
```

The *modifiers* can specify access level i.e public, private or protected. *Parameters* or arguments are the values that are passed to procedures to accomplish its specific task. The arguments that are to be passed to a procedure are defined at the time of procedure creation. Each argument



looks like a variable declaration and act like a variable in a procedure. The *syntax* for each argument is

Variablename [as datatype]

If you don't provide a type, the argument accepts a variant data type.

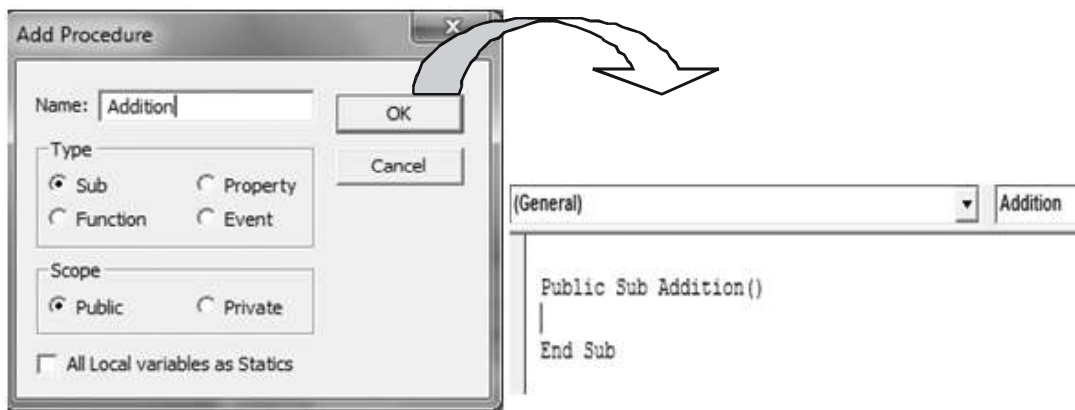
**Example:** Public Sub Addition (num1 as integer, num2 as integer)

Msgbox( " Sum of numbers are :." & (num1+num2)

End Sub

### 5.3.2 Add Procedure Menu Option

Open the code window and type the code for your procedure. VB provides a tool to add procedures. For this firstly open the code window and then Select **Tools Add Procedure** command to bring up the add procedure window where you can name your new procedure , its type and scope and whether to use static variables or not. When you click **OK** , VB places a skeleton code for your procedures in the code window. Now you can write your procedure code between **Sub** and **End Sub**.



### 5.3.3 Calling Sub-Procedures

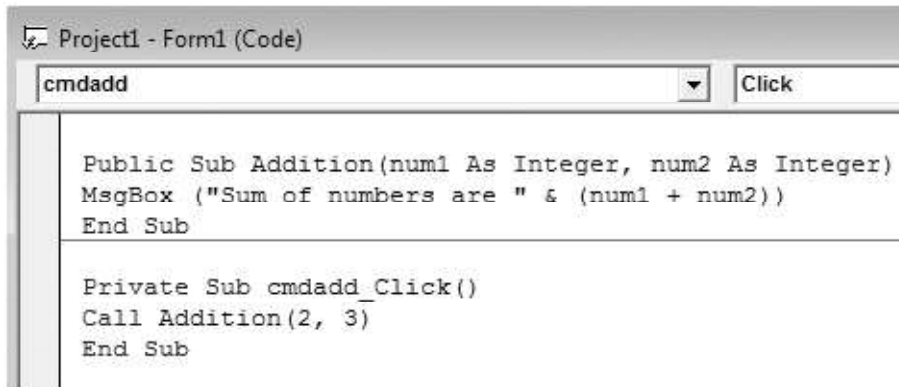
You invoke a Sub procedure explicitly with a stand-alone calling statement. You cannot call it by using its name in an expression. You must provide values for all arguments that are not optional, and you must enclose the argument list in parentheses. If no arguments are supplied, you can optionally omit the parentheses. The use of the Call keyword is optional but not recommended.

The syntax for a call to a Sub procedure is as follows:

[Call] Procedurename [( Parameterlist )]

**Example 1:** WAP to add two numbers by using procedure .

Create the procedure addition and call it on click event of add button to add numbers 2 and 3



```

Public Sub Addition(num1 As Integer, num2 As Integer)
    MsgBox ("Sum of numbers are " & (num1 + num2))
End Sub

Private Sub cmdadd_Click()
    Call Addition(2, 3)
End Sub

```

You may call procedure in two ways:

(i) With a call statement

Eg. Call Addition (2, 3)      ‘must use parentheses

(ii) Without a call statement

Eg. Addition 2, 3      ‘must call without parentheses

**Exit Sub** statement immediately exits the Sub procedure in which it appears and the execution continues with the statement following the statement that called the Sub procedure.

## 5.4 FUNCTION PROCEDURE

A function is a procedure that performs a specific task and returns a value. It is a series of Visual Basic statements enclosed by the **Function** and **End Function** statements. The function procedure performs a task and then returns control to the calling code. When 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.

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.

There are two types of functions in VB:

(i) **Built-in-Function/Intrinsic Function:** These functions are already defined in VB. When required user just have to call these functions. We discuss this later in this chapter.

(ii) **User Defined Function:** These functions are created by user to perform their own task.

### 5.4.1 Declaring Syntax

The syntax for declaring a Function procedure is as follows:

```
[modifiers] Function functionname [( parameterlist )] As returntype
    Statements of function
    Return functionname
End Function
```

The *modifiers* can specify access level i.e public, private or protected. You declare each parameter the same way you do for Sub Procedures.

Every Function procedure has a data type, just as every variable does. This data type is specified by the *As* clause in the Function statement, and it determines the data type of the value the function returns to the calling code. The value a Function procedure sends back to the calling code is called its *return value*. The procedure returns this value in one of two ways:

- It assigns a value to its own function name in one or more statements of the procedure. Control does not return to the calling program until an **Exit Function** or **End Function** statement is executed.

```
Function functionname [( parameterlist )] As returntype
    functionname = expression
End Function
```

- It uses the **Return** statement to specify the return value, and returns control immediately to the calling program.

```
Function functionname [( parameterlist )] As returntype
    Return expression
End Function
```

The advantage of assigning the return value to the function name is that control does not return from the procedure until it encounters an *Exit Function* or *End Function* statement. This allows you to assign a preliminary value and adjust it later if necessary.

**Example:** Public Function Addition (num1 as integer, num2 as integer) As integer

```
    Ans = num1 + num2
    Addition = Ans
End Sub
```

### 5.4.2 Calling Function

You invoke a Function procedure by including its name and arguments either on the right side of an assignment statement or in an expression. You must provide values for all arguments that are not optional, and you must enclose the argument list in parentheses. If no arguments are supplied, you can optionally omit the parentheses.

The syntax for a call to a **Function** procedure is as follows:

$$lvalue = functionname [( argumentlist )]$$

Now, solve the example1 by using functions.

**Example 2:** WAP to add two numbers by using Function .

Create the procedure addition and call it on click event of add button to add numbers 2 and 3

```

cmdadd
Click

Public Function Addition(num1 As Integer, num2 As Integer) As Integer
    Dim ans As Integer
    ans = num1 + num2
    Addition = ans
End Function

Private Sub cmdadd_Click()
    Dim a As Integer
    a = Addition(2, 3)
    MsgBox ("Sum of numbers are " & a)
End Sub

```

**Exit Function** statement immediately exits the function procedure in which it appears and the execution continues with the statement following the statement that called the Function.

## 5.5 PASSING PARAMETERS TO PROCEDURES

In most cases, a procedure needs some information about the circumstances in which it has been called. A procedure that performs repeated or shared tasks uses different information for each call. This information consists of variables, constants, and expressions that you pass to the procedure when you call it.

A *parameter* represents a value that the procedure expects you to supply when you call it. The procedure's declaration defines its parameters. You can define a procedure with no parameters, one parameter, or more than one. The part of the procedure definition that specifies the parameters is called the parameter list. An argument represents the value you supply to a procedure parameter when you call the procedure. The calling code supplies the arguments when it calls the procedure. The part of the procedure call that specifies the arguments is called the argument list.

In Visual Basic, you can pass an argument to a procedure **by value** (ByVal) or **by reference** (ByRef). This is known as the *passing mechanism*, and it determines whether the procedure can modify the programming element underlying the argument in the calling code.

You should choose the passing mechanism carefully for each argument to provide

- **Protection.** In choosing between the two passing mechanisms, the most important criterion is the exposure of calling variables to change. The advantage of passing an argument by reference is that the procedure can return a value to the calling code through that argument. The advantage of passing an argument by value is that it protects a variable from being changed by the procedure.
- **Performance.** Although the passing mechanism can affect the performance of your code, the difference is usually insignificant. One exception to this is a value type passed by value. In this case, Visual Basic copies the entire data contents of the argument. Therefore, for a large value type such as a structure, it can be more efficient to pass it by reference.

The default in Visual Basic is to pass arguments by value. You can make your code easier to read by using the **ByVal** keyword. It is good programming practice to include either the **ByVal** or **ByRef** keyword with every declared parameter.

### 5.5.1 Call by Value

The call by value method copies the values of actual parameters into the formal parameters, that is, the procedure takes its own copy of argument value and uses them. When referencing something by value in your method you are only taking a ‘copy’ of the value, so any changes are only made to your copy. Thus, in call by method, the changes are not reflected back to the original values.

This can be done by using keyword **ByVal** when declaring arguments as

```
Sub procedurename(ByVal variable As datatype)
    '
    '
    '
EndSub
```

**Example:** Private Sub Welcome(ByVal Language As String)  
    MsgBox(“Welcome to the world of “ & Language)  
End Sub

When To Pass an Argument by Value

- If the calling code element underlying the argument is a nonmodifiable element, declare the corresponding parameter **ByVal**. No code can change the value of a nonmodifiable element.
- If the underlying element is modifiable, but you do not want the procedure to be able to change its value, declare the parameter **ByVal**. Only the calling code can change the value of a modifiable element passed by value.

### 5.5.2 Call By Reference

An alternative to passing an argument as done so far is to pass the address of the argument to the procedure. When this is done, the procedure doesn’t receive a simple copy of the value of the

argument: the argument is accessed by its address. That is, at its memory address. With this technique, any action carried on the argument will be kept. If the value of the argument is modified, the argument would now have the new value, dismissing or losing the original value it had. This technique is referred to as passing an argument by reference.

In call by reference, the called procedure does not create its own copy of original values, rather, it refers to original values only by different names i.e reference. Thus, using the reference method you are taking the reference of your object so any changes made are made to the actual object.

This can be done by using keyword **ByRef** when declaring arguments as

```
Sub procedurename(ByRef variable As datatype)
    '
    '
    '
EndSub
```

**Example:** Private Function Addition(ByRef Value1 As Integer, ByRef Value2 As Integer)

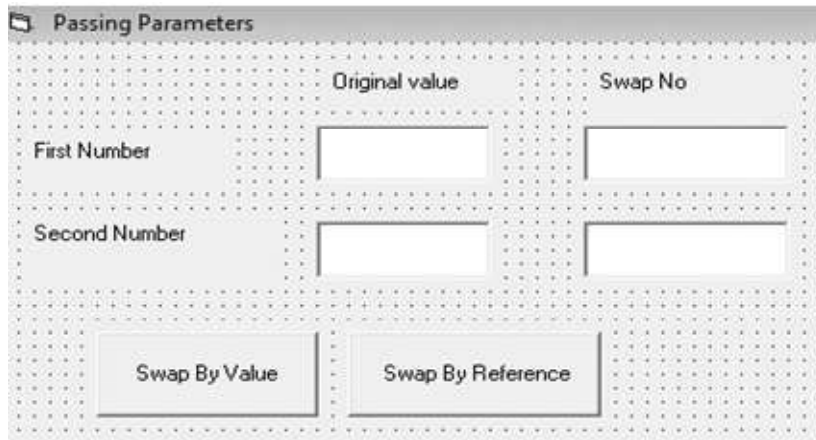
```
Value1 = InputBox("Enter First Number: ")
Value2 = InputBox("Enter Second Number: ")
Addition = Value1 + Value2
End Function
```

When To Pass an Argument by Reference

- If the procedure has a genuine need to change the underlying element in the calling code, declare the corresponding parameter **ByRef**.
- If the correct execution of the code depends on the procedure changing the underlying element in the calling code, declare the parameter **ByRef**. If you pass it by value, or if the calling code overrides the **ByRef** passing mechanism by enclosing the argument in parentheses, the procedure call might produce unexpected results.

**Example 3:** WAP to swap two values by using both (ByVal & ByRef) methods.

Design the form as follows



Control	Property
Form	Name – frmswap Caption – Passing Parameters
Label 1	Name – lblfirst Caption – First Number
Label 2	Name – lblsecond Caption – Second Number
Label 3	Name – lblorig Caption – Original Value
Label 4	Name – lblswap Caption – Swap No
Textbox 1	Name – txtfirst Text – [Empty]
Textbox2	Name – txtsec Text – [Empty]
Textbox 3	Name – txtswapf Text – [Empty]
Textbox2	Name – txtswaps Text – [Empty]
CommandButton	Name – cmdref Caption – Swap By Reference
CommandButton	Name – cmdval Caption – Swap By Value

Write the following code:

```
Public Sub swapByRef(ByRef one As Integer, ByRef two As Integer)
    Dim temp As Integer
    temp = one
    one = two
    two = temp
End Sub

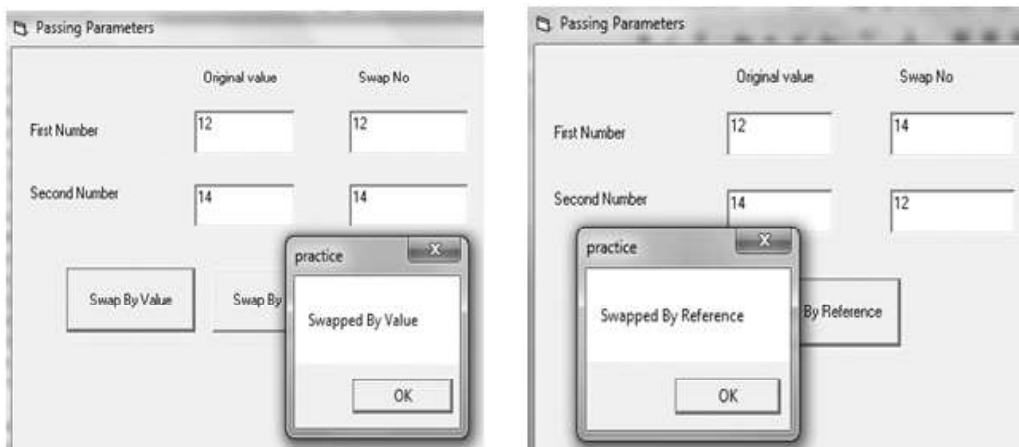
Public Sub swapByVal(ByVal one As Integer, ByVal two As Integer)
    Dim temp As Integer
    temp = one
    one = two
    two = temp
End Sub

Private Sub cmdref_Click()
    Dim first As Integer, sec As Integer
    first = Val(txtfirst.Text)
    sec = Val(txtsec.Text)
    Call swapByRef(first, sec)
    txtswapf.Text = first
    txtswaps.Text = sec
    MsgBox ("Swapped By Reference")
End Sub

Private Sub cndval_Click()
    Dim first As Integer, sec As Integer
    first = Val(txtfirst.Text)
    sec = Val(txtsec.Text)
    Call swapByVal(first, sec)
    txtswapf.Text = first
    txtswaps.Text = sec
    MsgBox ("Swapped By Value")
End Sub
```

At run time when you click on value is invoked by value, the original value remains unchanged but when this is invoked by reference the original value get swapped. It appears as :





## 5.6 PROPERTY PROCEDURE

Till now you have worked with controls that are. To use these controls you need to set some of their properties and define a few of the event procedures. But these properties and events are predefined.

You can also create your own controls which can be added in the form like OCX controls. The objects that you create yourself are called class modules. That is, if you want you can also create your own controls and add them to toolbox as OCX control.

Property procedures are the code which runs when a property of a control gets new value or the value is retrieved. It is used to create and manipulate custom properties. It is used to create read only properties for Forms, Standard modules and Class modules. Visual Basic provides three kind of property procedures-

- **Let procedure:** Sets the value of a property.
- **Get procedure:** Returns the value of a property.
- **Set procedure:** Sets the references to an object.

## 5.7 CODE MODULE

A key part of developing applications using Visual Basic is ensuring that the code is carefully structured. This involves segmenting the code into projects, modules and procedures so that it is easy to understand and maintain. A complete Visual Basic application is typically contained in a single project. Within a project, code is placed in separate code files called modules, and within each module, the Visual Basic code is further separated into self contained and re-usable procedures.

A **module** is a code container in VB, that contains some procedure and definitions.

Following are three types of modules in VB –

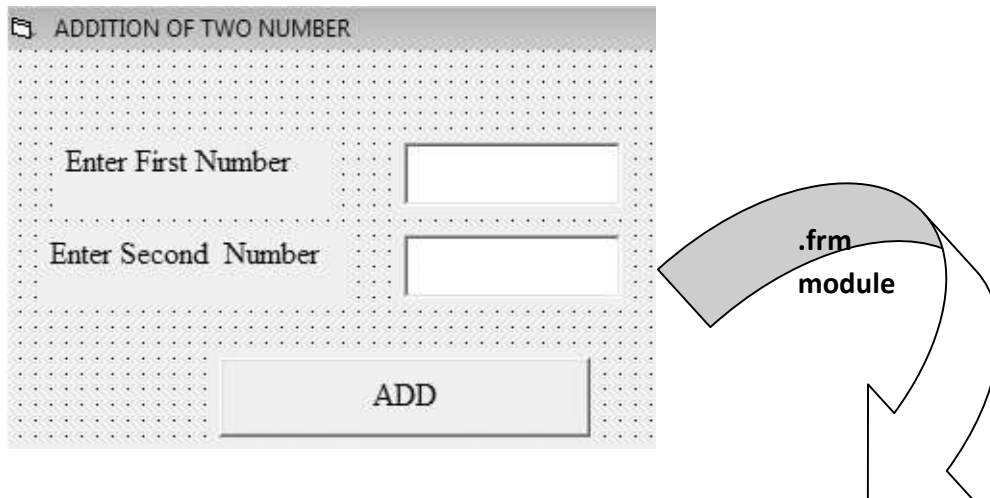
- Form Modules
- Standard Modules
- Class Modules

### 5.7.1 Form Module

You have developed many applications in VB. An application contains many forms. All the code of a single form resides in a module called a form module. Basically, a form is a module that stores all the procedures and declarations pertaining to single form.

The form modules can contain procedures that handle events, general procedures and form level declarations of variables, constants, types and external procedures. All the declarations under general section, sub or functions of a form, is accessible from anywhere within the form. Thus, all these are part of a one single module – the form module.

The form modules are saved with extension `.frm`. You can open a `.frm` file in an editor window and see description of the form and its control including their property setting. Following window shows the form module of given form -





```
EX2 - Notepad
File Edit Format View Help
VERSION 5.00
Begin VB.Form frmadd
    Caption = "ADDITION OF TWO NUMBER"
    ClientHeight = 3030
    ClientLeft = 120
    ClientTop = 450
    ClientWidth = 4560
    LinkTopic = "Form1"
    ScaleHeight = 3030
    ScaleWidth = 4560
    StartupPosition = 3 'windows default
Begin VB.CommandButton cmdadd
    Caption = "ADD"
BeginProperty Font
    Name = "Times New Roman"
    Size = 12
    Charset = 0
    weight = 400
    underline = 0 'False
    Italic = 0 'False
    Strikethrough = 0 'False
EndProperty
    height = 615
    Left = 1680
    TabIndex = 4
    Top = 2400
    width = 2895
End
Begin VB.TextBox txtsecond
BeginProperty Font
    Name = "Times New Roman"
    Size = 12
    Charset = 0
    weight = 400
    underline = 0 'False
    Italic = 0 'False
    Strikethrough = 0 'False
EndProperty
    height = 495
    Left = 3120
    TabIndex = 1
End
```

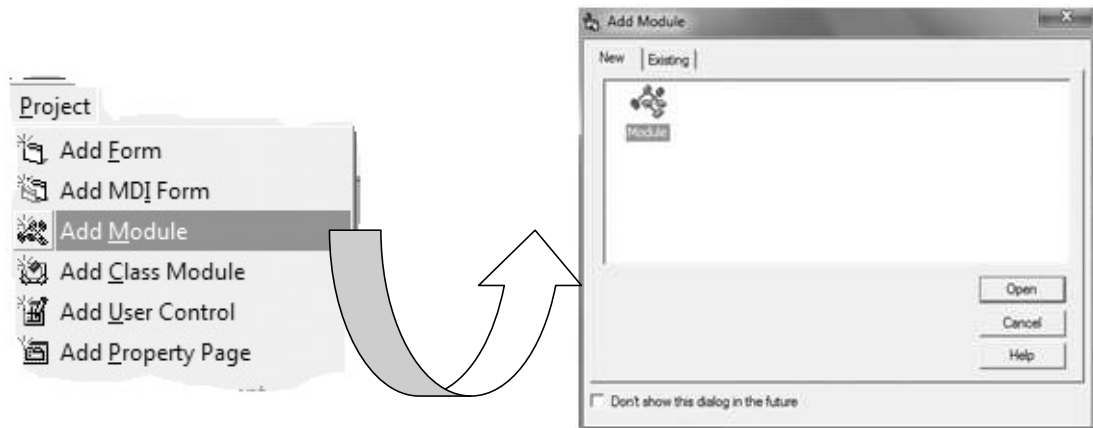
## 5.7.2 Standard Module

The standard modules are the modules that store general purpose code of the application i.e. the code and declaration are not specific to one single form of the application.

It is like a form without graphical interface – it is a code container, consisting only of a general declarations section, accessible from anywhere within the program. The standard module stores the procedures and declarations commonly accessed by other modules within the application. It stores with extension *.bas* .

Steps to add standard module

- Select Add Module from Project Menu.
- Select Module from Add Module dialog box and click on Open.

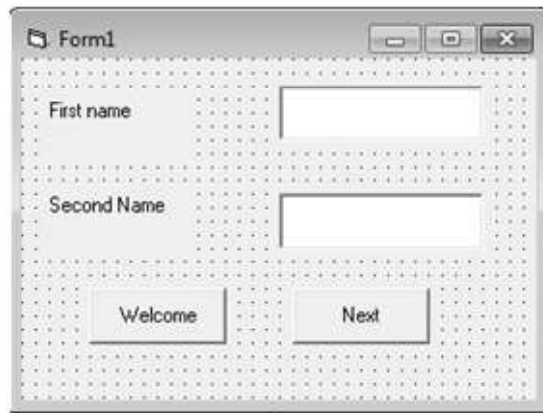


This will open the code window for new module and add it to the project explorer. When module is selected, the Form view button is disabled because module does not contain any control. The module itself is a kind of control, it has only property Name.



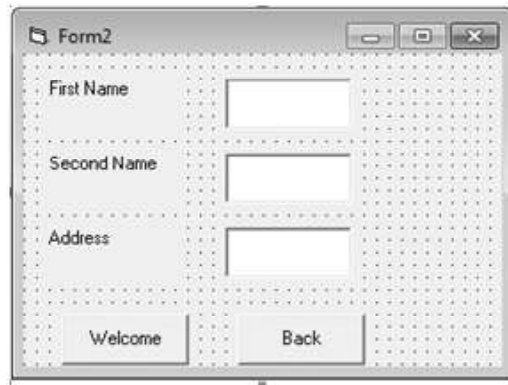
**Example 4:** Design an application with two forms .Both forms should be able to share data with each other,

Design the first form as



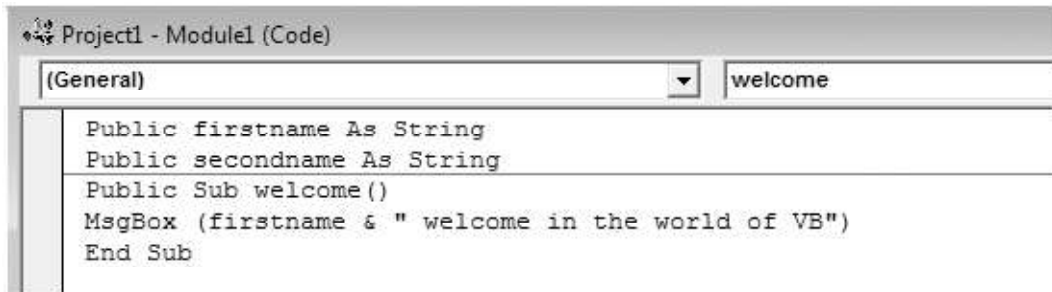
Control	Property
Form	Name – frmmod1
Label 1	Caption – Form1
Label 2	Name – lblfirst
Textbox 1	Caption – First Number
	Name – lblsecond
	Caption – Second Number
	Name – txtfirst
Textbox 2	Text – [Empty]
	Name – txtsec
Command 1	Text – [Empty]
	Name – cmdwelcome
Command 2	Caption – Welcome
	Name – cmdnext
	Caption – Next

Add another form in the same project by selecting **Project — Add Form**. Design the second form as



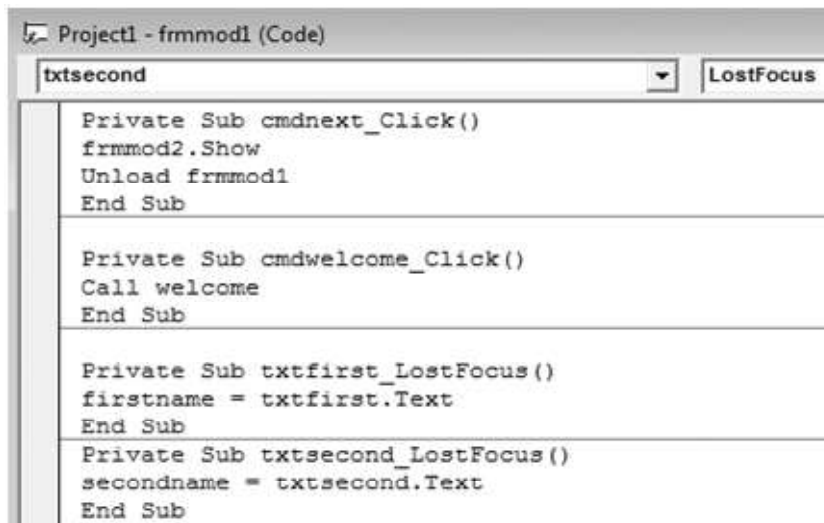
Control	Property
Form	Name – frmmod2
Label 1	Caption – form2
Label 2	Name – lblfirst
Label 3	Caption – First Number
Textbox 1	Name – lblsecond
	Caption – Second Number
Textbox 2	Name – lbladdress
	Caption – Address
Textbox 2	Name – txtfname
	Text – [Empty]
	Locked – True
Textbox 2	Name – txtsname
	Text – [Empty]
	Locked – True
Command 1	Name – txtaddress
	Text – [Empty]
	Locked – false
Command 1	Name – cmdwelcome
	Caption – Welcome
Command 2	Name – cmdback
	Caption – Back

Now add a module in the same project by selecting **Project — Add Module**. Design the module as



```
Project1 - Module1 (Code)
(General) welcome
Public firstname As String
Public secondname As String
Public Sub welcome()
MsgBox (firstname & " welcome in the world of VB")
End Sub
```

Write the following code in code window of Form1



```
Project1 - frmmod1 (Code)
txtsecond LostFocus
Private Sub cmdnext_Click()
frmmod2.Show
Unload frmmod1
End Sub
Private Sub cmdwelcome_Click()
Call welcome
End Sub
Private Sub txtfirst_LostFocus()
firstname = txtfirst.Text
End Sub
Private Sub txtsecond_LostFocus()
secondname = txtsecond.Text
End Sub
```

Now write the following code in code window of Form2

```
Project1 - frmmod2 (Code)
Form Load
Private Sub cmdback_Click()
    frmmod1.Show
    Unload frmmod2
End Sub

Private Sub cmdwelcome_Click()
    Call welcome
End Sub

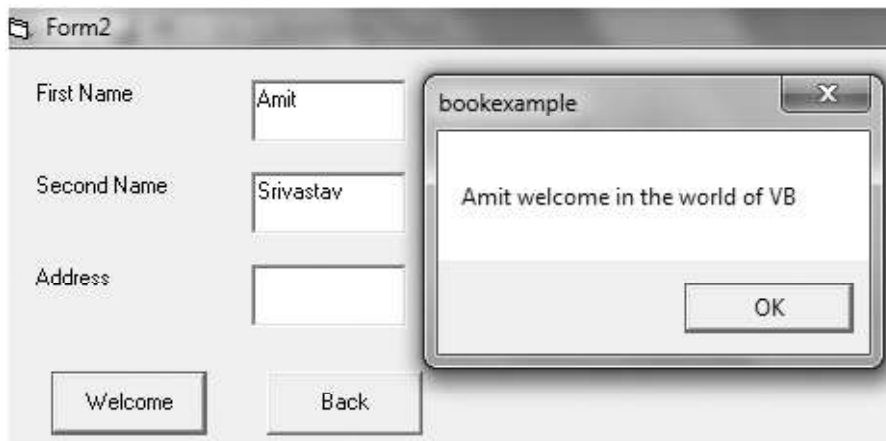
Private Sub Form_Load()
    txtfname.Text = firstname
    txtsname.Text = secondname
End Sub
```

At run time it appear as



When you click on the next button second form will open. When you click on welcome button of any form following msgbox will appear

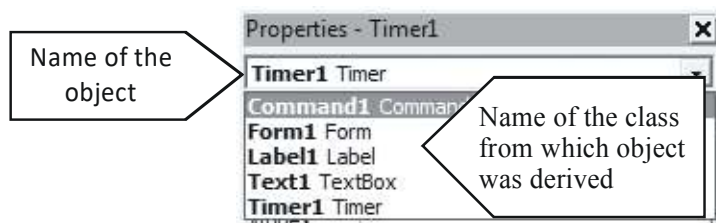




### 5.7.3 Class Module

Object Oriented programming was introduced to make large program easier to make and handle. The OOP resolves the problem by creating objects each with their own properties, methods, functions etc. The entire programming interface of VB is based upon some objects that you place on the forms. The toolbox and components of project are loaded with objects that you can add to your project. These are the objects that someone else has created; you can also create your own control with the help of class module.

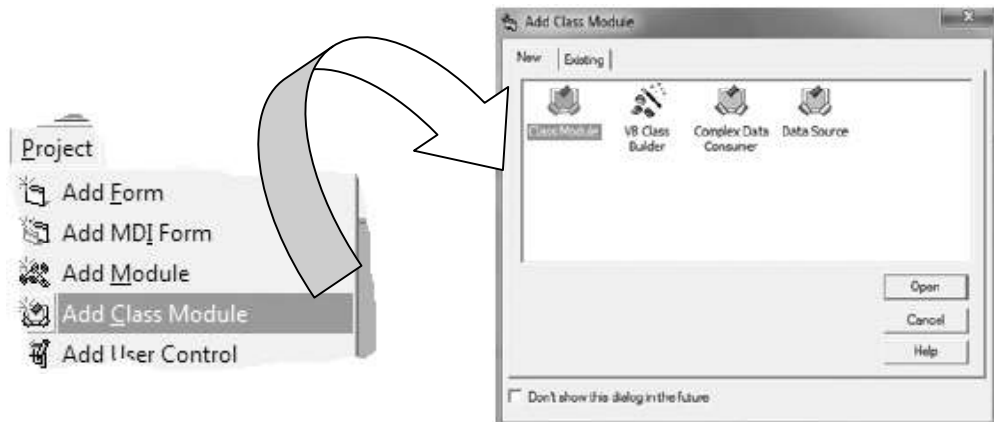
A class module is a special code module that stores the blueprint for user created custom objects. When you need the functionality of a particular object you create an instance of it, based on the code contained in the class module. Each class module can contain only one class that defines everything about the new object. Basically class defines the properties and behavior for a type and an object is created instance of its class type. You can create any number of objects from a class.



### Adding a class module

To add a class module in a project, just click on Project — Add class module. This will open a class module dialog box. Select class module and then open. Now you find a code window opened to

type the code for new class, its properties, methods etc. The class module is saved with .cls extension.



### Creating properties and methods

Properties are used to define characteristics of the object. To create a property of a new object, a module level variable is defined to hold property value and two procedures property Let and property Get. Property procedures are the code which runs when a property gets a new value (Let) or when the value is retrieved (get).

Methods are defined to implement the behaviors of the object. Methods are written through sub or function procedures.

## 5.3 LIBRARY FUNCTION

The functions which are available in VB to use directly without passing their definition are called built-in-function. They are also called library function. It offers a set of functions to manipulate strings, date, times and number. These built in functions are useful as they save time and efforts.

### 5.3.1 String Function

The string functions allow you to work with strings in numerous ways such as changing cases, extracting characters from a string, determining whether a character is a part of a string etc. etc.

- 1. The LCase and UCase Functions:** These two functions convert strings to all lower or all upper case. The LCase( ) functions converts a string into all lower case and UCase( ) does exactly the opposite. These functions might be useful if you want to compare strings. The Syntax of this function is

#### Syntax:

LCase(String)

UCase(String)

Consider the following example

Print UCase("hello world")

The output would be HELLO WORLD.

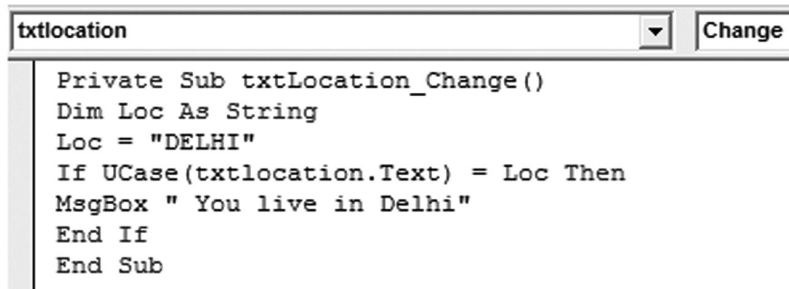
Similarly,

Print LCase('Hello WORLD')

The output would be hello world.

**Example 5:** Obtain text from textbox txtLocation and determine whether the user lives in Delhi or not. The user is allowed to enter the location in any possible case combination.

Write the following code at change event of txtlocation



```
txtlocation Change
Private Sub txtLocation_Change()
Dim Loc As String
Loc = "DELHI"
If UCase(txtlocation.Text) = Loc Then
MsgBox " You live in Delhi"
End If
End Sub
```

- 2. The Len Function:** This Function gives you the length of the string i.e. how many characters long the string is. It counted all the characters i.e. punctuation, numbers, alphabets, special characters and blank spaces etc.

### Syntax

Len(String)

### Consider this:

Len(" Visual Basic")

The output will be 12

- 3. The Trim, LTrim and RTrim Functions:** These functions remove leading (LTrim function) or trailing(RTrim Function) from a string. These functions don't affect any spaces between words. The Trim() function simply accomplishes both LTrim() and RTrim() function, i.e. removes all leading and trailing blanks.

### Syntax

LTrim(String)

RTrim(String)

Trim(String)

Users may inadvertently type spaces into a text box, and you can use these functions to account for that, but mostly the Trim( ) function is used with fixed-Length strings, user defined types, and Random Access Files.

**Consider the example:**

Trim(“ Visual Basic “)

The output will remove a blank space before Visual i.e Visual Basic

- 4. Left and Right Functions:** These are two functions that are used to extract a certain number of characters from the leftmost or rightmost portions of a string. These functions require two arguments: the original string and the number of characters to extract from that string.

**Syntax**

Left(string, no-of characters)

Right(string, no-of characters)

consider the following example code.

name = “ Dinesh Ch. Srivastav”

Print Left(name, 6)

Print Right(name, 8)

The above code produces the result:

Dinesh      ‘extract 6 leftmost characters from the name string.

rivastav      ‘extracts 8 rightmost characters from the name string.

- 5. Mid Function and Mid Statement:** Mid function is used to extract characters from the middle of a string. We need three arguments here: the Original String, the place to Start Extracting characters, and the number of characters to extract.

**Syntax**

Mid(String, start-position, no-of-characters)

for example:

Print Mid(“The Pink City “, 7, 3)

The output will be ‘nk ‘

Starting from the 7th character extracting 3 characters including the start position character. Also consider that all characters are included in the operation.

The Mid Statement not only extracts the characters, but also replaces them with the text you specify.

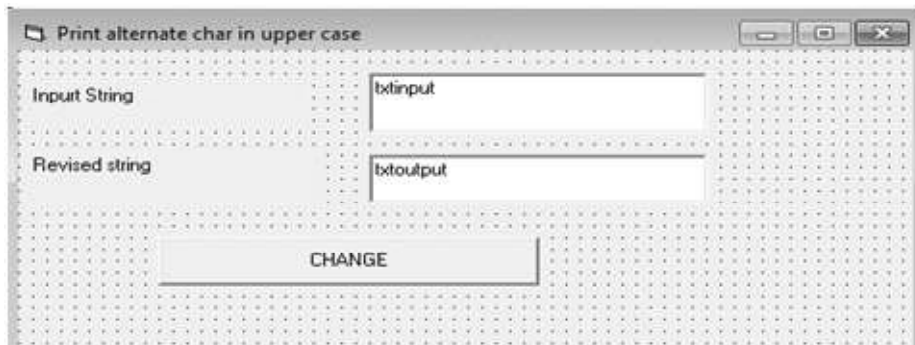
For example, in the following code we put ‘-‘ in the given string

Sinput = “Object oriented”

Mid(sinput,7,1) = “-”

**Example 6:** Write a function namely AltCap that receives a String argument and returns the string wherein each alternate character is in Uppercase.

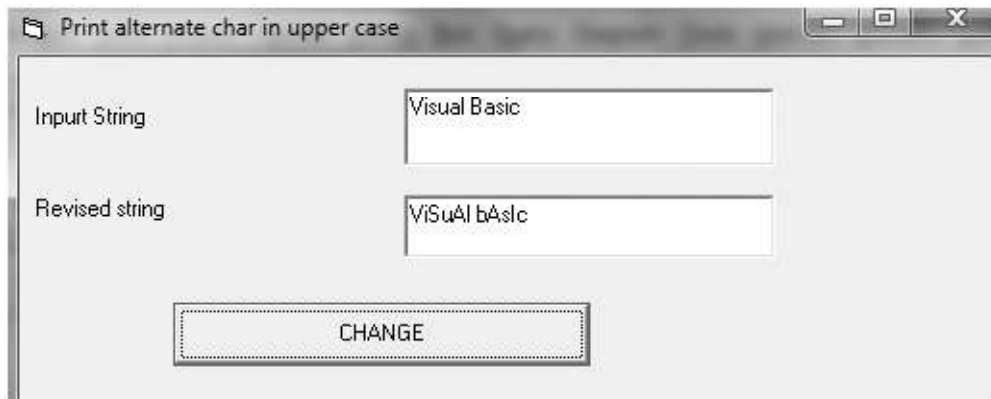
Design the following form:



Write the following code at click event of change button

```
cmdchange
Private Sub cmdchange_Click()
Dim i As Integer, j As Integer
Dim a As String, b As String
a = txtinput.Text
j = Len(a)
For i = 1 To j
    b = Mid(a, i, 1)
    If i Mod 2 = 0 Then
        Mid(a, i, 1) = LCase(b)
    Else
        Mid(a, i, 1) = UCase(b)
    End If
Next i
txtoutput.Text = a
End Sub
```

At run time it appear as



**6. InStr Function:** The InStr function searches for strings within strings.

Syntax

InStr([start],string1,string2,[Compare]) ‘ [ ] means optional.

Where

*Start* - Is a numeric expression that sets the starting position for each search, if omitted, the search begins at the first character of the string.

*String1* - Is the String in which to search.

*String2* - Is the string to be searched for.

*Compare* - Specifies the type for string comparison. 0 for case sensitive search and 1 for case insensitive search.

There are two ways to compare strings - case sensitive and case insensitive.

- case sensitive is a Binary Comparison  
e.g. string “VISUAL” and string “VisUal” are not equal in this case.
- case insensitive is a Text Comparison.  
e.g. string “VISUAL” and string “VisUal” are equal in this case.

By default VB6 will use the Binary method to compare strings unless explicitly specified.

**Now Consider the Following Code.**

```
Dim SearchString, SearchChar, MyPos
```

```
SearchString = “The earth is the third planet in the Solar system”
```

```
SearchChar = “S”
```

```
‘Textual Comparison starting at position 1 returns 12
```

```
MyPos = InStr( 1,SearchString, SearchChar,1)
```

```
Print MyPos
```

```
‘Binary Comparison starting at position 1 returns 38
```

```
MyPos = InStr( 1,SearchString, SearchChar,0)
```

```
Print MyPos
```

```
‘If you simply omit the compare and start argument, it will return 38
```

```
MyPos = InStr( SearchString, SearchChar)
```

```
Print MyPos
```

InStr is a function and will return the position of the first occurrence of the search string . if the string is not found then it will return 0

InStrRev() is a related function here. It works similar to the InStr function, but it performs the search BACKWARDS.

**7. Space Function:** This function by itself produces a certain number of spaces.

### Syntax

Space(number) ‘Number argument is the number of spaces you want in the string.

Consider the following example code.

```
Dim MyString
```

```
MyString = “HelloWorld”
```

```
Print MyString
```

```
‘Following line inserts 10 blank spaces in between “Hello” and “World”
```

```
MyString = “Hello” & Space(10) & “World”
```

```
Print MyString
```

```
The output will be “ Hello      World”
```

**8. String Function:** This function is used for producing a string with certain number of repeating characters.

### Syntax

```
String(number,character)
```

Where Number is number of characters to be repeated and character is the character to repeat

ForExample, the following code

```
sResult = String(10,66)
```

```
will return BBBBBBBBBB as 66 is character code of ‘B’
```

Now consider this:

```
MyString = String(10, “ABC”)
```

Will return AAAAAAAAAA

Remember that no matter how long the string you enter in the character argument, it will always select the first character of that string.

**9. Str Function:** This function converts a number into equivalent string.

**Syntax**

Str(number)

Where number argument is a long containing any valid numeric expression.

When numbers are converted to strings, a leading space is always reserved for the sign of number. If number is positive, then the returned string contains a leading space and the plus sign is implied, else a “-” sign is put in front of the number.

The following example uses the Str Function to return a string representation of a number.

```
Dim MyString
```

```
MyString = Str(1302)
```

```
Print MyString          ‘Returns 1302
```

```
MyString = Str(-12503.54)
```

```
Print MyString          ‘Returns -12503.54
```

**10. Asc Function:** This function is used to get a character’s equivalent ASCII code.

**Syntax**

Asc(String)

Normally you would use a single character enclosed in quotes or one character resulting from another function, but using a multi character string doesn’t cause an error, the function will simply take the first character of the string.

For Example the following code will print 68, the ASCII code of character “D”.

```
City = “Dehradun”
```

```
Result = Asc(City)
```

```
Print Result
```

**11. Chr Function:** This function returns a String containing the character associated with the specified character code.

**Syntax**

Chr(charcode)

where the required charcode argument is a Long that identifies a character.

Numbers from 0 - 31 are the same as standard, non printable ASCII codes. For example,



Chr(10) returns a linefeed character. The normal range for charcode is 0 - 225. However on DBCS systems, the actual range for charcode is from -32768 to 65535.

consider the following example:

```
Dim MyChar
```

```
MyChar = Chr(65) 'Returns A
```

```
MyChar = Chr(97) 'Returns a
```

- 12. StrReverse Function:** This Function returns a string in which the order of a specified string is reversed.

### Syntax

```
StrReverse(String1)
```

Where String1 is the argument whose characters are to be reserved.

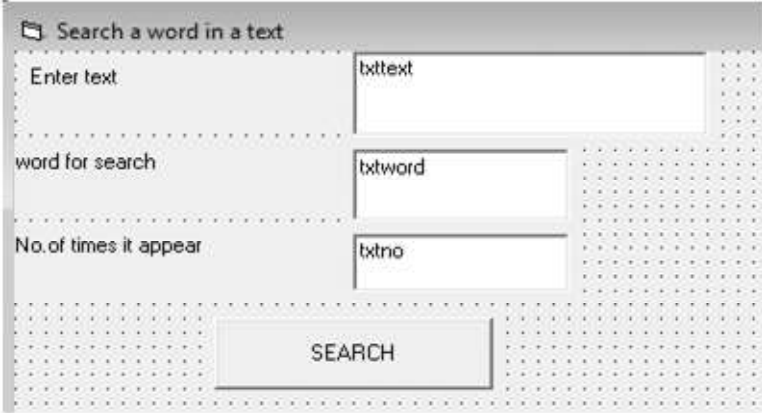
For Example:

```
StrReverse("Visual")
```

Will return "lausiV"

**Example 7:** WAP that lets the user to enter a line of text and a word to be searched for. It then reports how many times the word has occurred in the line of text.

Design the following form



The image shows a graphical user interface for a search application. It is a dialog box with a title bar that says "Search a word in a text". Inside the dialog, there are three text input fields arranged vertically. The first field is labeled "Enter text" and has the ID "txttext". The second field is labeled "word for search" and has the ID "txtword". The third field is labeled "No. of times it appear" and has the ID "txtno". At the bottom center of the dialog, there is a button labeled "SEARCH".

Write the following code at click event of search button

```

cmdsearch
Click
Private Sub cmdsearch_Click()
Dim l As Integer, lenword As Integer, freq As Integer
Dim word As String
l = Len(txttext.Text)
lenword = Len(txtword.Text)
For a = 1 To l
    word = Mid(txttext, a, lenword)
    If word = txtword Then
        freq = freq + 1
    End If
Next a
txtno.Text = Str(freq)
End Sub

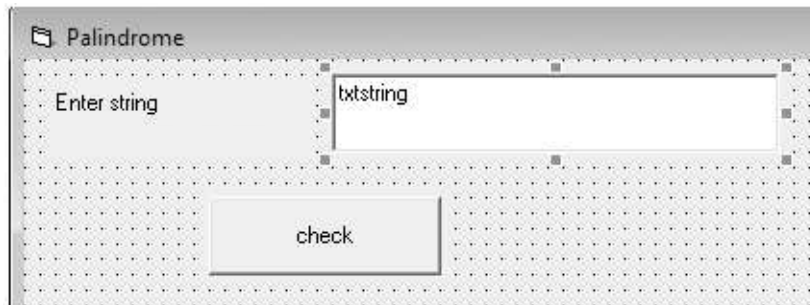
```

At runtime , it appear as

The screenshot shows a window titled "Search a word in a text". It has a text area labeled "Enter text" containing the text "This is user interface that the user sees and which is responsible for interacting with the user.". Below the text area are two text boxes: "word for search" containing "user" and "No. of times it appear" containing "3". A "SEARCH" button is located at the bottom of the window.

**Example 8:** WAP to accept a string from user. And then check whether that string is palindrome or not.

Design the following form



Write the following at click event of check button

```
cmdcheck Click
Private Sub cmdcheck_Click()
Dim strg As String, revstr As String
Dim onechar As String * 1
Dim a As Integer, l As Integer, p As Integer
strg = Trim(Txtstring.Text)
l = Len(strg)
p = 1
revstr = ""
For a = 1 To l
    onechar = Mid(strg, p, 1)
    revstr = revstr & onechar
    p = p - 1
Next a
If strg = revstr Then
    MsgBox ("String is palindrome")
Else
    MsgBox ("string is not palindrome")
End If
End Sub
```

At run time it appear as



### 5.8.2 Numeric Function

The mathematical functions are very useful and important in programming because very often we need to deal with mathematical concepts in programming such as chance and probability, variables, mathematical logics, calculations, coordinates, time intervals and etc.

Some commonly user functions are:

- 1. Int Function:** Int is the function that converts a number into an integer by truncating its decimal part and the resulting integer is the largest integer that is smaller than the number. It returns the first negative integer less than or equal to the number.

#### Syntax

Int(number)

#### Example:

Int(2.4)=2,

Int(4.8)=4,

Int(-4.6)= -5,

Int(0.032)=0

- 2. Fix Function:** Fix truncates the fractional portion of a number. If the number is a positive number as both truncate the decimal part of the number and return an integer. However, when the number is negative, it will return the first negative integer greater than or equal to the number.

#### Syntax

Fix(number)

#### Example:

Fix(-6.34)= -6

Fix(-14.7) = -14

- 3. Round Function:** Round is the function that rounds up a number to a certain number of decimal places.

**Syntax**

Round(number,m)

Where m is number of places to round of. The Format Round (n, m) means to round a number to m decimal places.

**Example:**

Round (7.2567, 2) =7.26

- 4. Sgn Function:** This function is used to determine the sign of a number.

**Syntax:**

Sgn(number)

It returns a value indicating the sign of a number as

If number is less than zero, -1.

If number is equal to zero, 0.

If number is greater than zero, 1.

**Example:**

Dim val1, val2 , ans

Val1 = 12

Val2 = -12.3

Ans = sgn(val1)     ‘Return 1 as number is greater than 0

Ans = sgn(val2)     ‘Return -1 as number is less than 0

- 5. Abs Function:** This function returns the absolute value of a number.

**Syntax**

Abs(number)

The return value matches the type of number. If *number* is a variant string , the return value is converted to double, long or integer.

**Example:**

Abs(-8) = 8

Abs(8)= 8.

- 6. Exp Function:** This function returns  $e$  (the base of natural logarithms) raised to a power. Exp of a number x is the value of  $e^x$ . Its return type is double.

**Syntax:**

Exp(number)

**Example:**

$\text{Exp}(1)=e^1 = 2.7182818284590$

**7. Sqr Function:** This function returns the square root of a number.

**Syntax**

Sqr(number)

**Example:**

Sqr(4)=2

Sqr(9)=2

**5.3.2 Date and Time Function**

This section deals with various date and time function. Before starting, remember one thing that, computer remembers its own date and time. So before proceeding check your system's date and time, to get the correct result.

The Date and Time functions return the system date and time in the four-byte Date format.

**1. Now Function:** It returns the current date and time.

**Syntax**

Now()

The value returned by Now function is of variant type. It considers both date and time and return the value in following format:

mm/dd/yy hh:mm:ss[AM][PM]

**Example:**

Print Now() / Now

Its output is 9/19/11 11.27AM

**2. Date & Date\$ Function:** Both Date and Date\$ function returns the system date.

**Syntax**

Date()

Date\$()

The Date() function returns the current date in variant datatype in the following format:

mm/dd/yy        '9/19/11

The Date\$() function returns the current date in string datatype in the following format:

mm-dd-yy        '09-19-11

**3. Time & Time\$ Function:** Both Time and Time\$ function returns the system time.

**Syntax**

Time()

Time\$()

The Time() function returns the current time in variant datatype in the following format:

hh:mm:ss[AM][PM]      hh can be 1-12, mm 0-59, ss 0-59

**Example:**

02:04:14 PM

The Time\$() function returns the current time in string datatype in the following format:

hh:mm:ss      hh can be 0-23, mm 0-59, ss 0-59

**Example:**

13:03:45

- 4. DatePart Function:** The DatePart function returns an Integer containing the specified part of a given date/time value.

**Syntax**

DatePart(interval, date)

Where date is a valid date value that you want to evaluate and interval is a string expression that is the interval of time you want to return. The interval can take one of the following values –

Interval	Description	Possible Range of Values
“yyyy”	Year	100 to 9999
“q”	Quarter	1 to 4
“m”	Month	1 to 12
“y”	Day of year	1 to 366 (a “Julian” date)
“d”	Day	1 to 31
“w”	Weekday	1 to 7
“ww”	Week	1 to 53
“h”	Hour	0 to 23
“n”	Minute	0 to 59
“s”	Second	0 to 59

**Example:**

Print Date()      ‘Return 5/12/11

Print DatePart(“m”,Now)      ‘Return 5

- 5. Day, Month and Year Function:** These functions return their date argument’s in the form of a day number, month number and year number.

Day () returns a number from 1 to 31 indicating the day portion of a given date.

**Syntax**

Day(Dateargument)

**Example:**

intDay = Day(Now) ‘ intDay = 12

Month() returns a number from 1 to 12 indicating the month portion of a given date.

**Syntax**

Month(Dateargument)

**Example:**

intMonth = Month(Now) ‘ intMonth = 8

Year() returns a number from 100 to 9999 indicating the year portion of a given date.

**Syntax**

Year(Dateargument)

**Example:**

intYear = Year(Now) ‘ intYear = 2001

- 6. Hour, Minute and Second Function:** These functions return their time argument's in the form of a hour number, minute number and second number.

Hour()Returns an integer specifying a whole number between 0 and 23 representing the hour of the day.

**Syntax**

Hour(Time argument)

**Example:**

intHour = Hour(Now) ‘ intHour = 21 (for 9 PM)

Minute Returns an integer specifying a whole number between 0 and 59 representing the minute of the hour.

**Syntax:**

Minute(Time argument)

**Example:**

intMinute = Minute(Now) ‘ intMinute = 15

Second Returns an integer specifying a whole number between 0 and 59 representing the second of the minute.

**Syntax**

Second(Time argument)

**Example:**

intSecond = Second(Now) ‘ intSecond = 20



**7. Weekday Function:** This function returns a number from 1 to 7 indicating the day of the week for a given date, where 1 is Sunday and 7 is Saturday.

**Syntax**

```
Weekday(Date)
```

**Example:**

```
intDOW = Weekday(Now)           'intDOW = 6
```

VB has a set of built-in constants that can be used instead of the hard-coded values 1 thru 7:

Constant Value

vbSunday 1

vbMonday 2

vbTuesday 3

vbWednesday 4

vbThursday 5

vbFriday 6

vbSaturday 7

**8. DateAdd() Function:** It Returns a date to which a specific interval has been added.

**Syntax**

```
DateAdd(Interval, Number date)
```

Where Interval is same as mentioned in datepart function. Number is the interval you want to add. Date is the processed date to which interval is being added.

**Example:**

```
Print DateAdd("m",4,Now)           'return 9/12/11
```

**9. DateDiff() Function:** DateDiff returns a Long data type value specifying the interval between the two values.

**Syntax**

```
DateDiff(Interval, date1, date2)
```

Where Interval is same as mentioned in datepart function Date1 & 2 are to determine the difference between two dates.

**Example:**

```
Print DateDiff("m",Now, # 9/12/11#)           'return 4
```

### LET US REVISE

- ✓ A procedure is a named unit of a group of statements that perform a well defined task.
- ✓ Advantages of procedures are : reusability, modularity and simplification of complex problems.
- ✓ There are three type of procedures: Sub procedures, Function Procedures and Property procedures.
- ✓ A sub procedure performs a task but does not return a value.
- ✓ Sub procedures are of two types: General Procedure and Event Procedure.
- ✓ A function performs a task and returns a value.
- ✓ Functions are of two types: Library Function and User Defined function.
- ✓ Property procedure contains a code which runs when a property of an object gets a new value or when the value is retrieved.
- ✓ Values are passed to procedures in two ways: Pass by value and Pass by reference.
- ✓ In pass by value method the original value remain unchanged by the changes made in the called procedure.
- ✓ In pass by reference method the original value gets changed through the change made in the procedure.
- ✓ A module is a code container in VB, which contains some procedures and functions.
- ✓ A form module is a module that stores all the procedures and declarations pertaining to a single form.
- ✓ A standard module stores general purpose code of application.
- ✓ A class module is a special code module that stores the blueprint for user created custom objects.
- ✓ Library function is used to perform various type of operations.
- ✓ The string function allows string manipulation.
- ✓ The numeric function allows number manipulation.
- ✓ The date and time function manipulates date and time.

### Assignments:

1. Write a program using function left & Mid\$ to scroll the given text in the label filed.
2. Write a program to check the given email address in the text box is a valid email address or not.
3. Write a program to split the given text which are separated by any delimiter by s/r = "VISUAL ; BASIC ; IS ; EASY ; LANGUAGE."

## Chapter-6

# VB Interface Style

### 6.1 INTRODUCTION

When you start to work on a VB Project you are no longer just a programmer - you are now a developer. Any project that you develop has to involve Users. They are the people who will sit in front of your interface for eight hours a day and decide if they like it or not. If they don't like it, no matter how efficient the code and how many millions of dollars were spent developing it, they will find ways to sabotage it.

As you develop more and more parts of the application, run them by the user to check for accuracy, completeness, clarity, etc. The user interface that you design is the most visible and perhaps the most important part of the application. The term commonly used for this type of interface is: GUI (Graphical User Interface). User interface refers to the fact that it is the part of the application between the user, in front of the screen, and the code behind the screen. How well the user can interact with the code depends on the quality of the interface.

### 6.2 INTERFACE STYLE

Interface is the visual part of the application, with which the user can interact. There are two main styles of user interface that you can create in VB. These interfaces are :

1. Single Document Interface (SDI)
2. Multiple Document Interface (MDI)
3. Explorer Style Interface

#### 6.2.1 Single Document Interface (SDI)

As the name suggests, it supports single document in its window. The moment you open a new document the previously opened documents is closed and new document is loaded in its window. Each window contains its own menu or tool bar, and does not have a "background" window or "parent" window containing its menu or tool bar. Example : Wordpad, Notepad. Applications which allow the editing of more than one document at a time, e.g. word processors, may therefore give the user the impression that more than one instance of an application is open.

## 6.2.2 Multiple Document Interface (MDI)

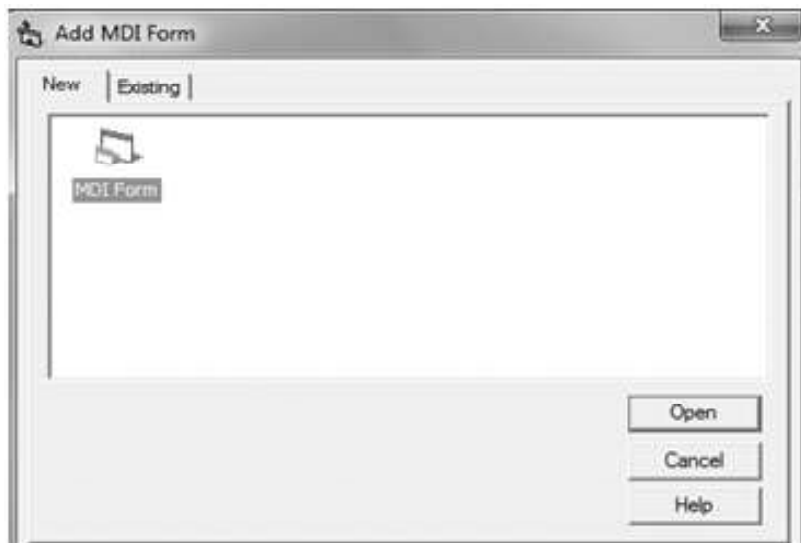
The Multiple Document Interface (MDI) was designed to simplify the exchange of information among documents, all under the same roof. The MDI allows you to create an application that maintains multiple forms within a single container form. Example : Microsoft Excel , Microsoft Word.

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. 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 but has only one MDI form. At run time, child forms are displayed within the workspace of the MDI parent form. When a child form is minimized, its icon appears within the workspace of the MDI form instead of on the taskbar,

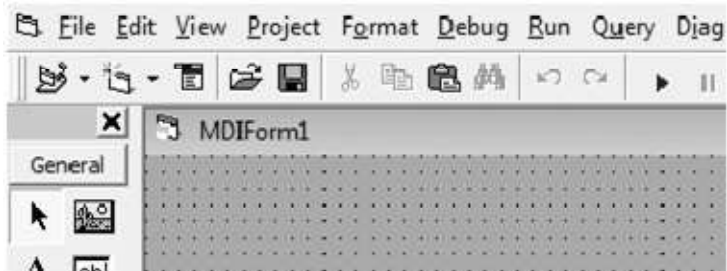
### 6.2.2.1 Creating MDI Form

To create an MDI application, follow these steps:

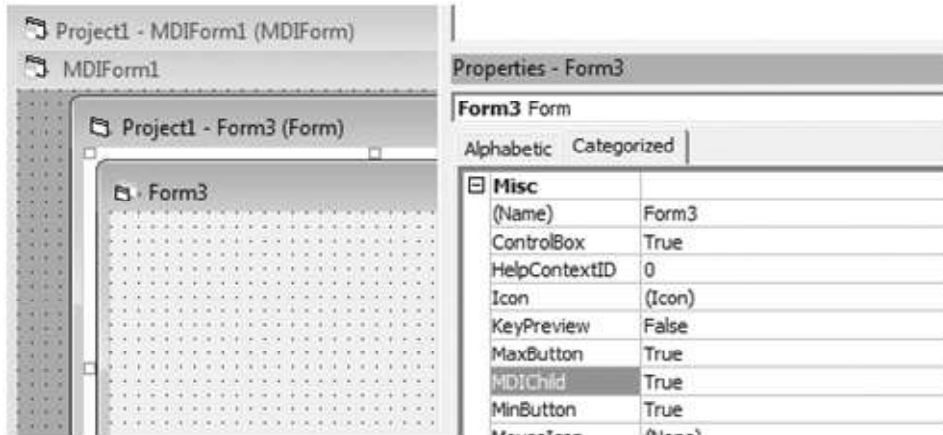
1. Start a new project and then choose Project → Add MDI Form to add the parent Form.



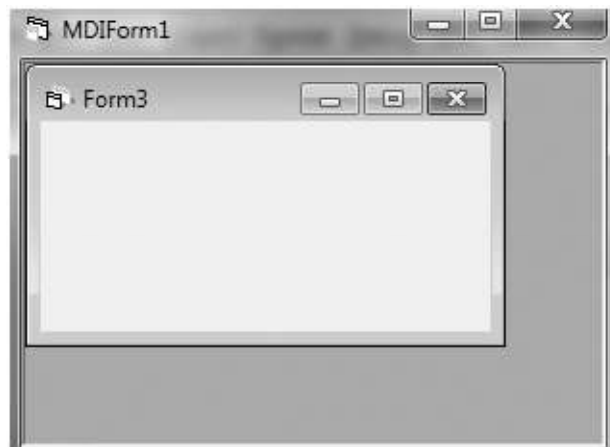
When you click on Open the window will appear as:



2. Choose Project → Add Form to add a SDI Form.
3. Make this Form as child of MDI Form by setting the MDI Child property of the SDI Form to True. Set the caption property to MDI Child window.



At runtime it will appear as:

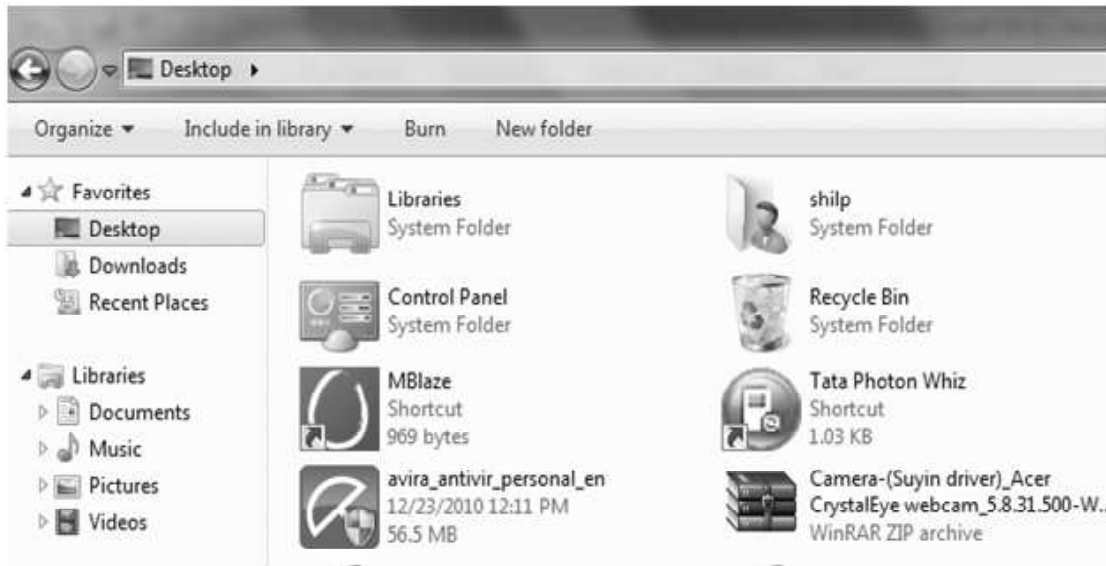


### 6.2.2.2 Change View From MDI to SDI

1. Choose Options from the Tools menu. The Options dialog box appears.
2. On the Advanced page, select the SDI Development Environment check box; then click OK. The Visual Basic IDE will reconfigure to the SDI view the next time you start a Visual Basic programming session.
3. Click OK then terminate and restart Visual Basic

### 6.2.3 Explorer Style Interface

The explorer style interface is a single window containing two panes or regions, usually consisting of a tree or hierarchical view on the left and a display area on the right, as in Microsoft Windows Explorer.

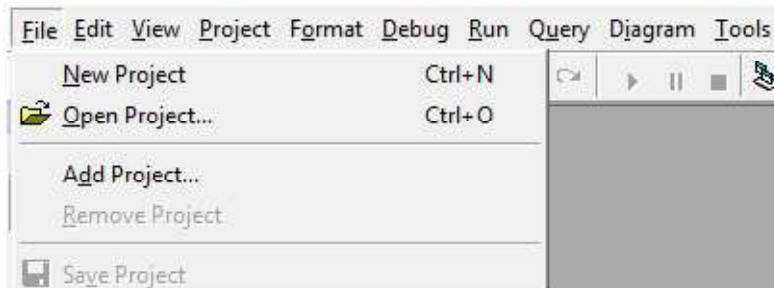


## 6.3 CREATING MENUS

Forms and controls comprise the basic interface for creating an application. You can make your application more user friendly by adding menus to them. Menus are a convenient and consistent way of grouping commands so that they become readily and easily accessible to users.

### 6.3.1 Menu Basics

As you have worked with menus of various GUI environments, you already know that there are generally two types of menus – horizontal menu i.e. Menu Bar and vertical menu i.e. popup menu. First we discuss, Menu bar.



Lets revise all the components of Menu

Menu Bar	This is the horizontal bar containing different menu options.
Menu Item	An option on a menu is called menu item.
Sub Menu	A menu attached to a menu is called sub menu.
Separator Bar	A bar on a menu that divides menu items into logical groups.
Access Key	A key combination used to open sub menu of a menu item.
Shortcut Key	A key combination used for directly invoking the command associated with a menu item.

### 6.3.2 Menu Title And Naming Guidelines

To make your code more reliable and easier to maintain, it's good to follow the naming conventions when setting the Name Property. To identify the menu object, you use prefix *mnu* for each control. For example: To represent File Menu name it *mnufile*.

To maintain consistency within an application, its good to follow following guidelines when assigning Caption for menu item:

1. Item name should be unique within a menu, but may repeat in different menus to represent similar actions.
2. Item name may be single, multiple or compound.
3. Each item name should have a unique access character for users who choose commands with keyboard.
4. The access character should be the first character of the name.
5. Keep the item name short.

### 6.3.3 Access Keys And Shortcut Keys

Access key and shortcut key provides keyboard access to menu command.

Access key allows user to open a menu by pressing ALT key and typing a designated letter. Once an application can open with a menu bar, the user can choose a control by pressing the letter assigned to it. An access key assignment appears as an underlying letter in menu control's caption. You have to type an ampersand (&) immediately in front of the letter you want to be the access key.

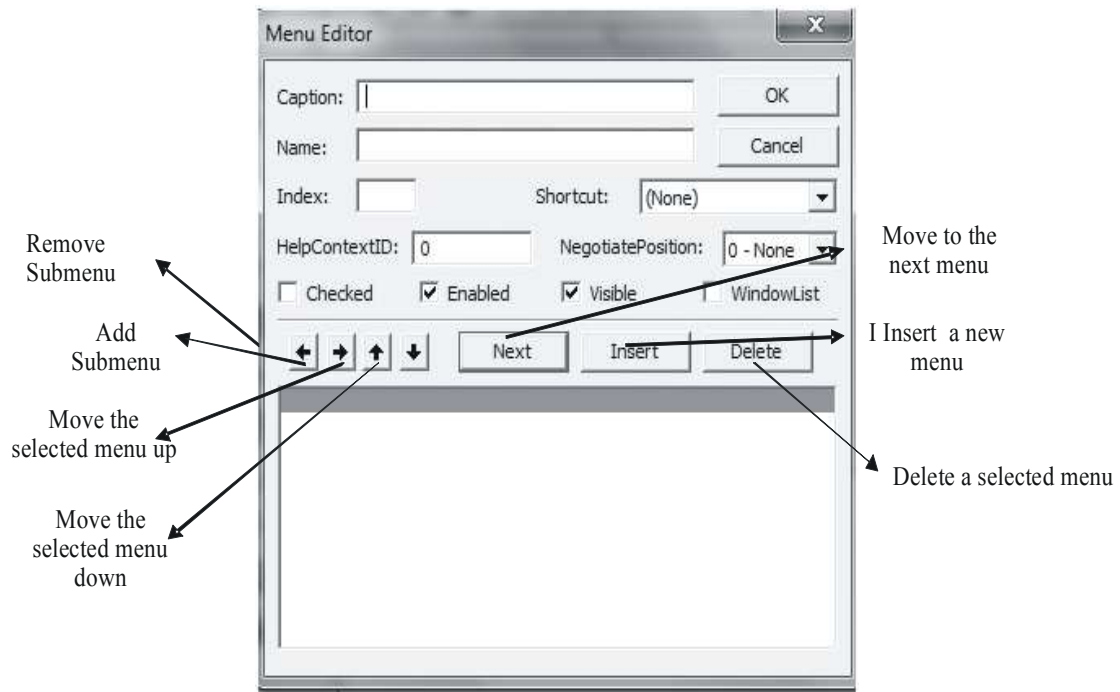
**For Example:** To make access key of Edit menu , write its caption property as &Edit. At run time you find Edit menu as Edit where E is underlined. To open Edit menu press ALT+E through keyboard.

Shortcut Keys directly activate a menu item i.e execute menu item command immediately when pressed. Command is activated by pressing CTRL key with a designated character. They appear on the menu to the right of the corresponding menu item.

## 6.4 DESIGNING MENUS

Visual Basic provides an easy way to create menus with the 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.

Basically, each menu item has a **Caption** and a **Name** property. Each item also exposes three Boolean properties, Enabled, Visible, and Checked, which you can set both at design time and at run time. At design time, you can assign the menu item a shortcut key so that your end users don't have to go through the menu system each time they want to execute a frequent command.

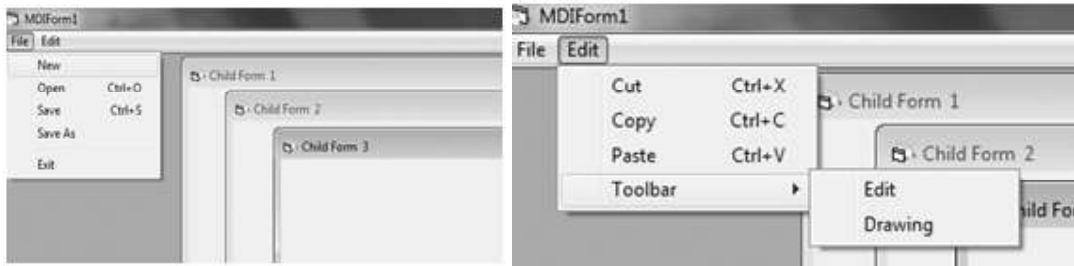


Menus that can grow or shrink at runtime is called dynamic menus.

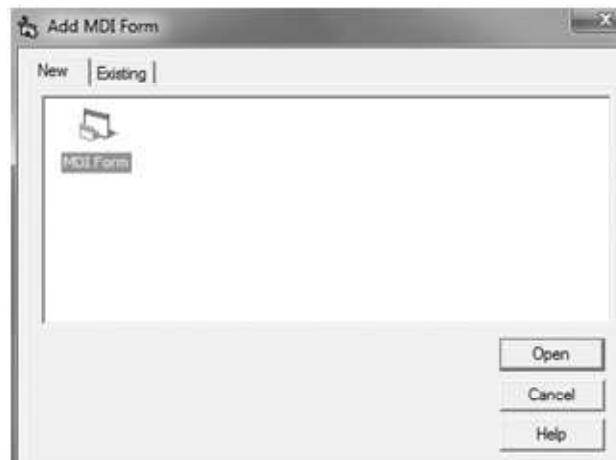


### 6.4.1 Steps To Create Menu

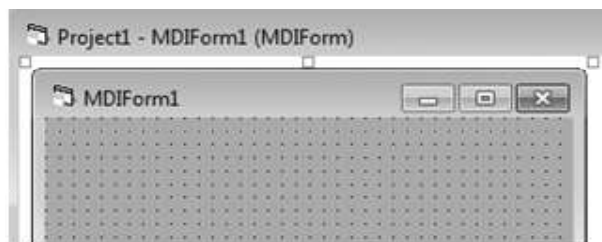
To learn designing of menu bar, you create a menu on MDI form with File and Edit option. Open a SDI form within MDI form like as follows:



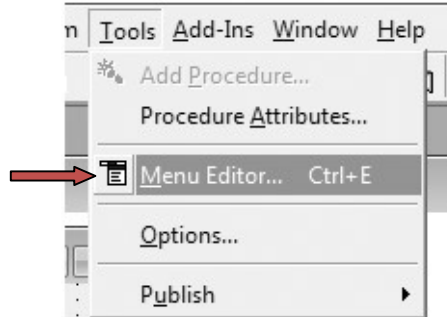
1. Open a VB standard EXE project.
2. Click on Project → Add MDI Form.



3. When you click on open MDI form will open as follows :

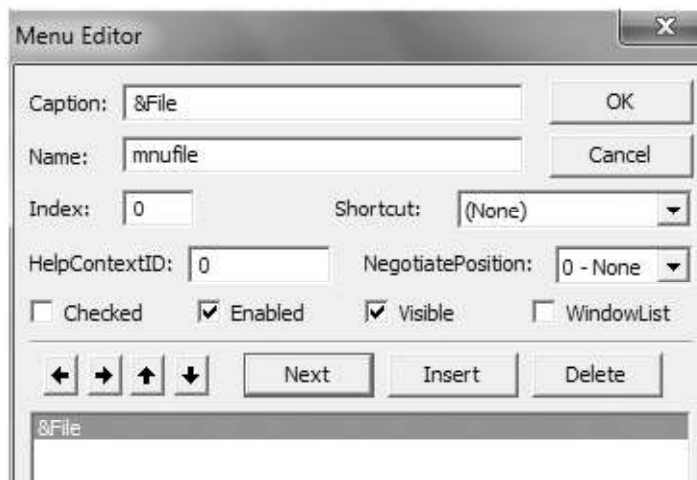


4. Click on Tool → Menu Editor.



This will open menu editor window.

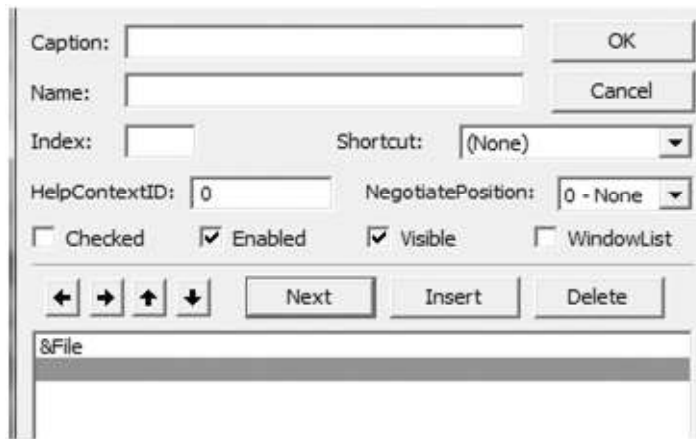
5. To make first menu item File, fill the boxes as mentioned




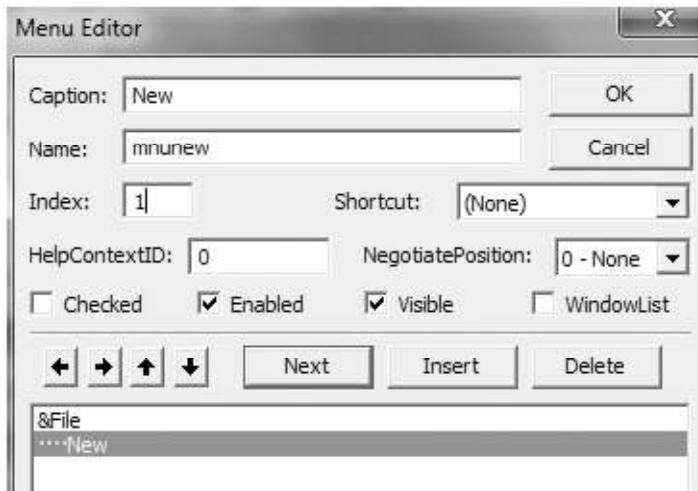
& in front of File in Caption indicates that file is a access key. When you click on OK button, it will displays as



6. To make sub menu under file button, first click on insert button. Then it will appear as



7. Now click on arrow  which is used to add submenu. This will create sub menu of file menu. Fill the editor as follows:



8. Now to write another item below New click on Next Button and fill the editor as

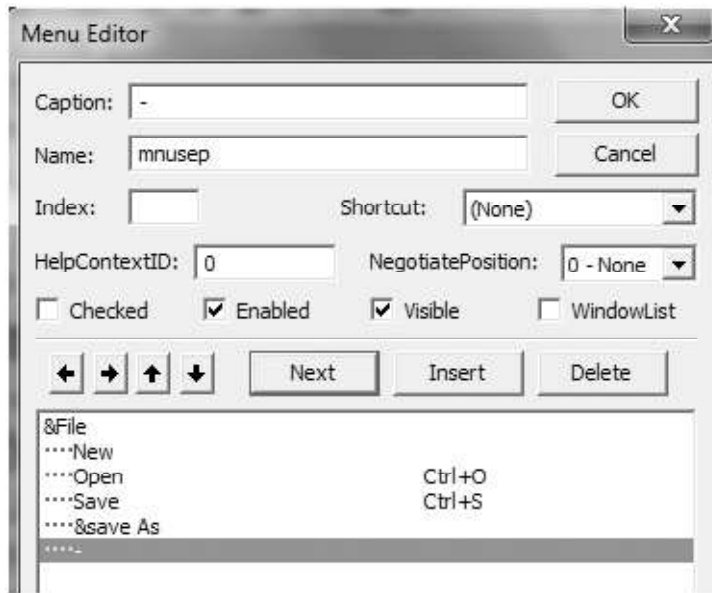


Here you also create shortcut key by selecting ctrl+O option from drop down list of shortcut.

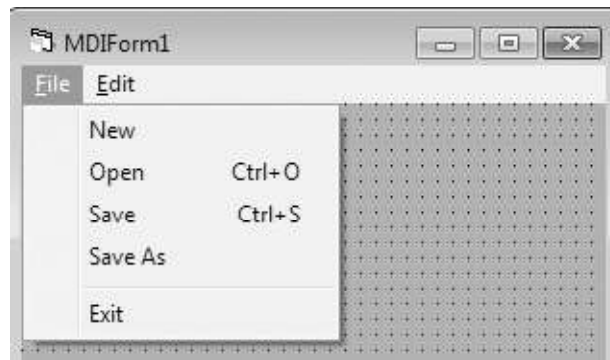
9. Repeat the steps 6 to design following sub menu of File:



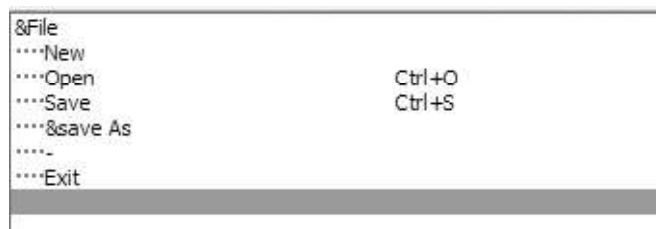
10. Now insert a separator bar, click on next button and type hyphen (-) in the caption box.



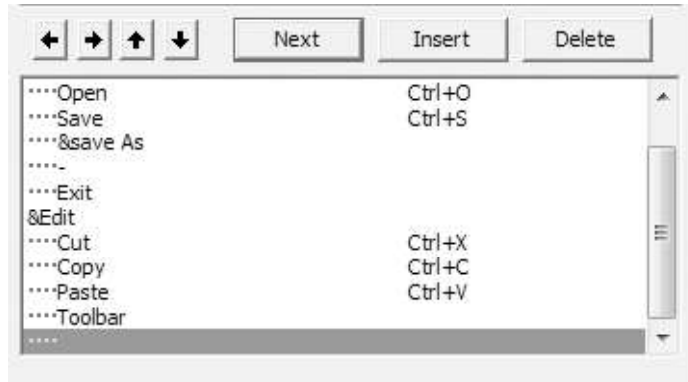
11. Now add exit control to your menu. File menu will appear as




12. Now design Edit menu, for this first click on  arrow to come on main menu, it will appear



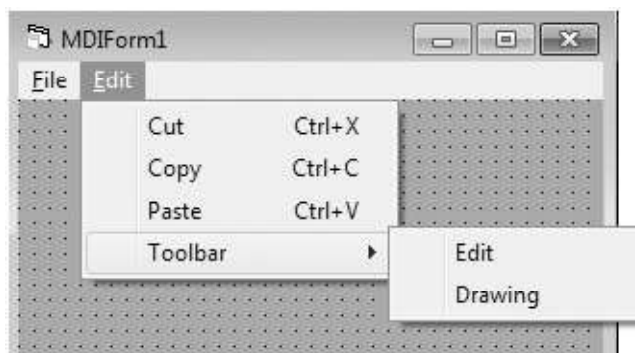
13. Repeat steps 3 to 6 to design the edit menu as follows:



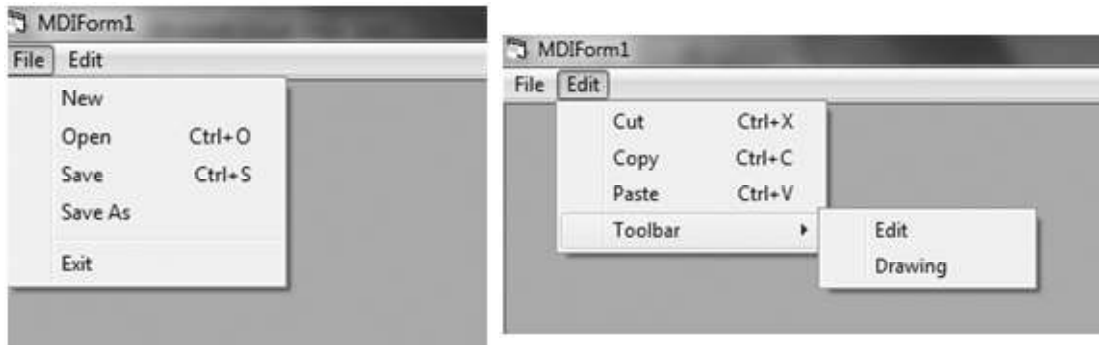
14. To create submenus of toolbar menu item, click on  arrow so that it comes like



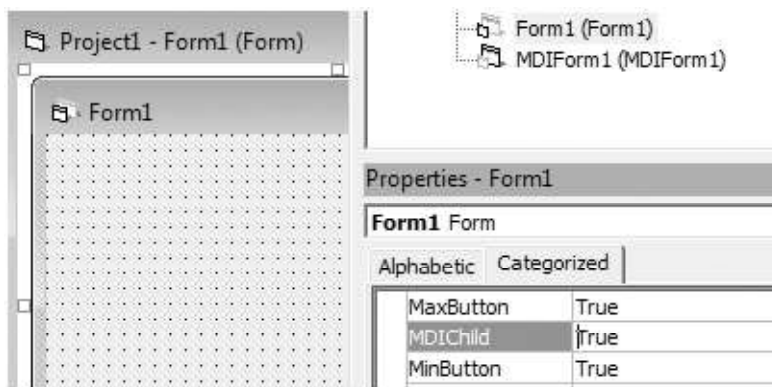
15. Now again repeat step 6 to create sub menu of toolbar. The Edit menu will appear as :



At runtime both menus are appear as



16. Now add code to New button , so that whenever you click on New menu item a new form will appear. For this first add an SDI form to your application by clicking on Project → Add Form.
17. Set Form1 MDIChild property to true.



18. Now go to MDI form and write following code on click event of New menu item:

```

mnunew
Click
Private Sub mnunew_Click()
    Dim newform As New Form1
    Static no As Byte
    no = no + 1
    newform.Caption = "Child Form " + Str(no)
End Sub

```

At runtime the window will be appear as:

### 6.4.2 Menu Control Array

A menu control array is a set of menu items on the same menu that share the same name and event procedure. Basically, a menu control array is a control array having elements as menu items. As control array all menu items share the common name and identified by its unique index number.

Uses

1. To create and add a new menu item to a menu at runtime.
2. To simplify code because common blocks of code can be used for all menu items.

Steps to create menu control array in Menu editor:

1. Select the form and open Menu Editor.
2. In caption text box, type the first menu title that you want to appear on menu bar.
3. In name text box, type the name of the first control. Leave the Index box empty.
4. At the next indentation level, create the menu item that will become the first element in the array by setting its Caption and Name.
5. Set the index for the first element in the array to 0.
6. Create a second menu item at the same level of indentation as the first.
7. Set the name of the second element to the same as first element and set its index to 1.
8. Repeat steps 5 – 7 for subsequent elements of the array.

## 6.5 POPUP MENUS

A popup menu is a floating menu that is displayed over a form, independent of the menu bar. The items displayed on the popup menu depend on where the pointer was located when the right mouse button was pressed. It is also called context menu. To display the popup menu, use the following **Syntax**: `PopupMenu menuname`

### 6.5.1 Creating Popup Menu

To create a popup menu, you first define a menu through Menu Editor and make sure that this menu will not display on menu bar. For this set its visible property to false and its submenu has true value. Then write the code for this that uses `PopupMenu` method.

Learn it with a help of an example : Create a text box with a popup menu that format the text of the textbox.

#### Steps

1. Open Project and click on Form1 and sets its caption as frmmenu and Name as Popup Menu.
2. Design the frmmenu as follows:  
Name the text box as txtsample.
3. Click on Tool → Menu Editor.



4. Design the menu as follows:

Make sure that visible property of Format menu item is unchecked and same property of rest of its submenu is checked.

5. Write the following code on click event of respective menu items:

6. To execute this menu floating, write the following code on Mouse Move event of text box:

At runtime it will appear as:

### LET US REVISE

- ✓ Interface is the visual part of the application with which user interact.
- ✓ VB supports mainly three type of interfaces – SDI, MDI and Explorer style interface.
- ✓ SDI supports single document in its window.
- ✓ MDI supports multiple documents at the same time.
- ✓ The explorer style interface is a single window containing two panes.
- ✓ A project can have only one MDI form.
- ✓ New form object can be created through New keyword.
- ✓ There are two type of menus : Menubar and popup menu.
- ✓ An option in a menu is called menu item.
- ✓ A bar in a menu that divides menu items in a logical group is called separator bar.
- ✓ A key combination used to directly execute the command is called shortcut key.
- ✓ Access key allows you to open a menu.
- ✓ A menu control array is a set of menu items on the same menu that share same name and event procedure.
- ✓ Menus that can grow or shrink at runtime are called dynamic menus.
- ✓ A popup menu is a floating menu that displayed over a form independent of the menu bar.

#### Assignment:

1. Create the following menu using menu editor. Also create popup menu for the same.

Format — **Bold**  
          — *Italic*  
          — Underline

Also perform the above menu operations on a given text to format the text.

## Chapter-7

# Error Handling and File Handling

### 7.1 INTRODUCTION

Although you have learnt a lot about VB programming but still you have to learn two very important things – the error handling and file handling. You deal with various functions, statements, properties and methods available in Visual Basic and the components used in Visual Basic expect to deal with certain types of data and behavior in your applications. For example, you have a program to divide two numbers. If instead of writing 200/10 you write 200/0 then what happens? To deal with such situations you use error handling.

If you want to be an expert VB programmer then you'll need to know a lot about how VB handles files - what is possible and what is not, plus how to do something in the smallest code or in the least amount of time. To do this you need to understand the file handling features which VB offers.

### 7.2 TYPES OF ERROR

An error is anything in the code that prevents a program from compiling and running correctly. An error is also called a bug. Some bugs are catastrophic in their effects, some are innocuous and others are so obscure that you will never discover them. They are classified in three sections:

1. Compile-time Error
2. Run-time Error
3. Logical Error

#### 7.2.1 Compile - Time Error

When a program is compiled, its source code is checked for whether it follows the programming language's rules or not. Errors that occur at this compile time are known as compile time error. There are two types of compile type error:

- (a) Syntax Errors:** It occurs when rules of programming languages are misused i.e when a grammatical rule of VB is violated. Example: Instead of writing (x-y) you write (x\_y).
- (b) Semantics Errors:** It occurs when statements are not meaningful. For example:

$x-y = z$

is a semantical error as an expression cannot come on the left side of an assignment statement.

When such compile time errors occur, VB raises an alert as:



### 7.2.2 Run-Time Error

Run-time errors are those that appear only after you compile and run your code. That is, errors that occur during the execution of a program are run time errors. These involve code that may appear to be correct in that it has no syntax errors, but that will not execute. These errors are harder to detect. Some run time errors stop the execution of the program which is then called program crashed or abnormally terminated.

For example, you might correctly write a line of code to open a file. But if the file is corrupted, the application cannot carry out the Open function, and it stops running. It appears as :



Most run time errors are easy to identify because program halts when it encounters an error and generates an alert as above figure. You can fix most run-time errors by rewriting the faulty code, and then recompiling and rerunning it.

### 7.2.3 Logical Errors

Sometimes, even if you don't encounter any error during compile time or run time and still your program does not provide the correct result. This is because of the wrong analysis of the programmer to solve the problem he is trying to solve. Such types of errors are called logical errors.

It appears once the application is in use. They are most often unwanted or unexpected results in response to user actions. For example- wrong parameters are passed. Logic errors are generally the hardest type to fix, since it is not always clear where they originate.

## 7.3 HANDLING ERRORS

Handling errors at run time, which will crash the program, requires special code so that errors can be trapped. In VB, it is possible to trap errors that normally cause the program to abort. Trapping error means that it is an indication to the operating system that you will handle the errors.

In an application, there may be a program logical error or it may be a data entry error on the part of the user. In the first case, you need to debug the program to fix the mistake. However, there is no way for you to anticipate the behavior of the end users of the application. If the user enters data you can't handle, you need to deal with the situation.

Dealing with errors at run-time is a two step process:

1. **Trap the Error:** Before you can deal with an error, you need to know about it. You use VB's On Error statement to setup an error trap.
2. **Handle the Error:** Code in your error handler may correct an error, ignore it, inform the user of the problem, or deal with it in some other way. You can examine the properties of the Err object to determine the nature of the error. Once the error has been dealt with, you use the Resume statement to return control to the regular flow of the code in the application

## 7.4 TRAP THE ERROR

Before you can do anything to deal with a run-time error, you need to capture the error. You use the On Error statement to enable an error trap. On Error will redirect the execution in the event of a run-time error. There are several forms of the On Error statement:

1. On Error GoTo 0
2. On Error Resume Next
3. On Error GoTo Line

### 7.4.1 On Error GoTo 0

On Error Goto 0 disables any error handler within the current procedure. A run-time error that occurs when no error handler is enabled or after an On Error Goto 0 is encountered will be handled using VB's default error handling logic.

**Example:** Write the following code to enter a number.

```
Dim num As Integer
On Error GoTo 0
    num = InputBox("enter a no.")
Exit Sub
MsgBox (" u entered a no.")
```

Instead of writing a number insert any other character and see how it works.

If the program encounters an error after this statement executes, it crashes.

#### 7.4.2 On Error Resume Next

On Error Goto 0 disables any error handler within the current procedure. A run-time error that occurs when no error handler is enabled or after an On Error Goto 0 is encountered will be handled using VB's default error handling logic. Execute the following code:

```
Dim num As Integer
On Error Resume Next
num = InputBox("enter a no.")
MsgBox (" u entered a no.")
Exit Sub
End Sub
```

It makes the program ignore errors that means when it encounters an error, the program continues execution after the statement that caused the error.

#### 7.4.3 On Error GoTo Line

This form of the On Error statement redirects program execution to the line label specified. When you use this form of On Error, a block of error handling code is constructed following the label. Execute the following code :

```
Dim num As Integer
On Error GoTo typehandler
    num = InputBox("enter a no.")
Exit Sub
typehandler:
    MsgBox ("numbers do not have character")
End Sub
```

The On Error GoTo Line statement registers a new error handler. If a program encounters an error, it passes control to the error handler beginning at the indicated line number or label. An error handling routine is not a Sub procedure or Function procedure. It is a section of code marked by a line label or number

You have done the same example with all three ways to trap an error, so you can easily judge the difference between all.

## 7.5 HANDLE THE ERROR

Trapping an error using the On Error statement is only the first step in dealing with run-time errors in your code. You must also deal with the error in some way, even if the error handling code is as simple as ignoring the error or displaying a message for the user.

The first step in handling an error is determining the nature of the error. This is accomplished by examining the properties of Visual Basic's Err object. The Err object includes the following properties:

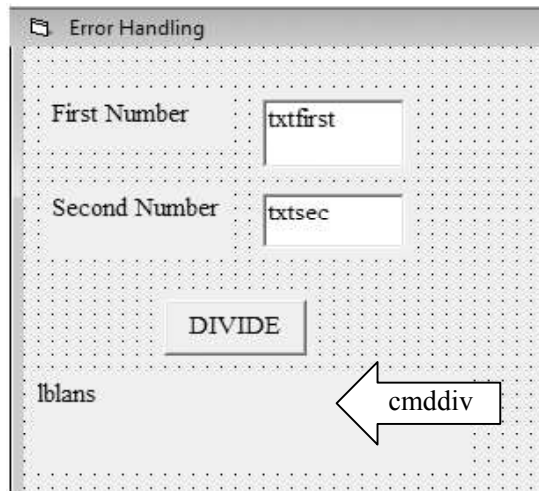
- **Number:** This is the error number that was raised.
- **Description:** This contains a descriptive message about the error. Depending on the error, the description may or may not be useful.
- **Source:** The Source property tells you what object generated the error.

When a run time error occurs, the properties of the Err object are filled with information that uniquely identifies the error and information that can be used to handle it. To find out which error has occurred, you have to use the Err.Number property. Following list gives you some sample error numbers and corresponding errors:

Error code	Error message
-----	-----
3	Return without GoSub
5	Invalid procedure call
6	Overflow
7	Out of memory
9	Subscript out of range
10	Duplicate definition (versions 5.0 and 7.0)
10	This array is fixed or temporarily locked (version 97)
11	Division by zero
13	Type mismatch
14	Out of string space
16	String formula too complex (versions 5.0 and 7.0)
16	Expression too complex (version 97)
17	Can't perform requested operation
18	User interrupt occurred
20	Resume without error

**Example:** WAP to divide first number by the second number. Use Err object to display different messages for different errors.

Design the form as follows:



Change the Name property of different controls with the text displayed in it and clears the boxes. Write the following code on click event of cmddiv button:

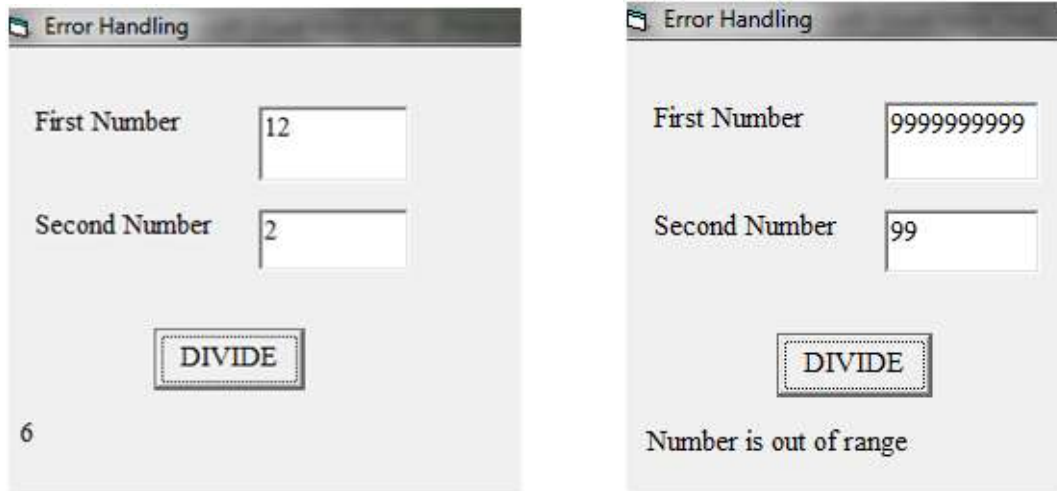
```

cmddiv Click
Private Sub cmddiv_Click()
    Dim fnumber As Integer
    Dim snumber As Integer
    Dim ans As Single
    On Error GoTo ErrorHandler
    fnumber = Val(txtfirst.Text)
    snumber = Val(txtsec.Text)
    ans = fnumber / snumber
    lblans.Caption = ans
    Exit Sub

ErrorHandler:
    Select Case Err.number
    Case 6
        lblans.Caption = " Number is out of range"
    Case 11
        lblans.Caption = " Cannot divide by 0"
    Case 13
        lblans.Caption = "Number do not contain characters"
    End Select
End Sub

```

At run time it appear as :



### 7.5.1 Leaving Error Handlers

There are several ways a program can leave error handling code and return to normal execution. These are:

- **Resume:** The Resume statement tells VB to continue execution with the line that generated the error.
- **Resume Next:** Resume Next instructs Visual Basic to continue execution with the line following the line that generated the error. This allows you to skip the offending code.
- **Resume label:** This allows you to redirect execution to any label within the current procedure. The label may be a location that contains special code to handle the error, an exit point that performs clean up operations, or any other point you choose.
- **Exit:** You can use Exit Sub, Exit Function, or Exit Property to break out of the current procedure and continue execution at whatever point you were at when the procedure was called.
- **End:** This is not recommended, but you can use the End statement to immediately terminate your application. Remember that if you use End, your application is forcibly terminated. No Unload, QueryUnload, or Terminate event procedures will be fired. This is the coding equivalent of a gunshot to the head for your application.

### 7.5.2 Error Object

Errors can be categorized into various groups such as

- **General File Errors:** Errors pertaining to file related operations such as File not found or File already exists etc.



- **Physical Media Errors:** Errors related to storage media such as Disk full or Disk not ready etc.
- **Program Code Errors:** Errors occurring in your code such as syntax errors or logical errors.
- **Database Errors:** Errors that occur when your code tries to connect to and access a database using a provider.

The mechanism of trapping and handling errors that you have covered so far can trap all type of errors but provider specific database errors. To trap provider specific database error , you need to use Error Object.

An Error Object contains details about data access errors pertaining to a single operation involving the provider.

Any operation involving ActiveX Data Object(ADO) objects can generate one or more provider errors. As each error occurs, one or more error objects are placed in the Error collection of the connection object. When another ADO operation generates an error , the Error collection is cleared and new set of Error objects is placed in the Errors collection.

There is a difference between Err Objects and Error Objects. The Err Object is an object that keeps track of all general type of errors where as Error Object contains details about data access errors pertaining to a single operation involving the provider.

## 7.6 FILE HANDLING

As far as Visual Basic is concerned, there are three modes in which a file can be accessed:

1. Text Mode (Sequential Mode)
2. Binary Mode
3. Random Access Mode

In the **Text Mode**, data is ALWAYS written and retrieved as CHARACTERS. Hence, any number written in this mode will result in the ASCII Value of the number being stored.

For Example, The Number 17 is stored as two separate characters “1” and “7”.

Which means that 17 is stored as [ 49 55 ] and not as [ 17 ].

In the **Binary Mode**, everything is written and retrieved as a Number. Hence, The Number 17 Will be stored as [ 17 ] in this mode and characters will be represented by their ASCII Value .

One major difference between Text Files and Binary Files is that Text Files support Sequential Reading and Writing. This means that we cannot read or write from a particular point in a file. The only way of doing this is to read through all the other entries until you reach the point where you want to start reading.

Binary Mode allows us to write and read anywhere in the file. For ex- we can read data directly from the 56th Byte of the file, instead of reading all the bytes one by one till we reach 56.

Just like the Binary Mode, the **Random Access Mode** allows us to gain instant access to any piece of information lying anywhere in the file. For example, if we need to store a few names in the file Random Access Mode requires us to mention the length of the 'Names' Field. Some Names might not fit and for the shorter names the space is inefficiently used. Random Access Mode allows us to read or write data at a particular **record** position rather than a **byte position** like in Binary Mode.

**The File System Object (FSO)** enables you to manipulate the files, folders and drives as well as read and write to sequential files. Before using the FSO, you have to add the "**Microsoft Scripting Runtime Library**" to the current project by selecting "**Project**", "**References**". Alternatively you can use the **CreateObject function** to create the reference at run-time.

There are five types of File System Object.

1. File.
2. Folder.
3. Drive.
4. TextStream.
5. Random Access Files.

The FileSystemObject is used to manipulate the files, folders and directories. Some of the common methods available to the FileSystemObject are –

CreateTextFile	Used to create a text file
DeleteFile	Used to delete a file
CopyFile	Used to copy an existing file.
OpenTextFile	Used to open an existing text file

Files can be opened in three modes in Sequential File Handling i.e

Reading -> Used to Read From a File.

Writing -> Used to Write to a File.

Appending -> Used to append the existing File.

**Example:** Make a text box on a form and create a text file to write, read and update data in the file.

*General Declaration*

```
Dim fs As New FileSystemObject
```

```
Dim fl As File
```

```
Private Sub Cmdsubmit_Click()
```

```
fs.CreateTextFile ("abc.txt")
```

```
fs.OpenTextFile("abc.txt", ForWriting).WriteLine (Text1.Text)
Text1.Text = ""
MsgBox ("data written")
End Sub
```

```
Private Sub cmdupdate_Click()
fs.OpenTextFile("abc.txt", ForAppending).WriteLine (Text1.Text)
Text1.Text = ""
MsgBox ("data updated")
End Sub
```

```
Private Sub cmdread_Click()
Text1.Text = fs.OpenTextFile("abc.txt", ForReading).ReadAll()
End Sub
```

```
Private Sub cmdclear_Click()
Text1.Text = ""
End Sub
```

### LET US REVISE

- ✓ Errors are also called bug.
- ✓ There are three types of errors – Compile time, run time error and logical error.
- ✓ Trapping error means that program is going to handle the errors.
- ✓ On Error statement is used to register error handling code.
- ✓ On Error GoTo 0 displays any enabled error handler.
- ✓ On Error GoTo line statement passes control to the error handler beginning at the indicated line number.
- ✓ On Error resume next ignores the program error.
- ✓ Err object number property gives the error identification number.
- ✓ The resume statement continues the execution by repeating the statement that causes the error.
- ✓ An error object contains details about data access error pertaining to a single operation involving the provider.

- ✓ There are three modes in which a file can be accessed : Text mode, binary mode and random access mode.
- ✓ In the **Text Mode**, data is always written and retrieved as characters.
- ✓ In the **Binary Mode**, everything is written and retrieved as a Number
- ✓ Text Files support Sequential Reading and Writing whereas Binary Mode allows us to write and read anywhere in the file.
- ✓ The **Random Access Mode** allows us to gain instant access to any piece of information lying anywhere in the file.
- ✓ The File System Object (FSO) enables you to manipulate the files, folders and drives as well as read and write to sequential files.

## Chapter-8

# Database Connectivity and Visual Database Tools

### 8.1 INTRODUCTION

Databases (DBs) are the systems that contain many different objects which work together to facilitate fast and efficient access to the data. You can use many type of DB as backend with you application. Some of these DB are Microsoft Access, Microsoft Foxpro, Oracel , MS SQL Server etc. These DBs are categorized as :

1. **Local Database:** DB that can access directly from VB through VB's database jet engines.
2. **Remote Database:** DB that cannot be accessed by using VB's standard DB access capabilities.

You can access both local and remote DBs using the same Db access controls and objects known as ActiveX Data Object (ADO).

**ADO** is an application program interface from Microsoft that lets a programmer writing windows applications, get access to a relational and non relational DB from both Microsoft and other Db providers. In this chapter, you will learn how you can access and manipulate DB using ADO data controls in VB.

### 8.2 DB CONCEPTS

Before working with DB , you must aware of some basic concepts of DB. These concepts are:

1. **Database:** A DB represents a set of data related to a particular topic. A DB contains tables and can also contain queries and table relationships.
2. **Table:** A table is a collection of data arranged in rows and columns. Each column would contain a certain type of information and each row will contain all the information about a specific author.
3. **Record:** Record is a complete set of information. It is a row of data in a database table consisting of a single value from each column of data in the table.
4. **Recordset:** It is a logical set of records. Recordset creates a temporary object which contains data from one or more tables in the DB. A recordset is not the DB itself; it is just a working copy of some parts of DB tables.

You can manipulate the contents of recordset through VB code but the actual DB file will not be changed until you or user take action to save the changes. There are five type of recordsets–

- (a) **Table-type Recordset:** It represents a complete table from a DB. You can use it to add, change or delete records. This is the simplest concept but it is not the best choice in many applications.
- (b) **Dynaset-type Recordset:** It represents the result of a query that can have updatable records. You can use it to add, change or delete records. It can contain fields from one or more tables. This provides the worse performance than a table-type recordset.
- (c) **Snapshot-type Recordset:** It represents a read-only set of records that you can use to find data or generate reports. It can contain fields from one or more tables but they cannot be updated. This recordset uses minimum resources and provides fast performance.
- (d) **Forward-only-type Recordset:** This is identical to a snapshot recordset except that no cursor is provided. Cursor indicates the current position in the recordset. You can only scroll forward through records. This improves performance in situations where you only need to make a single pass through a recordset.
- (e) **Dynamic-type Recordset:** This recordset stores a query result set from one or more base tables in which you can add, change or delete records from row returning query.

Out of above five, mostly first three are used.

### 8.3 DATA ACCESS MECHANISM

In VB you can access DB through three different data access mechanism. These mechanisms are:

1. **Data Access Object (DAO):** It was first object oriented interface that exposed the Microsoft Jet Database Engine used by Microsoft Access and allowed VB developers to directly connect to Access DB through open DB Connectivity, commonly known as ODBC. ODBC refers to a standard protocol that permits applications to connect to a variety of external DB servers or file. It is best suited for either single system applications or for small deployments.
2. **Remote Data Object (RDO):** It is an interface to ODBC combined with easy to use style of DAO. RDO is limited; it doesn't access Jet DB very well and can access relational DB through existing ODBC drivers.
3. **ActiveX Data Object (ADO):** It is successor of DAO and RDO. This technology allows users to access data easily from many existing databases such as Access or Paradox or from ODBC compliant databases like Oracle or MS SQL Server. ADO is quite simple and allows programmers to provide flexible database front ends to users that are reliable and include many features.

### 8.3.1 Difference Between DAO, RDO and ADO

DAO (Data access object)	RDO (Remote Data Object)	ADO (ActiveX data object)
DAO is used for accessing data before ADO. It was used for database installed on same system where the application resides.	RDO was used for accessing remote data.	Ado most powerful till date it is nothing but combination of both DAO and RDO
It can be used to access local database.	RDO are used to access remote databases.	ADO can be used for different data bases i.e. For both local & remote database.
It uses Jet Database Engine.	It uses ODBC drivers	It uses OLEDB driver
DAO is used for small data base access.		ADO is used to handle a large number of records.

## 8.4 DB ENGINE

The DB Engine is the highest-level object in the DAO object model. It contains all other objects and collections. The DB object is the member of the DB collection of the default workspace object, which is member of the workspace collection of the DB engine object.

The **Workspace object** defines a session for a user based on user's permissions and allows managing the current sessions. It also contains open DB and offers mechanism for simultaneous transaction and for securing the application.

### 8.4.1 Microsoft Jet Database Engine

The Microsoft Jet Database Engine is a database engine on which several Microsoft products have been built. A database engine is the underlying component of a database, a collection of information stored on a computer in a systematic way. The first version of Jet was developed in 1992, consisting of three modules, which could be used to manipulate a database.

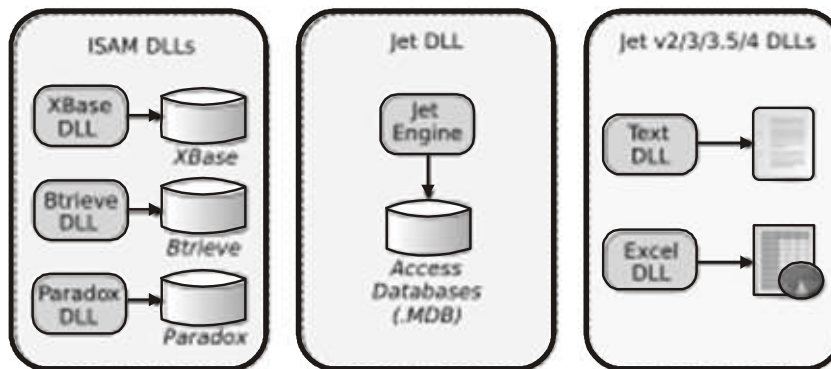
JET stands for *Joint Engine Technology*, sometimes being referred to as *Microsoft JET Engine* or simply *Jet*. Microsoft Access and Visual Basic use or have used Jet as their underlying database engine. The Jet engine can be said to be comprised of two broad components:- a Data Definition Language to create and edit the data structures, and a Data Manipulation Language used for adding, editing and deleting data, and for querying.

### Architecture

Jet allows the manipulation of a relational database and is part of a Relational Database Management System (RDBMS). It offers a single interface that other software can use to access Microsoft databases and provides support for security, referential integrity, transaction processing, indexing, record and page locking, and data replication. In later versions, the engine has been extended to be

able to run SQL queries, store character data in Unicode format, create database views and allow bi-directional replication with Microsoft SQL Server.

There are three modules to Jet: One is the *Native Jet ISAM Driver*, a dynamic link library (DLL) that can directly manipulate Microsoft Access database files (MDB) using Indexed Sequential Access Method (ISAM). Another one of the modules contains the *ISAM Drivers*, DLLs that allow access to a variety of ISAM databases, among them Xbase, Paradox, Btrieve and FoxPro, depending on the version of Jet. The final module is the *Data Access Objects (DAO) DLL*. DAO provides an API (Application Programming Interface) that allows programmers to access JET databases using any programming language.



#### 8.4.2 Open DB Connectivity (ODBC)

In computing, Open DB Connectivity (ODBC) provides a standard software interface for accessing and alerting the contents of relational and non-relational DB management system (DBMS). It was developed by SQL Access Group in 1992 in order to facilitate easier communication between applications and DBs across computing platforms. The designer of ODBC aimed to make it independent of programming languages, DB systems and operating systems.

The goal of ODBC aimed to make it possible to access any data from any application, regardless of which DBMS is handling the data. ODBC manages this by inserting a middle layer, called a DB driver, between an application and the DBMS. The purpose of this layer is to translate the applicant's data queries into commands that are understandable by DBMS.

An application can communicate through ODBC is referred to as ODBC compliant. Any ODBC compliant application can access any DBMS that has a corresponding driver. For the driver, the ODBC models allow for two different solutions, either having the driver reside on the client machine or as part of the server side solution.

Regardless of where the driver exists, the basic model is the same. The application sends ODBC commands to the driver, which then translates those commands into the native format of the DBMS. Once the DBMS has performed the query, it then sends those results back through ODBC driver which then translate them back into a standard format.



### 8.4.3 Object Link Embedding (OLE DB)

OLE DB is Microsoft's strategic low-level application program interface (API) for access to different data sources where OLE stands for "Object Link Embedding" and "DB" for database. OLE DB includes not only the Structured Query Language (SQL) capabilities of the Microsoft-sponsored standard data interface Open Database Connectivity (ODBC) but also includes access to data other than SQL data.

OLE DB is building on the success of ODBC by providing an open standard for accessing all kinds of data. Unlike ODBC but not imposes specific limitation on either the query syntax, or the structure of the data exposed.

As a design from Microsoft's Component Object Model (COM), OLE DB is a set of methods for reading and writing data. The objects in OLE DB consists mainly of a data source object, a session object, a command object, and a row set object. An application using OLE DB would use this request sequence:

1. Initialize OLE.
2. Connect to a data source.
3. Issue a command.
4. Process the results.
5. Release the data source object and uninitialize OLE.

OLE DB is, in turn, an open standard for providing data access. Various OLE DB data providers now exist. **Data Provider** is a control or mechanism that provides data for use by connecting to a source of data i.e a DB. Most commonly used data providers are –

Database	Data Provider
Access	Microsoft Jet DB engine/ Microsoft.Jet.OLEDB.4.0
Oracle	MSDAORD

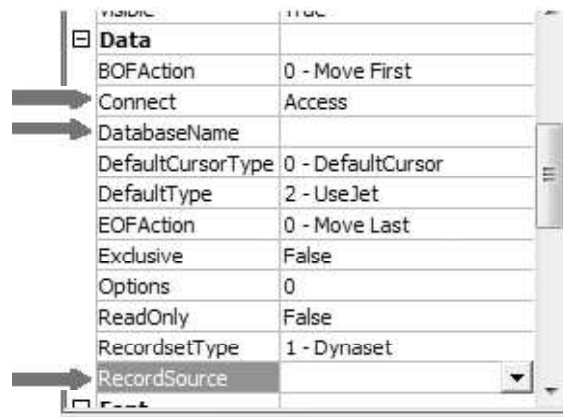
A programmer using ADO can connect to a data source through one of the existing OLE DB providers. The programmer can then manipulate this data by using the ADO object model.

## 8.5 VB DATA CONTROL

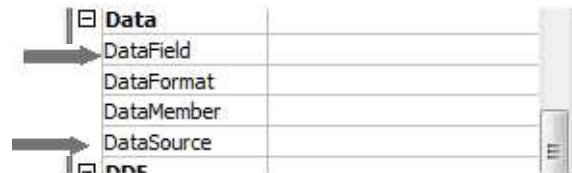
VB provides two controls – the Intrinsic Data Control and the ADO Data Control (ADO DC) i.e Bound Control that make the link to the DB file and which creates the Recordset that is exposed to the rest of the controls in the application. The two controls are identical in concept but differ in flexibility they offer to the programmer.

### 8.5.1 Data Control

Data Control is an older type of control used with DAO connections. The concept is simple, once you place the DAO control on your form. First set the **.connect** property of DB to tell which type of DB you are going to connect. Then set the **.Database** property tells which DB files to read. Then set **.Recordset** property tells it which table or query from the DB is to be open.



For any control which supports DB, bind that control with a field of DB. For this first set **.DataSource** property to the data control name and then **.Datafield** property to the specific field within the table.



Now, the data control provides access to data stored in the connected table of the DB.

### 8.5.2 Data Bound Controls

Data Bound Control is a new ADO control which makes use of all the ADO features. A bound control is a control that can provide access to a specific column in a data source through a data control. When a data control moves from one row to the next, all bound controls connected to data control changes to display data from columns in the current row. If user changes in a bound control and then move to a different row, the changes are automatically saved in the data source.

The data bound control has three properties - **.Datasource**, **.Datafield** and **.Datachanged**. The datasource and datafield properties are same as discussed in above section. **.Datachanged** property is set to true by VB when a value displayed in the control has changed. After setting the properties of data bound control, set the control to display the data of table by using **.Datasource** and **.Datafield** properties.

## 8.6 COMPANY DATABASE

Before learning DB connection, we first create a DB **Company** you use the same DB to establish connection by different means.

Now, make a simple DB Company with two table and query in Microsoft Access .

Design first table **Employee** as follows:

Field Name	Data Type	
ECode	Number	Primary Key
Ename	Text	Stores maximum 25 Characters
Ecity	Text	Stores maximum 25 characters
Hire_Date	Date/Time	Date of joining the company
Salary	Currency	Salary of employee in standard format
Department	Text	Department in which he joins

Insert the following data in Employee Table

ECode	Ename	Ecity	Hire_Date	Salary	Department
1001	Amit	Delhi	4/10/2001	55,560.00	Production
1002	Shilpi	Dehradun	9/19/2005	22,670.00	Sales
1003	Aaditya	Mumbai	12/6/2004	34,990.00	Production
1004	Aarya	Chennai	4/5/2009	20,000.00	Marketing
1005	Swadha	Lucknow	7/7/2001	45,550.00	Personnal

Design second table **Department** as:

Field Name	Data Type	
DeptNo	Number	Department Number
Dname	Text	Department name in 25 characters
Dhead	Number	Ecode of head of the department

Insert following data in the Department table

DeptNo	Dname	Dhead
11	Production	1001
12	Finance	1120
13	Sales	1002
14	Personnal	1110
15	Marketing	1004

One by one you will learn to connect this DB initially with Data control then with ADO DC and finally with ADODB.

## 8.7 DB CONNECTION BY DATA CONTROL

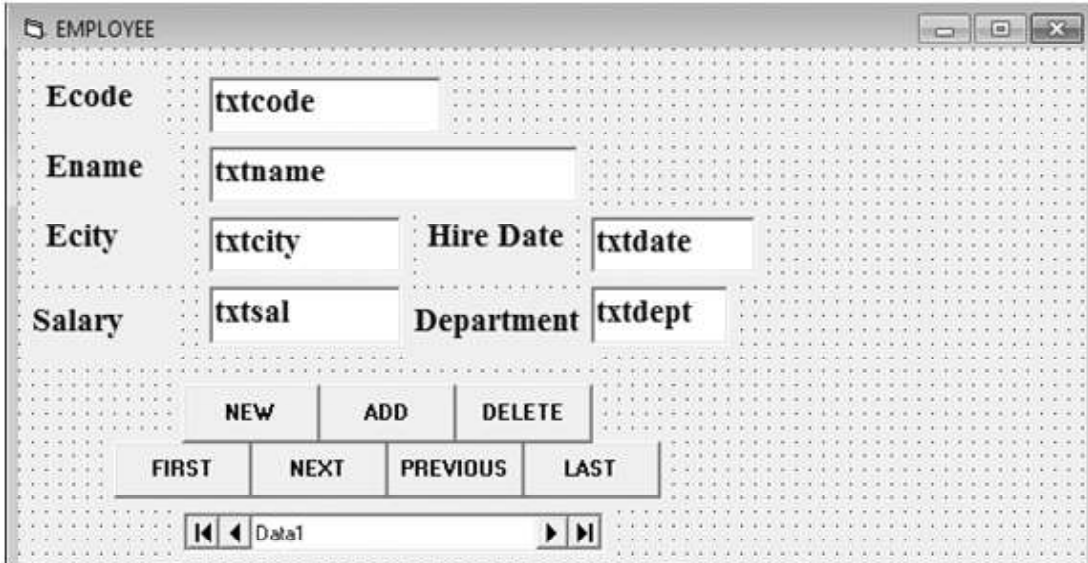
You start with the simplest type of connection object i.e data control. You connect the above DB with an interface of VB. And put all the important navigation buttons.

Steps to access Employee table of Company DB

1. **Design the form** which displays attributes of table as label and place a text box in front of each label to accept and display values from and to user. Save the textbox with the name displayed in the textbox make them empty. Make seven command buttons and name them as follows:

Control	Caption	Name Property
Command1	NEW	Cmdnew
Command2	ADD	Cmdadd
Command3	DELETE	Cmddel
Command4	FIRST	Cmdfirst
Command5	NEXT	Cmdnext
Command6	PREVIOUS	Cmdpre
Command7	LAST	cmdlast

2. Put data control  named Data1 on the form as follows.

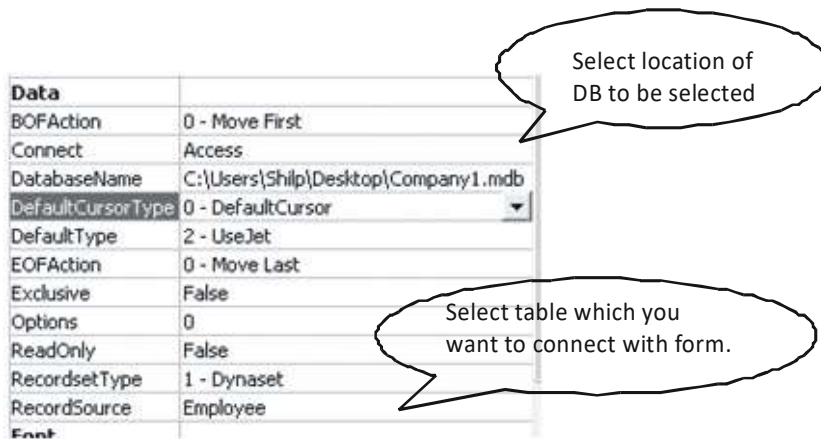


The screenshot shows a Visual Basic form titled "EMPLOYEE" with a grid background. It contains the following elements:

- Labels and text boxes: Ecode (txtcode), Ename (txtname), Ecity (txtcity), Hire Date (txtdate), Salary (txtsal), and Department (txtdept).
- Command buttons: NEW, ADD, DELETE, FIRST, NEXT, PREVIOUS, and LAST.
- Data control: A Data1 data control at the bottom with navigation arrows and the text "Data1".

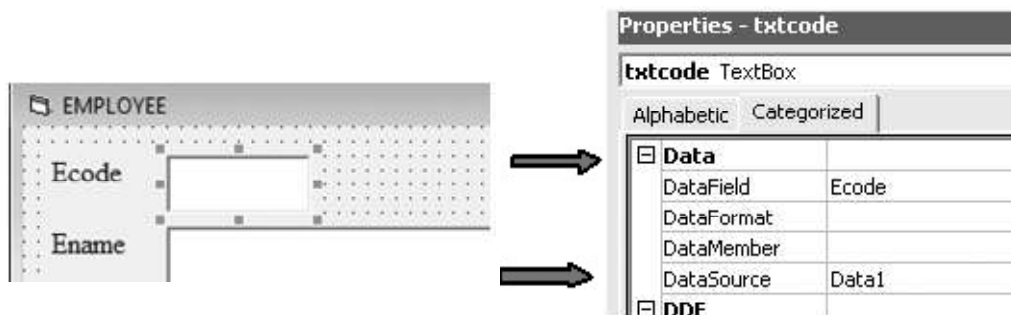
3. Select location of DB to be selected

To establish connection with data control, Set Data1 control's connect, DB name and Recordset property.



Select table which you want to connect with form.

4. Now, one by one **bind every text box** with their respective attribute name of MS-Access. For this select textbox, set its Data Source property to Data1 and select respective attribute from dropdown list of Data Field property.



5. **Navigating the Recordset.** Use the first, next, previous and last button to display data of the DB. The **first** button will take you to the very first record of the table. **Next** shows the followed data whereas **previous** displays the preceded data of the currently shown data of the table. **Last** button takes you to the last record of the table. Write the following code for navigation:

```
cmdlast Click
Private Sub cmdfirst_Click()
' Move to first record of the table

Data1.Recordset.BOF
' or you may apply Data1.Recordset.MoveFirst

End Sub

Private Sub cmdnext_Click()
'Move to the next record

Data1.Recordset.MoveNext

End Sub

Private Sub cmdpre_Click()
' Move to the previous record

Data1.Recordset.MovePrevious

End Sub

Private Sub cmdlast_Click()
' Move to the last record

Data1.Recordset.EOF
' or you may apply Data1.Recordset.MoveLast

End Sub
```

**BOF** stands for Beginning Of The Table and **EOF** stands for End Of The Table.

The screenshot shows a window titled "EMPLOYEE" with the following fields and values:

Ecode	1001		
Ename	Amit		
Ecity	Delh	Hire Date	4/10/2001
Salary	55,560	Department	Production

Below the fields are two rows of buttons:

- Row 1: NEW, ADD, DELEE
- Row 2: FIRST, NEXT, PREVIOUS, LAST

At the bottom is a data grid with a header "Data1" and navigation arrows.

6. The **New** Command button will clear all textbox to make new entry. Write the following code at click event of new button:

```
Private Sub cmdnew_Click()  
txtcode.Text = ""  
txtname.Text = ""  
txtcity.Text = ""  
txtdate.Text = ""  
txtsal.Text = ""  
txtdept.Text = ""  
  
End Sub
```

When you click on new button screen will display as

Now, you are ready to write new record in the table.

7. The **Add** command button will add the entered record into the selected table of the database. This will execute by writing:

```

cmdadd
Private Sub cmdadd_Click()
    Data1.Recordset.AddNew
End Sub

```

Write a new entry in the blank textboxes as



The screenshot shows a form titled "EMPLOYEE" with the following fields and values:

- Ecode: 1006
- Ename: Dinesh
- Ecity: Delhi
- Hire Date: 14/9/1999
- Salary: 85,500
- Department: Personnal

Below the fields are buttons for "NEW", "ADD", and "DELEE". Below these are buttons for "FIRST", "NEXT", "PREVIOUS", and "LAST". At the bottom is a Data1 grid control with navigation arrows.

When you click on the add button the entered data is saved into the Employee DB.

Employee						
ECode	Ename	Ecity	Hire_Date	Salary	Department	
1001	Amit	Delhi	4/10/2001	55,560.00	Production	
1002	Shilpi	Dehradun	9/19/2005	22,670.00	Sales	
1003	Aaditya	Mumbai	12/6/2004	34,990.00	Production	
1004	Aarya	Chennai	4/5/2009	20,000.00	Marketing	
1005	Swadha	Lucknow	7/7/2001	45,550.00	Personnal	
1006	Dinesh	Delhi	9/14/1999	85,500.00	Personnal	
*						

8. The **Delete** button will remove the currently shown record from the table. This will done by writing:

```

cmdDel
Click
Private Sub cmdDel_Click()
    Data1.Recordset.Delete
End Sub

```

With the help of navigation button select the desired record which you want to delete. Let it be:

The screenshot shows a form titled 'EMPLOYEE' with the following fields and values:

Ecode	1005		
Ename	Swadha		
Ecity	Lucknow	Hire Date	7/7/2001
Salary	45,550	Department	Personnal

Below the fields are two rows of buttons:

- Row 1: NEW, ADD, DELEE
- Row 2: FIRST, NEXT, PREVIOUS, LAST

At the bottom, there is a field labeled 'Data1' with navigation arrows on either side.

When you click on delete button above record is removed from the Table.

ECode	Ename	Ecity	Hire_Date	Salary	Department
1001	Amit	Delhi	4/10/2001	55,560.00	Production
1002	Shilpi	Dehradun	9/19/2005	22,670.00	Sales
1003	Aaditya	Mumbai	12/6/2004	34,990.00	Production
1004	Aarya	Chennai	4/5/2009	20,000.00	Marketing
1006	Dinesh	Delhi	9/14/1999	85,500.00	Personnal
*					

**Note:** Sometime you find it difficult to attach your DB with form. In such case , first convert your DB in windows 1997 format and then try to connect.

## 8.8 DB CONNECTIVITY BY ADO DC

ADO is an object oriented programming interface. This is also called Universal data Access as it access different kinds of data using OLE DB as a data provider and ADO as data access technology. ADO enables us to write an application to access and manipulate data in a DB server through an

OLE DB provider.

OLE DB is an the underlying system service , that is, it is a set of interfaces that provide applications with uniform access to data stored in diverse information sources of data sources, for the programmer using ADO. OLE DB can access all type of DB s in the same manner.

You can create VB application using ADO Data Control (ADO DC) in two ways:

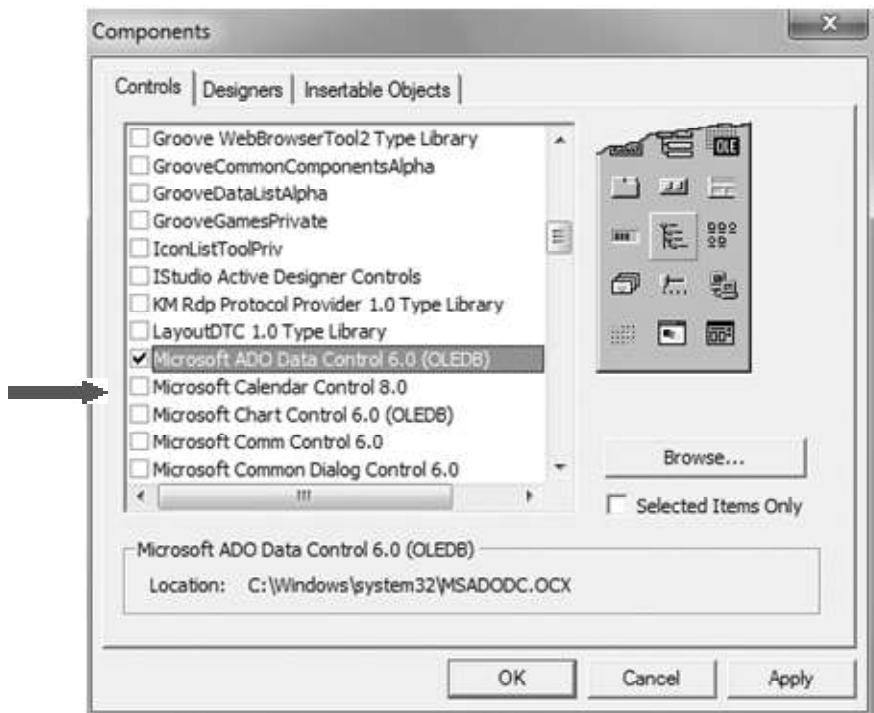
1. By employing ADO DC interactively using VB wizard.
2. Writing code for ADO. This is called ADO DB

### Steps to access DB using ADO DC

Let us, again connect Employee table of Company DB with the help of ADO DC. Initially you have to add ADO DC on your tool box of VB.

Adding ADO Control in toolbox

1. Open VB standard EXE project.
2. Click on Project Menu and click on Components to open.
3. Select Microsoft ADO Data Control (OLEDB) and click on apply then Ok.



This will add ADO DC in the toolbox.

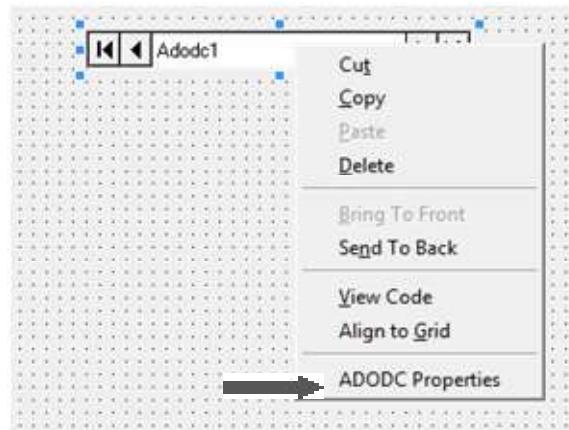


Design the form

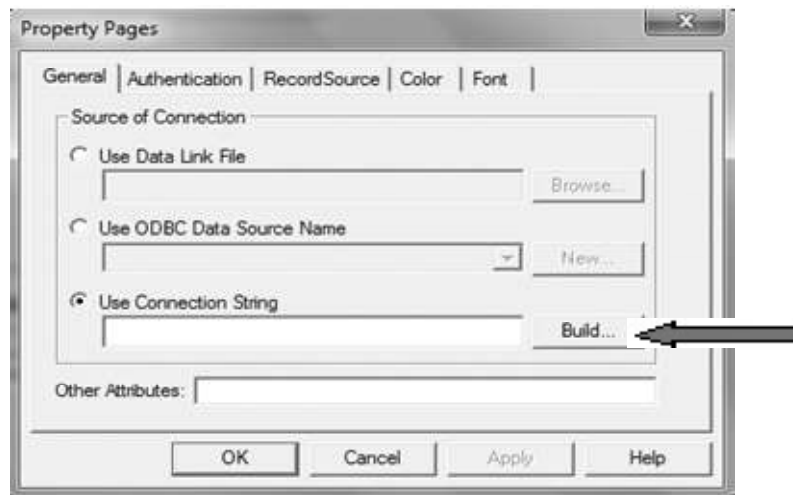
4. Design the form same as you designed in above section (Connection through DAO).
5. Place ADO DC on form.

### Setting Connection

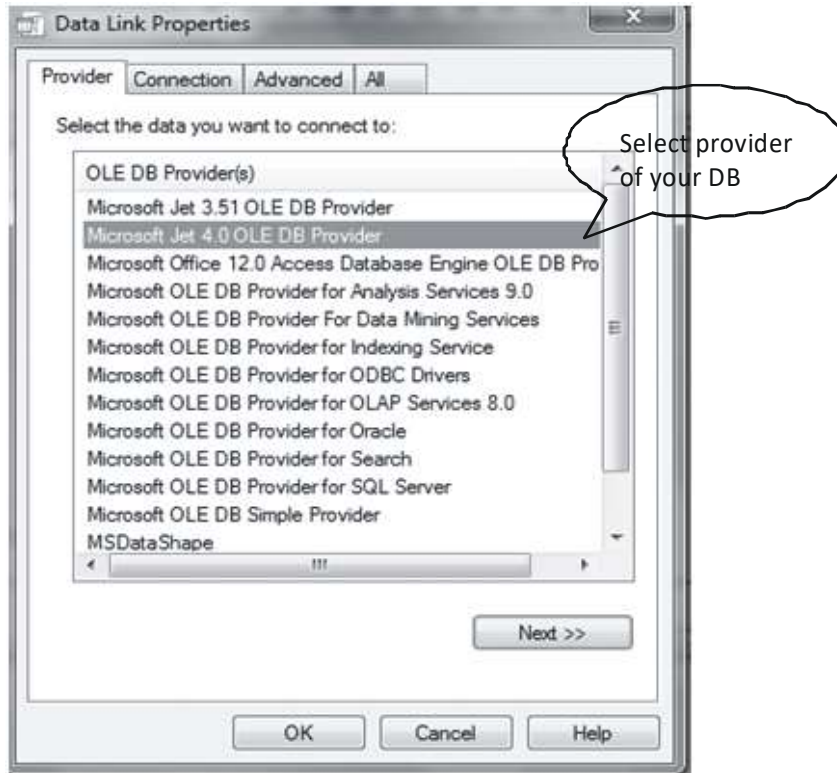
6. Right click on ADODC ® select Properties. This will open property page window of ADO DC



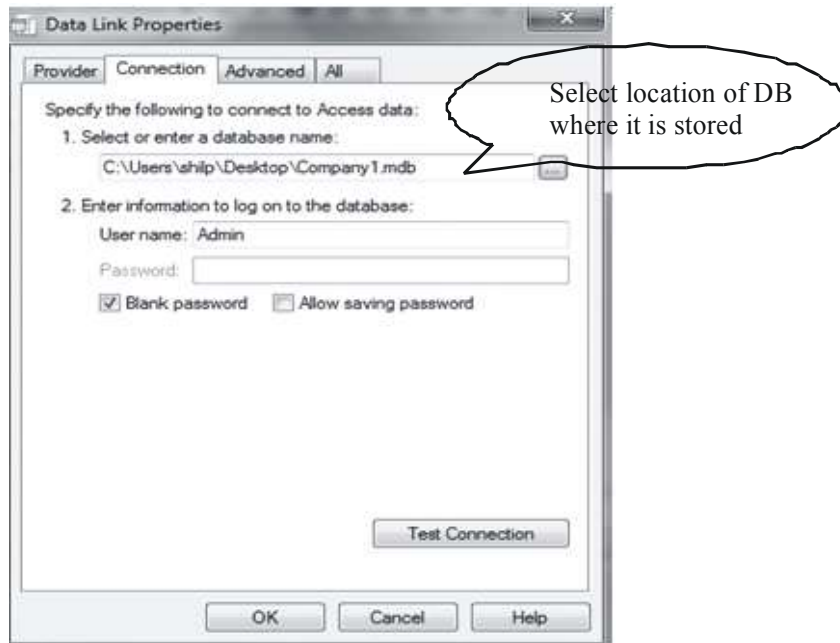
7. Select General Tab, select Use Connection String



8. Click on Built button of connecting string. This will open Data Link Property window.

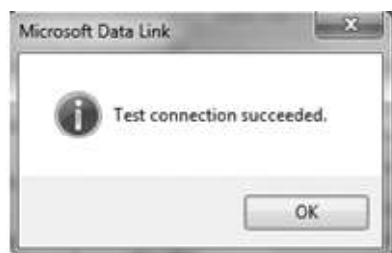


9. As your DB is made in Access thus select provider 'Microsoft Jet 4.0 OLE DB Provider' from Provider tab and click on Next button.
10. This will open the connection tab. Under connection tab select the DB which you want to connect with the Project .i.e Company.

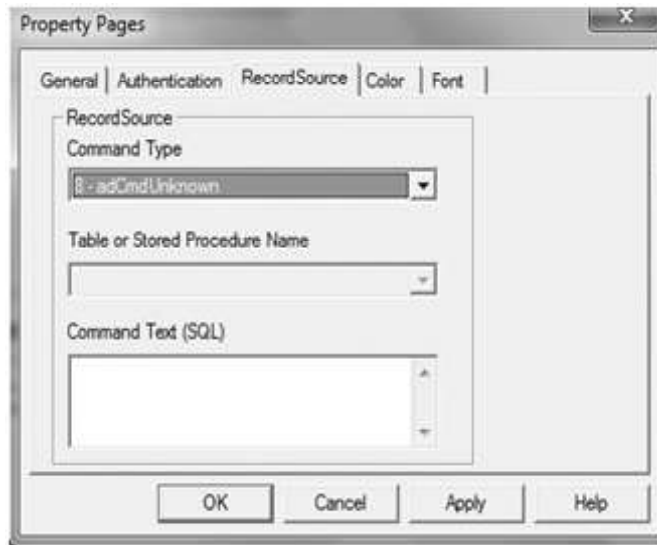


Now, click on Test Connection.

11. Wait for message “ Test Connection Succeed “® Click OK

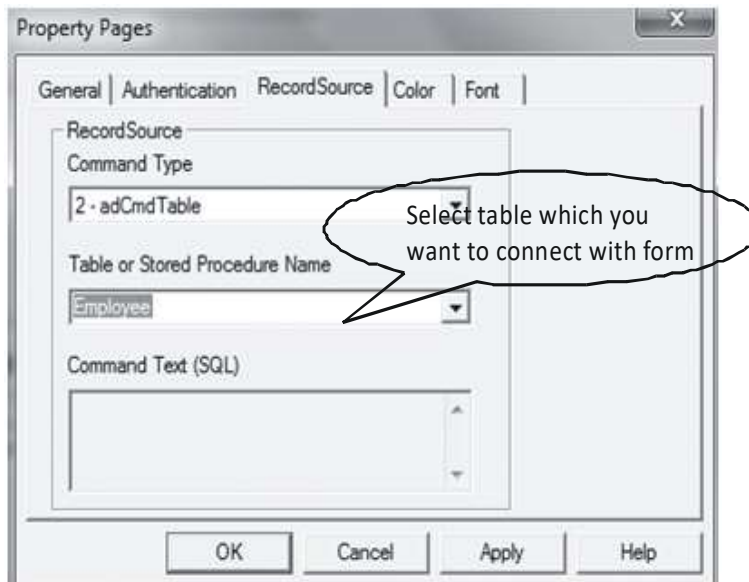


12. Select record source tab



13. Select adcmdTable for command type field

14. Select Employee table name for table or stored procedure name field

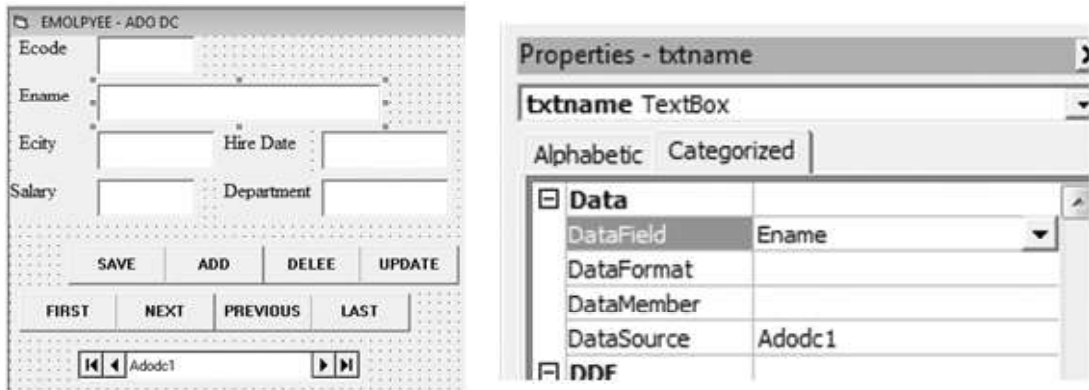




15. Click on Apply and then Ok.

**Attach form with different attributes of table**

16. Now, one by one bind every text box with their respective attribute name of MS-Access. For this select textbox, set its Data Source property to ADODC1 and select respective attribute from dropdown list of Data Field property.



17. **Navigating the Recordset.** Use the first, next, previous and last button to display data of the DB. The **first** button will take you to the very first record of the table. **Next** shows the followed data whereas **previous** displays the preceded data of the currently shown data of the table. **Last** button takes you to the last record of the table. Write the following code for navigation:

---

```
Private Sub cmdfirst_Click()  
On Error GoTo message  
Adodc1.Recordset.MoveFirst  
Exit Sub  
message:  
Adodc1.Recordset.MoveFirst  
End Sub
```

---

```
Private Sub cmdlast_Click()  
On Error GoTo message  
Adodc1.Recordset.MoveLast  
Exit Sub  
message:  
Adodc1.Recordset.MoveLast  
End Sub
```

---

```
Private Sub cmdnext_Click()  
On Error GoTo message  
    If Adodc1.Recordset.EOF Then  
        Adodc1.Recordset.MoveFirst  
    Else  
        Adodc1.Recordset.MoveNext  
        If Adodc1.Recordset.EOF Then  
            Adodc1.Recordset.MoveFirst  
        End If  
    End If  
Exit Sub  
message:  
Adodc1.Recordset.MoveFirst  
End Sub
```

---

```
Private Sub cmdnext_Click()  
On Error GoTo message  
    If Adodc1.Recordset.EOF Then  
        Adodc1.Recordset.MoveFirst  
    Else  
        Adodc1.Recordset.MoveNext  
        If Adodc1.Recordset.EOF Then  
            Adodc1.Recordset.MoveFirst  
        End If  
    End If  
Exit Sub  
message:  
Adodc1.Recordset.MoveFirst  
End Sub
```

---

```
Private Sub cmdpre_Click()  
On Error GoTo message  
If Adodc1.Recordset.BOF Then  
    Adodc1.Recordset.MoveLast  
Else  
    Adodc1.Recordset.MovePrevious  
    If Adodc1.Recordset.BOF Then  
        Adodc1.Recordset.MoveLast  
    End If  
End If  
Exit Sub  
message:  
Adodc1.Recordset.MoveLast  
End Sub
```

---

We also include error handling in each procedure to prevent our program from improper termination.

18. **To Add new record**, first you should make empty all the textboxes. For this you use Add button. And to save the entered record into the table, you then click on Save button. Both button contains the following code.

```
Private Sub cmdadd_Click()  
Adodc1.Recordset.AddNew  
End Sub
```

---

This will clear all the textboxes. Now enter the new record as

Now, execute the following command to save the record into the DB.

```
Private Sub cmdsave_Click()
    Adodc1.Recordset.Fields("ECode").Value = txtcode.Text
    Adodc1.Recordset.Fields("EName").Value = txtname.Text
    Adodc1.Recordset.Fields("Ecity").Value = txtcity.Text
    Adodc1.Recordset.Fields("Hire_Date").Value = txtdate.Text
    Adodc1.Recordset.Fields("Salary").Value = txtsal.Text
    Adodc1.Recordset.Fields("Department").Value = txtdept.Text
    Adodc1.Recordset.Update
    Adodc1.Refresh
End Sub
```

When you click on the save button , control will automatically shift to the first record of the DB.

See, the record is entered into the Table.

ECode	Ename	Ecity	Hire_Date	Salary	Department
1001	Amit	Delhi	4/10/2001	55,560.00	Production
1002	Shilpi	Dehradun	9/19/2005	22,670.00	Sales
1003	Aaditya	Mumbai	12/6/2004	34,990.00	Production
1004	Aarya	Chennai	4/5/2009	20,500.00	Marketing
1005	Swadha	Lucknow	7/7/2001	45,550.00	Personnal
1006	Dinesh	Delhi	9/14/1999	85,500.00	Personnal
1007	Tanu	Banglore	6/23/2000	40,500.00	Development
*					

19. **To Update Record.** If you want to make changes in the existing record , then use the following command

```
Private Sub cmdupdate_Clickn()
Adodc1.Recordset.Update
End Sub
```

Lets change some values of record no. 1006.

When you click on the update button, the change will save in the table.

ECode	Ename	Ecity	Hire_Date	Salary	Department
1001	Amit	Delhi	4/10/2001	55,560.00	Production
1002	Shilpi	Dehradun	9/19/2005	22,670.00	Sales
1003	Aaditya	Mumbai	12/6/2004	34,990.00	Production
1004	Aarya	Chennai	4/5/2009	20,500.00	Marketing
1005	Swadha	Lucknow	7/7/2001	45,550.00	Personnal
1006	Dinesh Srivastav	Delhi	9/14/1999	85,500.00	Personnal
1007	Tanu	Banglore	6/23/2000	40,500.00	Development
*					

20. **To Delete Record.** Display the desired record on the form and click on delete button. The delete button will carry the following command:

```
Private Sub cmdDel_Click()
    Adodc1.Recordset.Delete
End Sub
```

Delete the following record no 1004:

Now click on delete button. See the record will be deleted from the table.

ECode	Ename	Ecity	Hire_Date	Salary	Department
1001	Amit	Delhi	4/10/2001	55,560.00	Production
1002	Shilpi	Dehradun	9/19/2005	22,670.00	Sales
1003	Aaditya	Mumbai	12/6/2004	34,990.00	Production
1005	Swadha	Lucknow	7/7/2001	45,550.00	Personnal
1006	Dinesh Srivastav	Delhi	9/14/1999	85,500.00	Personnal
1007	Tanu	Banglore	6/23/2000	40,500.00	Development
*					

**Note:** If you find it difficult to connect DB made in 2007, then convert the DB in 2000 .

## 8.9 DB CONNECTIVITY BY ADODB

After using ADO DC , let us now learn to use ADO programmatically i.e ADODB . The ADO is a DB access paradigm that enables client applications to access and manipulate data in a DB server through an OLEDB provider. It provides you high speed, low requirement of disk space , low memory overheads and easy to use.

### 8.9.1 Objects Of ADO

ADO providers collections, a type of object that conveniently contains other objects of a particular type. The objects in the collection can be retrieved with a collection method .It includes four major objects:

1. **Connection:** The connection object has the Errors collection, which contains all Error objects created in response to a single failure involving the data source. That means, it allows establishing connection with data source. *Open*, *Close* and *Execute* are some common methods of connection object.

**Open** method opens a connection to a data source. It uses a connection string to connect to the DB. The connection string uses the following attributes:

- (a) File Name – Name of the file that contains connection information.
  - (b) Provider – Name of the provider.
  - (c) Data Source – Name of the server or name of the DB to which you want to connect.
  - (d) User ID – The user’s name
  - (e) Password – The user’s password.
2. **Command:** This object contains information about a command such as query string, parameter definition, which contains all parameter objects that apply to that command object. This may be either a SQL statement or an invocation of a stored procedure.
  3. **Recordset:** The Recordset object has a field collection, which contains all field objects that define the columns of that Recordset object. It is used to manipulate rows in the DB and contain the result of a query.
  4. **Field:** This object contains information about a single column of data in a Recordset.

### 8.9.2 Locktype

The *LockType* argument is an optional value that determines the type of locking that the provider should use when opening the recordset. The possible values of *lockType* are:

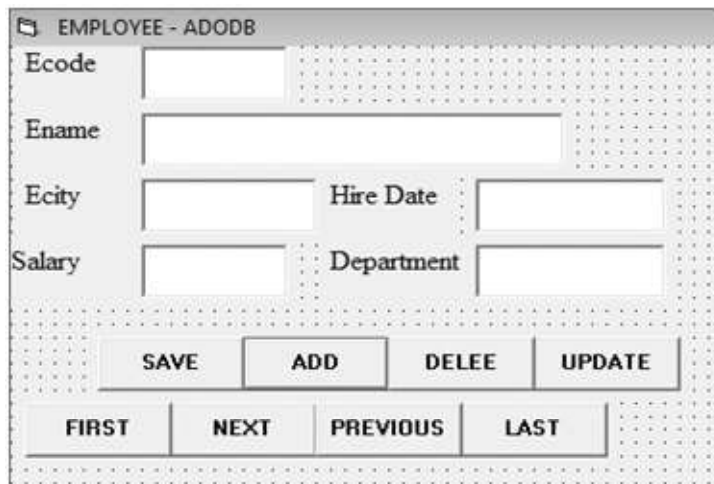
1. **adLockReadOnly:** (default) Specifies read-only locking. Records can be read, but data cannot be added, changed, or deleted. This is the locking method used with static cursors and forward-only cursors.
2. **adLockPessimistic:** Specifies pessimistic locking. The provider does what is necessary to ensure successful editing of records, usually by locking records at the data source immediately upon editing.
3. **adLockOptimistic:** Specifies optimistic locking. The provider locks records only when you call the Update method, not when you start editing.
4. **adLockBatchOptimistic:** Specifies optimistic batch locking. Records are locked in batch update mode, as opposed to immediate update mode. This option is required for client-side cursors.

### 8.9.3 Steps To Access DB Through ADODB

#### Design the form

1. Design the form same as you designed in above section (Connection through DAO).

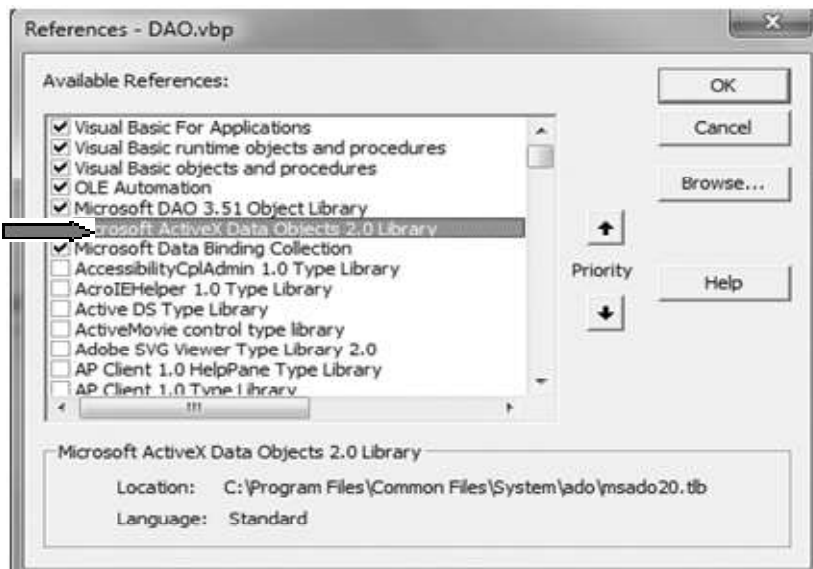




Names of all text boxes and command buttons are same as earlier.

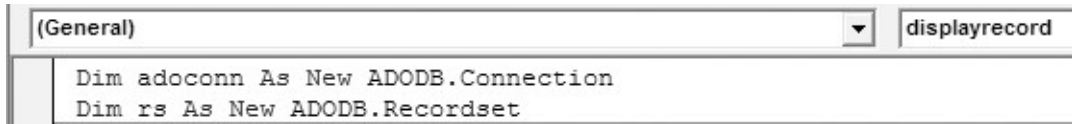
### Set reference to ADODB

2. Click on Project menu and select Reference.
3. Select Microsoft ActiveX Data Objects 2.x Library. Click on Ok.



### Create connection object

4. Declare the following object in general section.



To open Employee Table of Company DB

5. Write the following code in Load event of form. This will connect form with DB.

```

Private Sub Form_Load()
dim connectingstring As String
connectingstring = "Provider = Microsoft, Jet, OLEDB,
4.0; Data Source=C:\Users\shilp\desktop\Company1.mdb.;"
adoconn.Open connectingstring
rs. Open "Employee", adoconn, adOpenDynamic, adLockOptimistic
End Sub

```

Provider will be selected according to the DB and Data Source displays the location of DB where it is stored.

### Bind different textboxes with Table fields

6. To connect different fields of access DB with different textboxes placed on form, you create a following procedure in general section of code window ,which you call whenever it is required.

```

Public Sub displayrecord()
txtcode.Text = rs!Ecode
txtname.Text = rs!Ename
txtcity.Text = rs!ECity
txtdate.Text = rs!Hire_Date
txtdept.Text = rs!Department
txtsal.Text = rs!Salary
End Sub

```

7. Now **create the navigation buttons**. To display data on the form with the help of navigation button, write the following code on respective buttons:

---

```
Private Sub cmdfirst_Click()  
    If rs.EOF = False Then  
        rs.MoveFirst  
        Call displayrecord  
    End If  
End Sub
```

---

```
Private Sub cmdlast_Click()  
    If rs.EOF = False Then  
        rs.MoveLast  
        Call displayrecord  
    End If  
End Sub
```

---

```
Private Sub cmdnext_Click()  
    If rs.EOF = False Then  
        rs.MoveNext  
    Else  
        rs.MoveFirst  
    End If  
    Call displayrecord  
End Sub
```

---

```
Private Sub cmdpre_Click()  
    If rs.EOF = False Then  
        rs.MovePrevious  
    Else  
        rs.MoveLast  
    End If  
    Call displayrecord  
End Sub
```

---

When you click on last button, last record of the table is displayed:

EMPLOYEE - ADODB

Ecode: 1007

Ename: Tanu

Ecity: Banglore Hire Date: 6/23/2000

Salary: 40500 Department: Development

SAVF ADD DELETE UPDATF

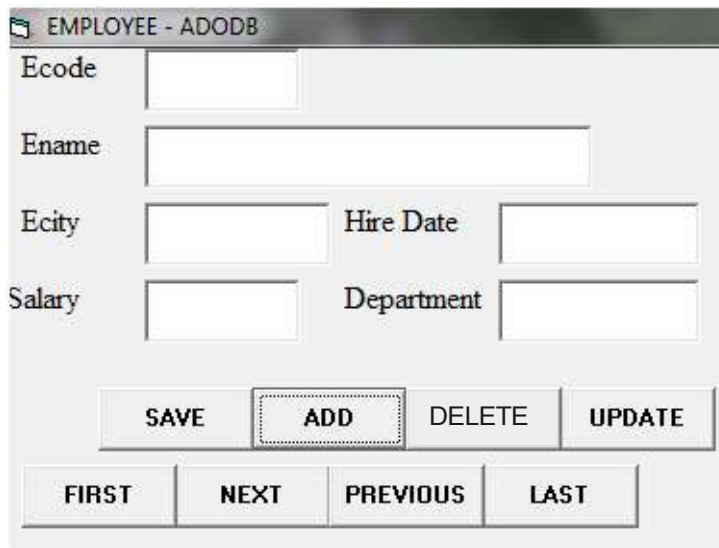
FIRST NEXT PREVIOUS LAST

8. To add new record first **clears all the text boxes**. To do this write the following code on click event of Add button.

```
Private Sub cmdadd_Click()  
Dim ctrl As Control  
    For Each ctrl In Me.Controls  
        If TypeOf ctrl Is TextBox Then  
            ctrl.Text = ""  
        End If  
    Next  
End Sub
```

---

This will clear all the text boxes:



The screenshot shows a form titled "EMPLOYEE - ADODB" with the following fields and buttons:

Ecode	<input type="text"/>		
Ename	<input type="text"/>		
Ecity	<input type="text"/>	Hire Date	<input type="text"/>
Salary	<input type="text"/>	Department	<input type="text"/>

Buttons: SAVE, ADD, DELETE, UPDATE

Navigation: FIRST, NEXT, PREVIOUS, LAST

9. Now **add new record** in the fields :



The screenshot shows the same form titled "EMPLOYEE - ADODB" with the following data entered:

Ecode	1008		
Ename	Shan		
Ecity	Allahabad	Hire Date	8/3/2003
Salary	34,000	Department	Sales

Buttons: SAVE, ADD, DELETE, UPDATE

Navigation: FIRST, NEXT, PREVIOUS, LAST

To add above data in the DB, click on save button which contains following code:

```

Private Sub cmdsave_Click()
    If (MsgBox("Are you sure you want to save this record?", vbYesNo) = vbYes) Then
        rs.AddNew
        rs!Ecode = Val(txtcode.Text)
        rs!Ename = txtname.Text
        rs!ECity = txtcity.Text
        rs!Hire_Date = txtdate.Text
        rs!Salary = txtsal.Text
        rs!Department = txtdept.Text
        rs.Update
    End If
End Sub

```

When you click on save button, the form will appear as follows:



Click on yes button to save the record.

ECode	Ename	Ecity	Hire_Date	Salary	Department
1001	Amit	Delhi	4/10/2001	55,560.00	Production
1002	Shilpi	Dehradun	9/19/2005	22,230.00	Sales
1003	Aaditya	Mumbai	12/6/2004	34,990.00	Production
1005	Swadha	Lucknow	7/7/2001	45,550.00	Personnal
1006	Dinesh Srivastav	Delhi	9/14/1999	85,500.00	Personnal
1007	Tanu	Banglore	6/23/2000	40,500.00	Development
1008	Shan	Allahabad	8/3/2003	34,000.00	Sales
*					

10. To **update existing record** write the following code:

```
Private Sub cmdupdate_Click()
    If (MsgBox("Are you sure you want to update this record?", vbYesNo) = vbYes) Then
        rs.Update
    End If
End Sub
```

The window will appear as



When you click on yes button the record will updated in the table.

ECode	Ename	Ecity	Hire_Date	Salary	Department
1001	Amit	Delhi	4/10/2001	55,560.00	Production
1002	Shilpi	Dehradun	9/19/2005	42,500.00	Sales
1003	Aaditya	Mumbai	12/6/2004	34,990.00	Production
1005	Swadha	Lucknow	7/7/2001	45,550.00	Personnal
1006	Dinesh Srivastav	Delhi	9/14/1999	85,500.00	Personnal
1007	Tanu	Banglore	6/23/2000	40,500.00	Development
1008	Shan	Allahabad	8/3/2003	34,000.00	Sales

11. To **delete** any record from the table write the following code:

```

Private Sub cmdDel_Click()
    If (MsgBox("Are you sure you want to delete this record?", vbYesNo) = vbYes) Then
        rs.Delete
        rs.Update
    End If
End Sub

```

To delete record display the desired record on the form and click on delete button:



When you click on yes button, the record will be deleted from the table.

ECode	Ename	Ecity	Hire_Date	Salary	Department
1001	Amit	Delhi	4/10/2001	55,560.00	Production
1002	Shilpi	Dehradun	9/19/2005	42,500.00	Sales
1003	Aaditya	Mumbai	12/6/2004	34,990.00	Production
1005	Swadha	Lucknow	7/7/2001	45,550.00	Personnal
1006	Dinesh Srivastav	Delhi	9/14/1999	85,500.00	Personnal
1007	Tanu	Banglore	6/23/2000	40,500.00	Development
*					

These are the different ways by which you can connect your database with your form i.e backend with frontend.



## **8.10 VISUAL DATABASE TOOLS**

Accessing DB becomes very important and crucial when you develop real life projects. In real life projects, you need to perform many complex activities, which may involve multiple tables or view. For this you need to use ADO programmatically because with ADO DC you get bound to the limits. For this VB offers you Visual DB Tools which uses – Connection, Recordset and Command Objects of ADO and create elaborate form and display reports. These tools are –

1. Data Environment
2. Data Report and Crystal Report

### **8.10.1 Data Environment Designer**

The Data Environment designer provides an interactive, environment for creating programmatic data access. At design time, you set property values for Connection and Command objects, write code to respond to ActiveX Data Object (ADO) events, execute commands, and create aggregates and hierarchies. You can also drag Data Environment objects onto forms or reports to create data-bound controls.

Thus, The data environment designer is a design time representation of ADO objects, which are created at runtime. It allows us to add Data environment object ,using which data is accesses from database.

With the Data Environment designer, we can accomplish the following tasks:

- I. Add a Data Environment designer to a Visual Basic project.
- II. Create Connection objects.
- III. Create Command objects based on stored procedures, tables, views, synonyms, and SQL statements.
- IV. Create hierarchies of commands based on a grouping of Command objects, or by relating one or more Command objects together.
- V. Write and run code for Connection and Recordset objects.
- VI. Drag fields within a Command object from the Data Environment designer onto a Visual Basic form or the Data Report designer.

### **8.10.2 Data Report**

The data that we view on the form is not always in the presentable form. To store hardcopy of the data presented in the formatted manner, we need to create reports. VB6.0 introduced the new data report designer to create reports visually from within VB.

Data Report is a visual database tool to create and design reports based on database data.

Each report part appears in a separate band of the report.

Following are the section of the data reports:

1. **Report Header:** It contains the text that appears at the very beginning of a report, such as report title, author or database name.
2. **Page Header:** It contains information that goes on the top of every page, such as report title.
3. **Group Header/Footer:** It contains information that signifies the beginning of a new group. Each group header is matched with group footer. The header and footer pair is associated with a single command object in the data environment designer.
4. **Details:** It contains the innermost repeating part of the reports. The detail band consists of the layout applicable to one record, which is repeated for every record.
5. **Page Footer:** It contains the information that goes on the bottom of every page, such as page footer.
6. **Report Footer:** It contains text that appears at the very end of the report, such as summary information, address or contact details.

### 8.10.3 Crystal Report

Crystal Reports allows users to graphically design data connections and report layout. In the Database Expert, users can select and link tables from a wide variety of data sources, including Microsoft Excel spreadsheets, Oracle databases, Business Objects Enterprise business views, and local file system information. Fields from these tables can be placed on the report design surface, and can also be used in custom formulas, using either BASIC or Crystal's own syntax, which is then placed on the design surface. Formulas can be evaluated at several phases during report generation as specified by the developer.

Both fields and formulae have a wide array of formatting options available, which can be applied absolutely or conditionally. The data can be grouped into bands, each of which can be split further and conditionally suppressed as needed. Crystal Reports also supports sub reports, graphing. With Crystal Reports, you can create complex and professional reports in a GUI-based program. Popular reporting and analysis software for Windows from SAP that is used to retrieve data from more than 30 types of databases. Queries and reports can be made via a Web browser, and the functionality can also be added to proprietary programs written in languages such as C, C++, J++, Delphi and Visual Basic.

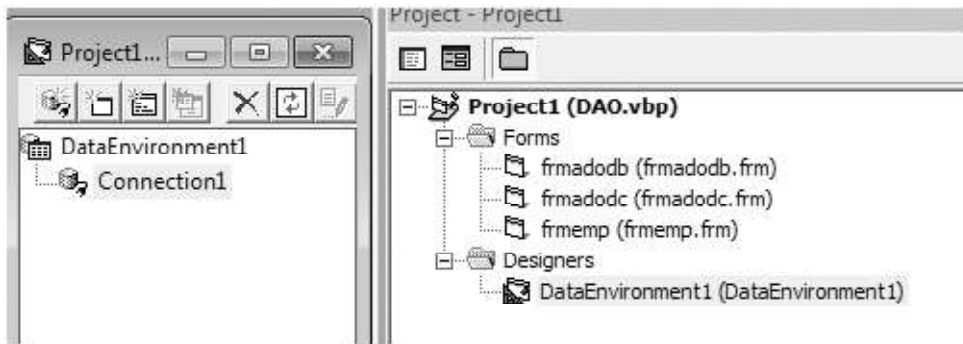
## 8.11 CREATING DATA REPORT

Let us create a report from *Company DB* which includes fields from *Employee* Table. You name this report *Emp\_Dept* which contains attributes *Ecode*, *Ename*, *Department* and *Hire\_Date*.

To create data report first we need to include Data Environment. So, first we create data environment and then data report.

### 8.11.1 Steps To Create Data Environment For Emp\_Dept

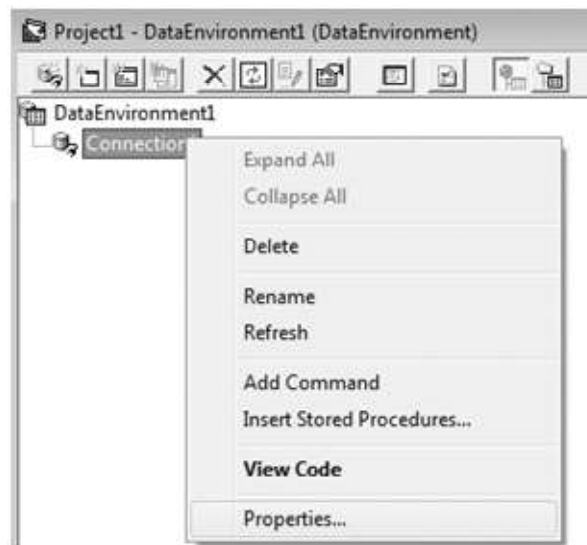
1. Click on Project in which you create the form ® Add data environment



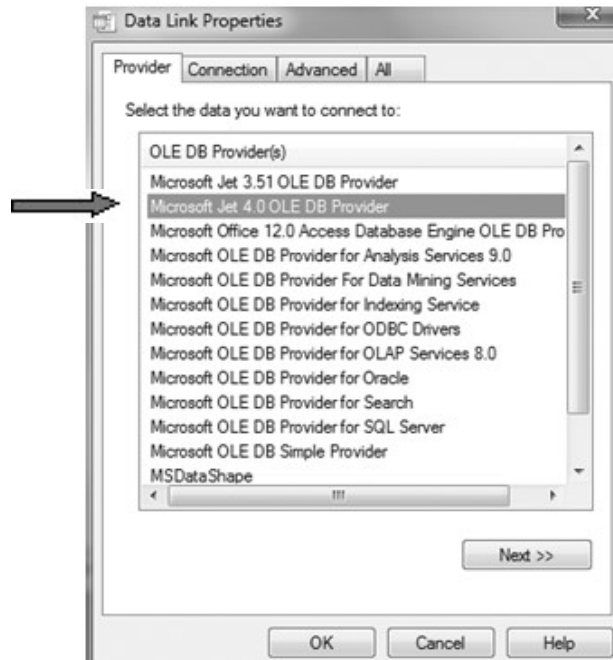
### Adding connections

To access data from data environment, create a connection object. Thus every data environment has atleast one connection object.

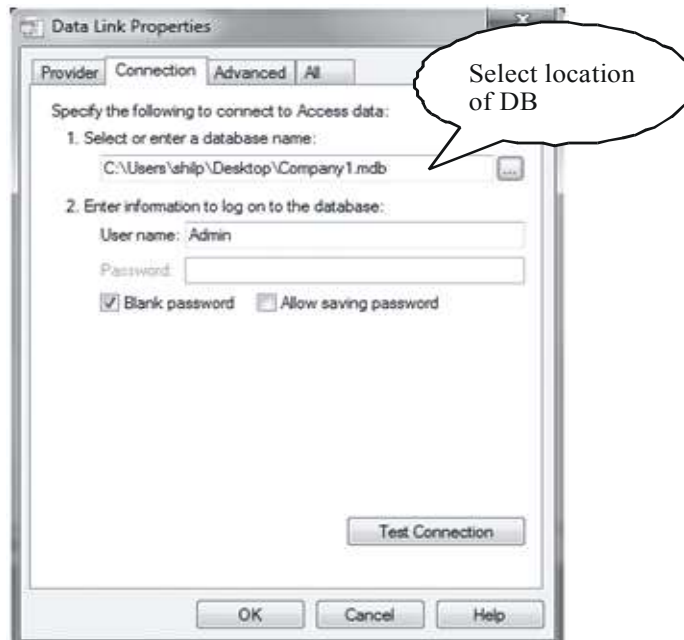
2. Click on Add connection. By default its name is connection1.
3. Right click on connection1. Select properties.



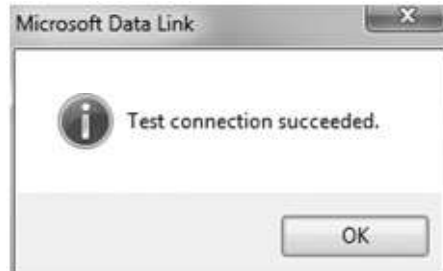
4. From Data Link Properties dialog box, First select provider from provider tab and click on next.



5. From connection tab, enter DB name



6. Click on Test Connection and wait for message “Test connection succeed”.



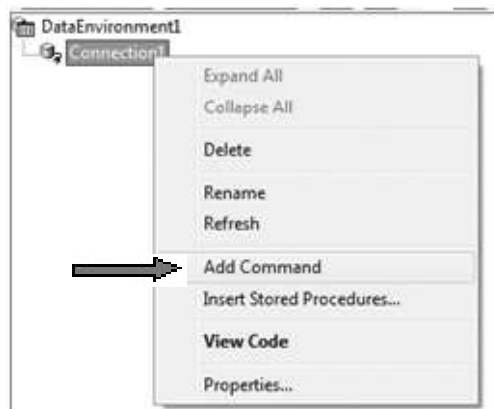
7. Click on Ok then again on Ok.

### Adding commands

After creating connection object, we have to create the command object. The command object fetches data from the database by using connection objects. The command can be based on a table, a view, a stored procedure or a SQL statement.

Thus, Command Object represents an action performed on a database.

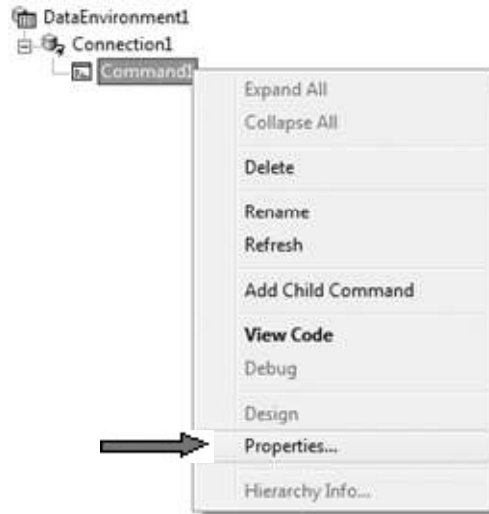
8. Right click on data environment connection1, click on ADD Command.



This will appear as



9. Right click on command object and select properties

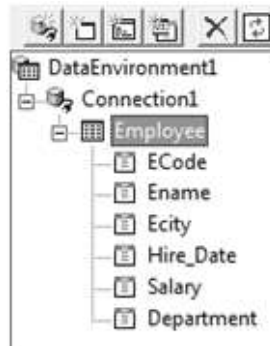


10. In general tab set values for command name (command1) , connection (connection1) Database Object ( Table/ Query), Object Name( Table /Query name)



11. Click on Apply and then OK.

This will add all the fields of Employee table in Command object. Rename command1 as Employee.



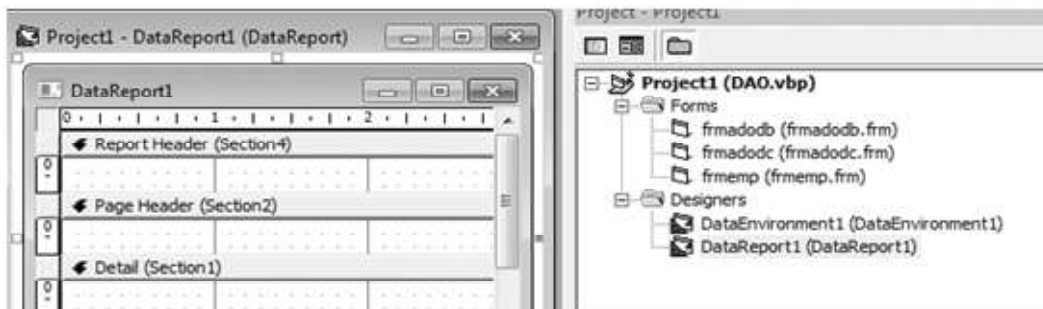
There are two type of command object –

- (a) **Recordset returning command object:** Command object returns rows of data and result can be accessed using a recordset object available from data environment.
- (b) **Non – Recordset returning command Type:** Command object does not return rows of data .

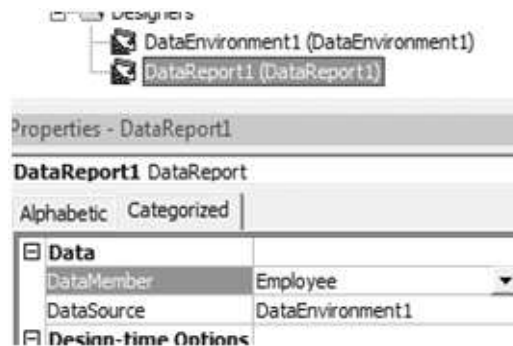
### 8.11.2 Steps To Create Data Report Emp\_Dept

After creating data environment you add data report to your project.

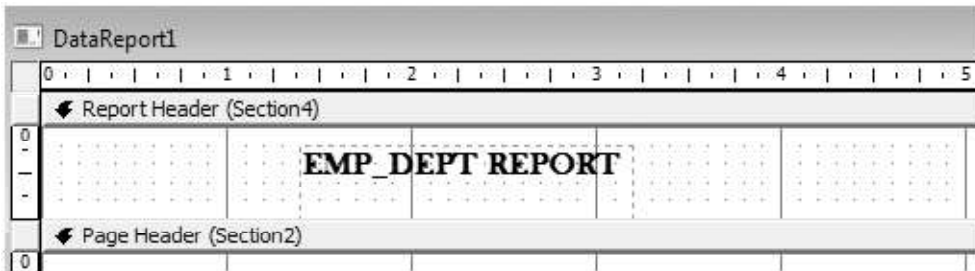
12. Click on Project menu ® Add Data report



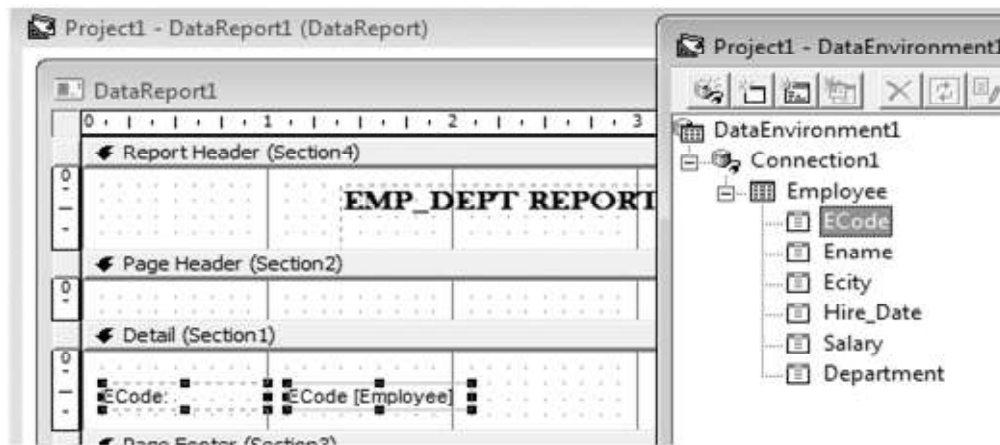
13. Set Name to rptemployee, Data Source Property to Data Environment1 and Data Member Property to Command object of data environment i.e Employee



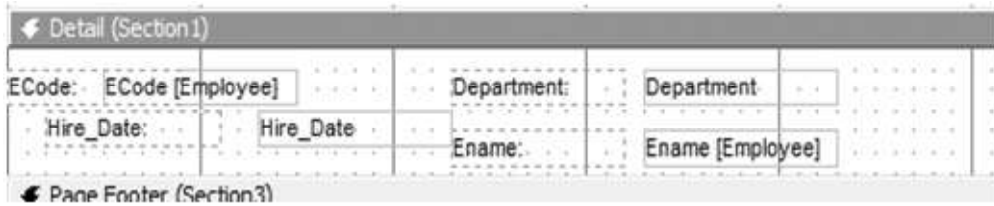
14. Write Report Heading by using tools of data report.



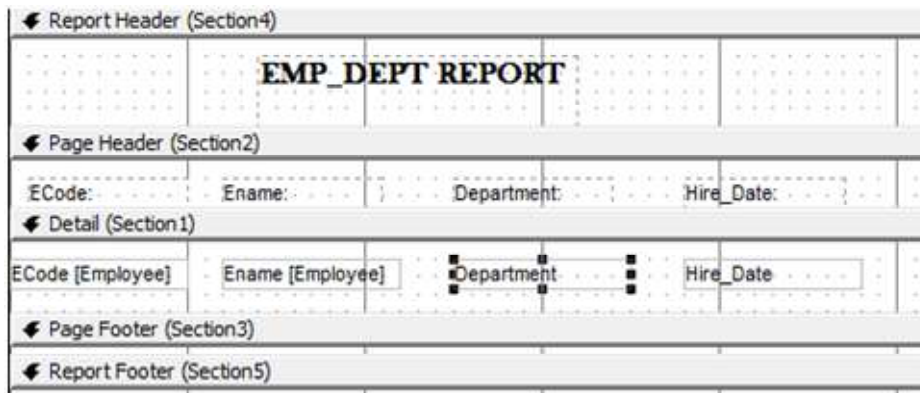
15. Drag all the desired fields of Employee command object from data environment window to Detail Section of Data Report.



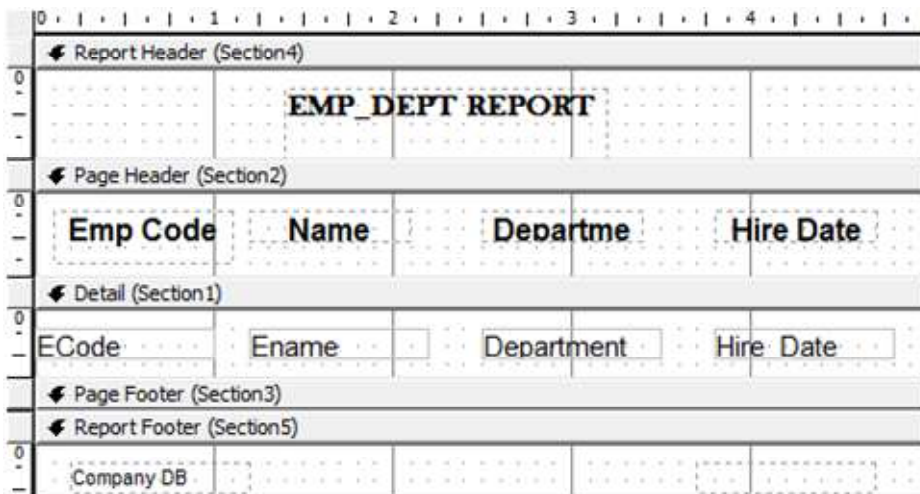




16. Place the Labels in Page Header section and text boxes in Detail Section.



17. Change the page header heading if required and arrange the view of report.



18. Save the report as rptemployee.

19. Create a command button in respective form i.e Employee - ADODB.

The screenshot shows a form titled "EMPLOYEE - ADODB". It has a grid-like background. The form contains the following fields and buttons:

- Text boxes: Ecode, Ename, Ecity, Hire Date, Salary, Department.
- Buttons: SAVE, ADD, DELETE, UPDATE (top row); FIRST, NEXT, PREVIOUS, LAST (middle row); EMP\_DEPT REPOT (bottom center).

20. To execute report , write code

---

```
Private Sub cmdreport_Click()  
    rptemployee.Show  
End Sub
```

---

21. To view report click on Emp\_Dept button. The report will appear as:

<b>EMP_DEPT REPORT</b>			
<b>Emp Code</b>	<b>Name</b>	<b>Department</b>	<b>Hire Date</b>
1003	Aaditya	Production	12/6/2004
1005	Swadha	Personnal	7/7/2001
1006	Dinesh	Personnal	9/14/1999
1007	Tanu	Development	6/23/2000
1001	Amit	Production	4/10/2001
1002	Shilpi	Sales	9/19/2005

Company DB 5/23/2011

Now, you are able to connect a DB with a VB form by different means of connectivity and also you are cable to generate reports. So, similarly as above connect Department DB with your form and create its report.

## 8.12 TRANSACTIONS AND CONCURRENCY CONTROL

A Database is a software system that defines a collection of predefined operations. Mainly it includes following operations:

- Efficient management of large amount of persistent data in a persistent storage (database)
- Transaction Management which includes Concurrency Control, Atomicity and backup recovery procedure.
- A DataModel which gives a separate level of abstraction

### 8.12.1 Transaction

A transaction is an abstract unit of concurrent computation that execute automatically. The effect of transaction does not interfere with other transactions that access the same data. Also a transaction happens with all of its effects or it doesn't happen none of its effects In the transaction control we generally define code in between a block where we perform mission critical operation. If all operations get completed successfully then that part is committed in the database otherwise whatever modification you might have done during the process is roll backed from the database so that it never affect other user's operations.

### 8.12.2 Concurrency Control

While doing certain modification in the database some time you need to lock the data so that no one can else perform modification in that data. There are two commonly known approaches for locking database they are optimistic locking and pessimistic locking.

Both these approaches are used to maintain concurrency in the database. Pessimistic concurrency locking is done at rows of the data source to prevent users from modifying data in a way that affects other users. In a pessimistic model, when a user performs an action that causes a lock to be applied, no one else can perform action until unless owner releases that lock. But , in optimistic concurrency model user does not lock row while reading it. User only locks the row while updating changes to the database.

### LET US REVISE

- ✓ There are two types of DB – Local DB and Remote DB.
- ✓ ADO is an application program interface that lets a programmer to get access relational as well as non relational DB.
- ✓ A Recordset is a logical set of records.
- ✓ VB supports three different types of data access mechanisms – DAO, RDO and ADO.
- ✓ ODBC refers to the standard protocol that permits application to connect to a variety of external DB servers or files.
- ✓ A bound control provides access to a specific column or columns in a data source through data controls.
- ✓ Data provider is a control that provides data for use by connecting to a source of data.
- ✓ ADO uses a underlying system service called OLE DB.
- ✓ To use ADO data control, Microsoft ActiveX Data Control 6.0 component is added through Component command under Project menu.
- ✓ To bind text boxes , its data source and data field properties are set.
- ✓ To navigate a Recordset following methods are used – Movefirst, Movenext, Moveprevious, Movelast.

- ✓ To modify DB following methods are used – Add, New, Update and Delete.
- ✓ To test end of file and beginning of file, EOF and BOF properties are used.
- ✓ To use ADO programmatically, Microsoft ActiveX Library 2.x is set through References command under Project menu.
- ✓ Four major objects of ADO DB are : connection, Recordset, command and field.
- ✓ Locktype determines which type of lock is used by the provider at the time of opening a recordset.
- ✓ OLE DB providers code for oracle DB is MSDAORA and for ODBC its MSDASQL.
- ✓ OLE DB provider for Ms-access DB is Microsoft jet OLE DB provider.
- ✓ A Recordset is opened through open method.
- ✓ Visual DB tools consist of data environment and data report.
- ✓ The data environment designer is a design time representation of ADO objects.
- ✓ Data environment is added through project menu.
- ✓ Data environment lets us create various ADO objects such as connection, command field object without programming.
- ✓ Data report is a visual DB tool to create and design reports based on DB data.
- ✓ All designer files are stored with .dsr extension.
- ✓ <reportname>.show method is used to display reports.
- ✓ A transaction is an abstract unit of concurrent computation that execute automatically.

**Assignment:**

Create the grouping report using the some record as on the page 207 (emp Dept. Report). Also count the number of employees in each department.

## Chapter-9

# Help Writing and Some Other Features

## 9.1 INTRODUCTION

### 9.2 HELP FILE

Online help is topic-oriented, procedural or reference information delivered through computer software. It is a form of user assistance. Most online help is designed to give assistance in the use of a software application or operating system, but can also be used to present information on a broad range of subjects. When online help is linked to the state of the application, it is called context-sensitive help.

Microsoft products such as Microsoft Visual Studio to locally installed help as Offline help or Local help, while help installed on their web server is referred to as Online help. However the general term Online help has always referred to the electronic documentation associated with an application.

#### 9.2.1 Help Modes

The help system can run in three modes: workbench (normal), information center, and standalone. Workbench mode is used for serving help integrated with the product, usually via a Help menu. This mode also offers context help and the help view, which are not available in the two other modes. **Standalone** mode has the same goal as workbench mode, but is for products that are not eclipse-based .. **Information center** mode is used to serve help content to the masses over the Web.

#### 9.2.2 Requirements

The following are recommended hardware, software, network infrastructure, skills and knowledge and service packs-

- HTML Help 1.3
- Visual Basic development.

Every statement, function, property, method and event is fully documented. However, that

doesn't really help us if we don't know what the keyword is that we need to find to accomplish our objective. The VB help files are well indexed and offer the capability of full-text searching. There are a number of different files needed to create a Windows Help file. A Help file is made up following—

- (a) **Rich Text Format (RTF) files:** Microsoft Windows on-line help files are generated by using one or more Rich Text Format (.RTF) source files. Many word processors can save documents in this format including MS Word. Each help file can have multiple topics and multiple links between topics.
- (b) **Hypertext links and pointers to any graphics file:** To create a link which will replace the currently displayed topic with a new one, first write the text or insert a graphic which will form the link, now double underline it, and then immediately after it write the topic ID of where the link will go and place it in hidden text.
- (c) **A Help Project (HPJ) file:** a Help Project (HPJ) file which holds settings and file names of all the source RTF files. The Help Project and provide a suitable file name with the extension '.HPJ'. All that has been created is a new help file project. This is quite similar to a Visual Basic project file, it is used to determine which files and settings are used in the compilation process.
- (d) **Help Compiler Workshop:** These three different file types are then processed by the Help Compiler Workshop to produce a help file (.HLP) and a contents file (.CNT). After the help file has been written and compiled it is ready for use by an application.

### 9.2.3 HTML Help File

The principal entry point for HTMLHelp operations in Microsoft Visual Basic is the HTMLHelpfunction. This application programming interface (API) function is declared as :

```
Private Declare Function HTMLHelp Lib "HHCtrl.ocx" Alias "HTMLHelpA" _  
    (ByVal hWndCaller As Long, _  
    ByVal pszFile As String, _  
    ByVal uCommand As Long, _  
    dwData As Any) As Long
```

The first parameter represents a parent window for your application. The second parameter is the name of the compiled (.chm) file that contains the help data. The third parameter is a value that represents an HTMLHelp command. The fourth parameter is additional data, the value and format of which depends on the HTMLHelp command.

### 9.2.4 Steps To Create Html Help File

1. Click Start, point to Programs, and then click HTML Help Workshop two times to start HTML Help Workshop.

2. On the File menu, click New.
3. In the New dialog box, click Project, and then click OK. The New Project wizard starts.
4. Follow these steps in the New Project Wizard:
  - In the first dialog box, press Next.
  - In the Destination dialog box, enter the folder and file name for the help project, and then click Next.
  - In the Existing Files dialog box, press Next.
  - Click Finish to create a blank project.
5. On the File menu, click New.
6. In the New dialog box, click HTML File, type Default for the title, click OK, typeDefault between the <BODY> and </BODY> tags, and then save this file as Default.htm.
7. On the File menu, click New.
8. In the New dialog box, click HTML File, type Sample Topic for the title, click OK, typeSample Topic between the <BODY> and </BODY> tags, and then save this file as Sample.htm.
9. On the left toolbar in the HTML Help Workshop window, click Add/Remove topic files.
10. Click Add, browse for both the Default.htm file and the Sample.htm file in the file selection dialog box, click Open, and then click OK. The two files are now listed in the Files section at the left of the HTML Help Workshop window.
11. Open Notepad, and then type the following context IDs:

```
#define DEFAULT 100
#define SAMPLE 101
```

Save the file as Map.h in the same folder as the other HTMLHelp project files that you have created.
12. On the left toolbar in the HTML Help Workshop window HTML Help Workshop, click**HtmlHelp API information**, click Header File, type the file name Map.h, and then click OK two times.
13. On the File menu, click Save All Files.
14. On the File menu, click Compile.
15. In the **Create a compiled file** dialog box, click Compile. This creates a compiled HTML help file named HHDemo.chm.  
Uses the Help File
16. Start a new Visual Basic 6.0 Standard EXE Project.
17. Add two Command buttons to Form1. The buttons have the default names Command1 and Command2.



18. Add the following code to the General Declarations section of Form1:

```
Option Explicit
Private Declare Function HtmlHelp Lib "HHCtrl.ocx" Alias "HtmlHelpA" _
    (ByVal hWndCaller As Long, _
    ByVal pszFile As String, _
    ByVal uCommand As Long, _
    dwData As Any) As Long
Const HH_DISPLAY_TOPIC As Long = 0
Const HH_HELP_CONTEXT As Long = &HF
Private Sub Form_Load()
    ChDir App.Path
End Sub
Private Sub Command1_Click()
    HtmlHelp hWnd, "HHDemo.chm", HH_DISPLAY_TOPIC, ByVal "Sample.htm"
End Sub

Private Sub Command2_Click()
    HtmlHelp hWnd, "HHDemo.chm", HH_HELP_CONTEXT, ByVal 100&
End Sub
```

Save the files for this project in the same folder as the sample HTMLHelp file.

### 9.3 CONTEXT-SENSITIVE HELP

Context-sensitive help is a kind of online help that is obtained from a specific point in the state of the software, providing help for the situation that is associated with that state. Context-sensitive help, as opposed to general online help or online manuals, doesn't need to be accessible for reading as a whole. Each topic is supposed to describe extensively one state, situation, or feature of the software.

Context-sensitive help can be implemented using tooltips, which either provide a terse description of a GUI widget or display a complete topic from the help file. Other commonly used ways to access context-sensitive help start by clicking a button. One way uses a per widget button that displays the help immediately. Another way changes the mouse pointer shape to a question mark, and then, after the user clicks a widget, the help appears. Context-sensitive help is mostly used in GUI environments

## Requirements

The following are recommended hardware, software, network infrastructure, and service packs that needs to provide online help pages and context-sensitive help for the controls in a Microsoft Windows application:

- Microsoft Visual Studio 2005 or Microsoft Visual Studio .NET
- Internet Connectivity

### 9.3.1 About The Helpprovider Class

The HelpProvider class provides context-sensitive help or online help pages for controls. Each instance of HelpProvider maintains a collection of references to the controls that are associated with the instance. To provide online help pages for the control, set the HelpNamespace property of HelpProvider, and then associate a Help file with the HelpProvider object.

You can specify the type of help that the application provides for the control by calling the SetHelpNavigator method, and by providing a HelpNavigator value for the specified control. Help and HelpProvider use the HelpNavigator enumeration to provide access to specified elements of the Help file. The members of HelpNavigator are the following:

1. AssociateIndex
2. Find
3. Index
4. KeywordIndex
5. TableOfContents
6. Topic

### About the Help class

The Help class provides help to an application by calling the static ShowHelp method and the static ShowHelpIndex method. The ShowHelp method displays the contents of a Help file. The ShowHelpIndex method displays the index of the specified Help file. You can use the Help object to display Help files such as Compiled Help Module (.chm) files or HTML files that are in the HTML Help format.

### 9.3.2 Steps To Create Context-Sensitive Help

1. Start Microsoft Visual Studio 2005 or Microsoft Visual Studio .NET.
2. On the **File** menu, point to **New**, and then click **Project**.
3. Under **Project Types**, click **Visual Basic Projects**. Under **Templates**, click **Windows Application**, and then click **OK**. By default, a form that is named Form1 is created.  
Add a HelpProvider component to the Form1 form.
4. Right-click **HelpProvider1**, and then click **Properties**.

5. In the **Properties** dialog box, set the **HelpNamespace** property to **<http://msdn.microsoft.com/library/en-us/vbcon/html/vbconbuttoncontroloverview.asp>**.
6. Add a Button control to the Form1 form. **Button1** is created.
7. In the **Properties** dialog box of **Button1**, set **ShowHelp on HelpProvider1** to **True**.
8. Set the **Text** property of **Button1** to **Submit**.  
Use the HelpProvider class to provide context-sensitive help for controls
9. When you click the **Help** button on the title bar of the Form1 form, and then click the control, the Help string appears.
10. Add two TextBox controls to the Form1 form.
11. Add the following sample code to the Form1\_Load event handler.  
‘Set the Help string for the TextBox control on the form.  
HelpProvider1.SetHelpString(Me.TextBox1, “Enter your UserName”)  
HelpProvider1.SetHelpString(Me.TextBox2, “Enter your Password in this TextBox”)  
‘Set the Help string for the Button control on the form.  
HelpProvider1.SetHelpString(Me.Button1, “Click submit after you type your UserName and Password”)
12. Right-click **Form1**, and then click **Properties**.
13. In the **Properties** dialog box, set **HelpButton** to **True**.
14. Set **MaximizeBox** to **False**.
15. Set **MinimizeBox** to **False**.  
Use the Help class to display an online Help file for a control
16. Add a ListBox control to the Form1 form.
17. Right-click **ListBox1**, and then click **Properties**.
18. In the **Properties** dialog box, click the **Items** property, and then click the ellipsis button (...).
19. In the **String Collection Editor** dialog box, type the following strings:  
Domain1  
Domain2  
Domain3  
Click **OK**.
20. Add the following sample code to the ListBox1\_KeyDown event handler.  
If e.KeyCode = Keys.F1 Then  
‘Display the Help file for the ListBox control when you press F1.

```
Help.ShowHelp(ListBox1, "http://msdn.microsoft.com/library/default.asp?url=/library/en-us  
vbcon/html/vboriListBoxControlProgramming.asp")
```

*End If*

## 9.4 COMPONENT OBJECT MODEL (COM)

Component Object Model (COM) is a binary-interface standard for software componentry introduced by Microsoft in 1993. It is used to enable interprocess communication and dynamic object creation in a large range of programming languages.

The essence of COM is a language-neutral way of implementing objects that can be used in environments different from the one in which they were created, even across machine boundaries. For well-authored components, COM allows reuse of objects with no knowledge of their internal implementation, as it forces component implementers to provide well-defined interfaces that are separate from the implementation. The different allocation semantics of languages are accommodated by making objects responsible for their own creation and destruction through reference.

COM is object-oriented programming model that defines how objects interact within a single application or between applications. Client software accesses an object through a pointer to an interface on objects. Both OLE and ActiveX are based on COM. Also COM provides the interfaces between objects, and Distributed COM (DCOM) allows them to run remotely. COM objects can be small or large. They can be written in several programming languages, and they can perform any kind of processing. A program can call the object whenever it needs its services.

## 9.5 DISTRIBUTED COMPONENT OBJECT MODEL (DCOM)

Distributed Component Object Model (DCOM) is a proprietary Microsoft technology for communication among software components distributed across networked computers. DCOM was first made available in 1995 with the initial release of Windows NT 4.

Additions to the Component Object Model (COM) that facilitate the transparent distribution of objects over networks and over the Internet. DCOM is part of the specification managed by The Open Group for deployment across heterogeneous platforms.

(Distributed Component Object Model) is an extension of the Microsoft Component Object Model (COM) that allows COM components to communicate across network boundaries. Traditional COM components can only perform intercrosses communication across process boundaries on the same machine. DCOM uses the Remote Procedure Call (RPC) mechanism to transparently send and receive information between COM components (ie, clients and servers) on the same network. DCOM had to solve the problems of

- **Marshalling:** serializing and deserializing the arguments and return values of method calls “over the wire”.

- **Distributed garbage collection:** ensuring that references held by clients of interfaces are released when, for example, the client process crashed, or the network connection was lost.

## 9.6 WINDOWS APPLICATION PROGRAMMING INTERFACE (API)

The Windows Application Programming Interface, or API, is a complex set of functions for interacting with windows. The API allows Visual Basic, C++, Java and Delphi programmers to gain access to many of the functions that windows uses internally, such as printing, thread-based resources, and the registry. API calls can be used to perform almost any Windows task, from shutting down a computer to installing a printer.

### 9.6.1 What Is The Windows API?

The Windows API is a name that collectively refers to the procedures and functions that comprise the Windows Operating System. The procedures and functions are shipped with Windows in libraries, called Dynamic Link Libraries. The main API functions reside in three system DLL files, which are located on every computer that is using the Windows operating system.

- User32.dll - Handles the user interface
- Kernel32.dll - Working with files and your computers memory
- Gdi32.dll - Graphical commands

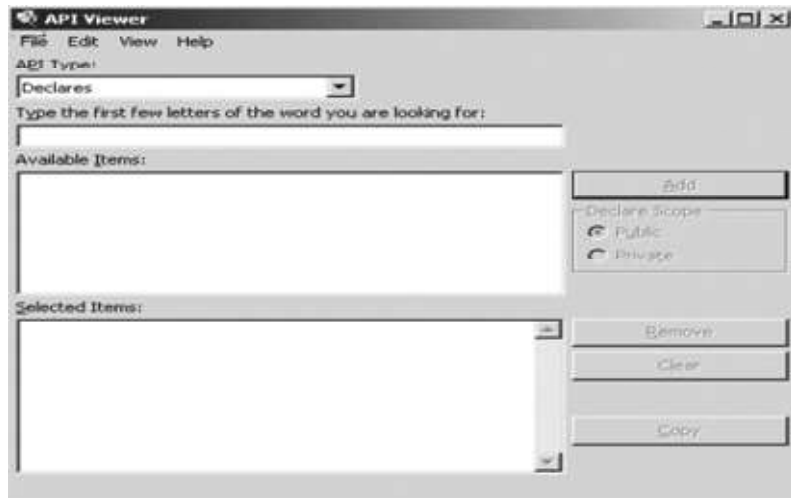
Microsoft bundled all of the API functions into system DLL's so that they were easily accessible for developers from a number of programming languages. Other developers how to create an API declaration for the functions contained within that DLL.

### 9.6.2 Exploring The API Function List

Make a program to show path of windows.

To start the API viewer,

- Click Start -> Programs -> Microsoft Visual Studio 6 -> Microsoft Visual Studio 6 Tools -> API Text Viewer.



- Click on File, then Load Text File. Find the file named Win32api.txt and load it up.  
If a box appears asking you if you want to change this file from text to database, accept it.
- Set API type to 'Declares'. The Available Items Box displays all of the API Type that we have selected.  
Select the function 'GetWindowsDirectory'. click the 'Add' button.

Its source code should appear in the Selected Items Box. Copy the source onto the Windows clipboard.

Keep open the API viewer.

### 9.6.3 Calling API

- Create a new project in Visual Basic 6. This project must contain at least a single form and one module.

Now access the source code that will call up the API function. With the source code copied to the Windows clipboard, return to Visual Basic and paste it at the top as part of the source code for the Module.

- Place a command Button on your form and set the caption to 'Show Windows Directory'. Write the following code to the Click event of this button -

```
Dim TheResult
```

```
Dim TheWindowsDirectory As String
```

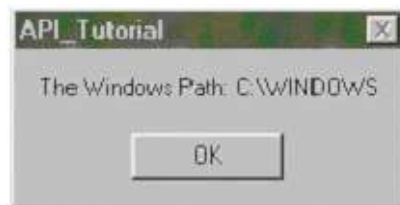
```
TheWindowsDirectory = Space(144)
```

*'Fill 144 spaces in TheWindowsDirectory string*

```
TheResult = GetWindowsDirectory(TheWindowsDirectory, 144)           'Get path and
                                                                    place it in TheResult string

If TheResult = 0 Then
    MsgBox "Cannot get the Windows Directory"
Else                                                                    'Prepare the String for preview,
                                                                    and then display in a Message Box
    TheWindowsDirectory = Trim(TheWindowsDirectory)
    MsgBox "The Windows Path: " & TheWindowsDirectory
End If
End Sub
```

- Run the program and click on the command button. Following message box is displayed:



## 9.7 MESSAGING APPLICATION PROGRAMMING INTERFACE (MAPI)

MAPI was originally designed by Microsoft. MAPI was the main e-mail data access method used by the Exchange Data Objects (EDO) and Collaboration Data Objects (CDO) interfaces.

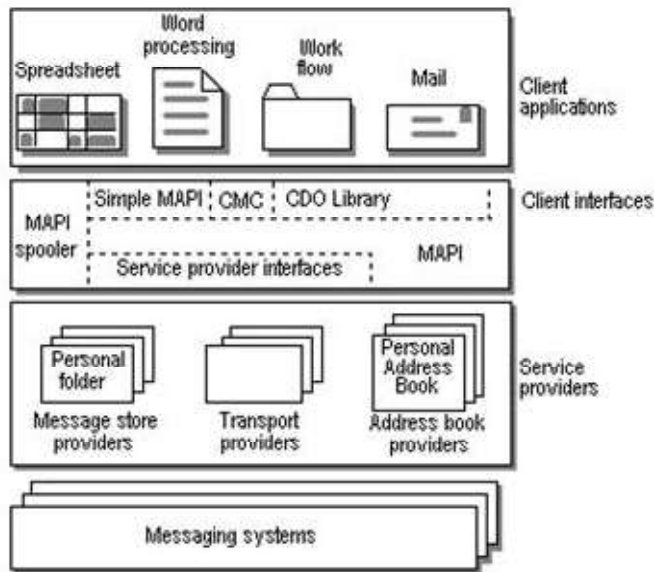
The MAPI architecture can be used for e-mail, scheduling, personal information managers, bulletin boards, and online services that run on mainframes, personal computers, and hand-held computing devices. The comprehensive architectural design allows MAPI to serve as the basis for a common information exchange.

Under MAPI architecture, to use the messaging services, a client must first establish a session. A session is a specific connection between the client and the MAPI interface based on information provided in a profile. After establishing a MAPI session, the client can use the following three primary services:

- **Address book:** A persistent database that contains valid addressing information.
- **Transport:** supports communication between different devices and different underlying messaging systems.
- **A message store:** stores messages in a hierarchical structure that consists of one or more folders.

A message in MAPI represents a communication that is sent from the sender to one or more

recipients or that gets posted in a public folder. It provides a consistent interface for multiple application programs to interact with multiple messaging systems across a variety of hardware platforms. A message can include one or more attachments.



MAPI is made up of a set of common application programming interfaces and a dynamic-link library (DLL) component. The interfaces are used to create and access diverse messaging applications and messaging systems, offering a uniform environment for development and use and providing true independence for both.

Simple MAPI is a subset of 12 functions, which enable developers to add basic messaging functionality. Extended MAPI allows complete control over the messaging system on the client computer, creation and management of messages, management of the client mailbox, service providers, and so forth. Simple MAPI ships with Microsoft Windows as part of Outlook Express/Windows Mail while the full Extended MAPI ships with Office Outlook and Exchange. Extended MAPI serves three main purposes:

- It's the programming interface used to write components that connect to different mail servers, provide access to custom address books and provide rich storage facilities — in other words, the components that you can add through see on the Tools | Services dialog in Outlook.
- You can use MAPI to develop new types of custom forms, not based on the built-in Outlook forms.



- You can create addins for Outlook, Exchange and Windows Messaging that extend the functionality of those clients.

## 9.8 MICROSOFT TRANSACTION SERVER

Microsoft Transaction Server was software that provided services to Component Object Model (COM) software components, to make it easier to create large distributed applications. The major services provided by MTS were automated transaction management, instance management and role-based security. MTS is a component-based transaction processing system for developing, deploying, and managing high-performance, scalable, and robust enterprise, Internet, and intranet server applications. MTS allows us to deploy and administer our MTS server applications with a rich graphical tool. It provides the following features:

1. The MTS run-time environment.
2. **The MTS Explorer:** A graphical user interface for deploying and managing application components.
3. **MTS APIs:** Application programming interfaces(API) and *resource dispensers* for making applications scalable and robust. Resource dispensers are services that manage non-durable shared state on behalf of the application components within a process.
4. **MTS Sample Applications:** Three sample applications that demonstrate how to use the application programming interface (API) to build MTS components, and use scriptable administration objects to automate deployment procedures in the MTS Explorer.

The three-tiered programming model provides more flexibility and opportunity for developers and administrators to move beyond the constraints of two-tier client/server applications because:

- The three-tier model emphasizes a logical architecture for applications, rather than a physical one. Any service may invoke any other service and may reside anywhere.
- These applications are distributed, which means you can run the right components in the right places, benefiting users and optimizing use of network and computer resources.

### 9.8.1 Microsoft Transaction Server Run-Time Environment

The MTS run-time environment is a middle-tier platform for running those components that encapsulate business logic. The MTS run-time infrastructure makes application development, deployment, and management easy by providing the application developer and system administrator a comprehensive but easy-to-use set of system services that include:

- Distributed transactions. A *transaction* is a unit of work that is done as an atomic operation that is, the operation succeeds or fails as a whole.
- Automatic management of processes and threads.
- Object instance management.
- A distributed security service to control object creation and use.

- A graphical interface for system administration and component management.

Application developers who rely upon these system services to make their applications scalable and robust can focus on solving their business problems rather than on developing a system infrastructure.

MTS is designed to work with a wide variety of resource managers, including relational database systems, file systems, and document storage systems. This allows developers to easily use two or more resource managers within a single application.

### 9.8.2 Microsoft Transaction Server Explorer

The MTS Explorer uses to register and manage components executing in the MTS run-time environment. The MTS Explorer is a graphical user interface for managing and deploying MTS components. System and web administrators as well as developers can use the MTS Explorer to administer, distribute, install, deploy, and test packages. Developers use the MTS Explorer to assemble components into pre-built packages, distribute and test components in the MTS environment. The Explorer allows us to monitor and manage transactions. The Explorer hierarchy depicts how the following items in the run-time environment are organized:

- Computers
- Components
- Interfaces
- Packages
- Roles
- Methods

MTS packages are installed on computers, contain components, and define roles. Components in a package define interfaces and methods.

### 9.8.3 Microsoft Transaction Server APIs

MTS application programming interfaces (APIs) uses to develop scalable and robust applications that take advantage of the features of the MTS run-time environment, and to automate administration of packages and components.

#### Developing Client Applications

Client applications that run outside the MTS run-time environment instantiate MTS objects by using the standard COM library functions

#### Developing Components

While developing MTS components one can use different property of MTS interfaces to:

- Declare that an object's work is complete
- Prevent a transaction from being committed
- Create other MTS objects
- Include other objects' work within the scope of the current object's transaction
- Determine if a caller is in a particular role

- Determine if security is enabled

### Automating MTS Administration

Using Visual Basic Scripting or any other Automation-compatible language, one can automate procedures in the MTS Explorer ranging from installing a prebuilt package to enumerating through related collections.

### 9.8.4 Microsoft Transaction Server Sample Applications

In addition to the documentation, MTS includes useful sample applications. One can copy any part of them into their own applications and modify them as necessary. MTS provides the following sample applications.

Sample	Description
Sample Bank	Sample Bank is a simple transactional database application that demonstrates how to use the MTS application programming interfaces
Tic-Tac-Toe	Tic-Tac-Toe is a simple multiuser game that shows nontransactional components managing shared state.
Administrative Sample Scripts	The administrative object scripts demonstrate how to automate MTS Explorer procedures using VBScript.

### 9.9 VISUAL SOURCESAFE

Microsoft Visual SourceSafe is a file-level version control system that permits many types of organizations to work on several project versions at the same time. This capability is particularly beneficial in a software development environment, where it is used in maintaining parallel code versions. However, the product can also be used to maintain files for any other type of team.

Visual SourceSafe supports cross-platform development by allowing collaborative editing and sharing of data. It is designed to handle the tracking and portability issues involved in maintaining one source control base. For developers, Visual SourceSafe accommodates reusable or object-oriented code. Visual SourceSafe does the following:

- Helps protect ones team from accidental file loss.
- Allows back-tracking to earlier versions of a file.
- Supports branching, sharing, merging, and management of file releases.
- Tracks versions of entire projects.
- Tracks modular code (one file that is reused, or shared, by multiple projects).

### Compatibility

The current release of Visual SourceSafe is fully compatible with database versions 6.0 and earlier.

## **Version Control and File Sharing**

Visual SourceSafe allows the quick and efficient sharing of files among projects. When you add a file to Visual SourceSafe, the file is stored on the database and made available to other users. Changes that have been made to the file are saved so that any user can recover an old version at any time. When a set of files is ready to deliver, Visual SourceSafe makes it easy to share and obtain different versions of the selected set of files.

## **Extensibility**

Using the Visual SourceSafe automation interfaces, you can write extensions based on Visual SourceSafe as needed for your environment. Extensions are usually provided in the form of stand-alone applications written to the automation interfaces by writing an add-in or plug-in that is compatible with the integrated development environment (IDE) of the third-party program that will run the software package.

## **Parallel Development**

Visual SourceSafe supports parallel development and cross-platform development techniques. Such support allows individual team members to complete different parts and versions of a project at the same time, instead of being stalled while waiting for each other to finish certain tasks. File merge operations enable independent work without the need to synchronize changes with those made by other individuals.

In support of parallel operations, Visual SourceSafe also includes a label promotion feature to advance files as needed to different versions of a project.

## **Developer Support**

More and more, developers are accessing Visual SourceSafe functions from their development environments within third-party programs. Visual SourceSafe can be easily integrated with Visual Studio and other development tools, such as Microsoft Access. Visual SourceSafe supports a developer environment in many ways by allowing:

- Setting of folder policies to enable group development scenarios.
- Bug fixes
- Easy transition to a new release of an existing project
- Batch/nightly builds
- Automation of source code control events
- Access to automation interfaces
- Source control over slow connections
- Configuration of new projects for isolated Web development
- Addition of a new Web developer to an existing team Web project
- Tracking of programming modules to allow reusable or object-oriented code

## Database Maintenance

Visual SourceSafe provides a number of powerful database maintenance tools to keep your databases operating efficiently and securely. It supports archival and restoration through easy-to-use wizards, as well as several command line-based maintenance utilities.

### 9.9.1 Limitation

Visual SourceSafe's stability is criticized due to the way Visual SourceSafe uses a direct, file-based access mechanism that allows any client to modify a file in the repository after locking it. If a client machine crashes in the middle of updating a file, it can corrupt that file. Many users of Visual SourceSafe mitigate this risk by making use of a utility provided by Visual SourceSafe that checks the database for corruption and, when able, corrects errors that it finds.

## 9.10 MICROSOFT'S VBSCRIPT

Microsoft's VBScript (Visual Basic Script) is a scripting language used to create dynamic and interactive web pages. VBScript is a subset of Visual Basic, a more developed scripting language, and is commonly used on the Web as a client side scripting language and server-side processing in ASPs (Active Server Pages). The interpreted script language VBScript is designed for Web Browser interpretation. VBScript is similar to scripting languages, including; Netscape's JavaScript, Sun Microsystems's Tcl, IBM's Rexx and the UNIX-derived Perl. These scripting languages have been designed to be used as an extension for html language. A Web Browser receives the scripts for websites through web page documents that are then parsed and processed.

It is designed as a "lightweight" language with a fast interpreter for use in a wide variety of Microsoft environments. VBScript uses the Component Object Model to access elements of the environment within which it is running; for example, the FileSystemObject (FSO) is used to create, read, update and delete files.

VBScript has been installed by default in every desktop release of Microsoft Windows . A VBScript script must be executed within a host environment, of which there are several provided with Microsoft Windows, including: Windows Script Host (WSH), Internet Explorer (IE), and Internet Information Services

### 9.10.1 Adding VbScript To Web Pages

VBScript, are designed as an extension to HTML. The web browser receives scripts along with the rest of the web document. It is the browser's responsibility to parse and process the scripts. Add scripts into your web pages within a pair of `<SCRIPT>` tags. The `<SCRIPT>` tag signifies the start of the script section, while `</SCRIPT>` marks the end. Example:

```
<HTML>
<HEAD>
<TITLE>Working With VBScript</TITLE>
<SCRIPT LANGUAGE="VBScript">
```

```
MsgBox "Welcome to my Web page!"
</SCRIPT>
```

### 9.10.2 Working With Variables

A variable is a named location in computer memory that can use for storage of data during the execution of scripts. variables can be used to:

- Store input from the user gathered via your web page
- Save data returned from functions
- Hold results from calculations

There are two methods for declaring variables in VBScript, explicitly and implicitly. Usually we declare variables explicitly with the *Dim* statement as

```
Dim Name
Dim Name, Address, City, State
```

Variables can be declared implicitly by simply using the variable name within your script. This practice is not recommended. It leads to code that is prone to errors and more difficult to debug.

Example:

```
< script >
Dim name
Sub cmdclickme_OnClick
Dim age
End Sub
< / script >
```

### 9.10.3 Objects And VbScript

Objects enhance the functionality that is provided with HTML. By using VBScript one can extend the capabilities of these controls, integrating and manipulating them from within our scripts. Scripting with objects involves two steps:

- Adding the object to web page using HTML
- Writing script procedures to respond to events that the object provides

#### Adding Objects to Web Pages

Objects are added to a page with the single `<OBJECT>` tag. The properties, or characteristics, of the object are configured using the several `<PARAM>` tag. Example :

```
<OBJECT ID="lblTotalPay" WIDTH=45 HEIGHT=24
CLASSID="CLSID:978C9E23-D4B0-11CE-BF2D-00AA003F40D0">
```

```
<PARAM NAME="ForeColor" VALUE="0">
<PARAM NAME="BackColor" VALUE="16777215">
<PARAM NAME="Caption" VALUE="">
<PARAM NAME="Size" VALUE="1582;635">
```

#### 9.10.4 Linking VbScript With Objects

Once you have added a control to your web page, it can be configured, manipulated and responded to through its properties, methods and events. *Properties* are the characteristics of an object. They include items like a caption, the foreground color and the font size. *Methods* cause an object to perform a task. *Events* are actions that are recognized by an object. The Script Wizard found in the Microsoft ActiveX Control Pad can be used to identify events provided by a control, and to generate script to respond to these events.

##### Example:

```
<SCRIPT LANGUAGE="VBScript">
Sub cmdCalculatePay_onClick
    Dim HoursWorked
    Dim PayRate
    Dim TotalPay
    HoursWorked = InputBox("Enter hours worked: ")
    PayRate = InputBox("Enter pay rate: ")
    TotalPay = HoursWorked * PayRate
    lblTotalPay.caption = TotalPay
End Sub
</SCRIPT>
```

#### 9.10.5 Using VbScript With Forms

As the popularity of web page forms increase, so does the need to be able to validate data before the client browser submits it to the web server. As a scripting language, VBScript is well suited for this task. Once the form has been validated, the same script can be used to forward the data on to the server. The process of validating forms involves checking the form to see if:

- All of the required data is proved
- The data provided is valid

## Chapter-10

# Advance Features of VB 2010

---

### 10.1 INTRODUCTION

In April 2010 Microsoft released Visual Studio 2010, the .NET Framework 4.0 (which includes ASP.NET 4.0), and new versions of their core programming languages: C# 4.0 and Visual Basic 10. Previously, the C# and Visual Basic programming languages were managed by two separate teams within Microsoft, which helps explain why features found in one language was not necessarily found in the other. For example, C# 3.0 introduced collection initializers, which enable developers to define the contents of a collection when declaring it. However, Visual Basic did not support collection initializers. Conversely, Visual Basic has long supported optional parameters in methods, whereas C# did not.

Now, Microsoft merged the Visual Basic and C# teams to help ensure that C# and Visual Basic grow together. Thus, the major functions which are introduced in one language, it should appear in the other as well. To this end, with version 4.0 C# now supports optional parameters and named arguments, two features that have long been part of Visual Basic's vernacular. And, likewise, Visual Basic has been updated to include a number of C# features that it was previously missing.

This chapter explores some of these new features that were added to Visual Basic 2010.

### 10.2 VB COMPILER RUNTIME SWITCH

Visual Basic has a long history, the first version of Visual Basic was released in 1991. VB1 was the first RAD tool for creating Windows applications and created a new way of programming by *drawing* the user interface using a control toolbox. But the actual language syntax was an extension of the older QuickBasic language.

Microsoft did their best to make the transition to .Net as smooth as possible and added a lot of legacy functions in the Microsoft.VisualBasic.dll assembly, such as Left(), Mid(), Right(), Chr(), ChDir(), MsgBox() and so on. These functions aren't really necessary since there are other ways of getting the same result, but it made the transition from VB6 a lot smoother. A lot of other stuff was also added to this assembly, such as the ability to use late binding, are On Error Goto and On Error



Resume Next that still works in VB.Net. The main reason to keep this is to Upgrade Wizard that converted VB6 code into VB.Net , should be make it able to work.

VB , C# and Visual C++ also have their own runtime assemblies. The difference is the size of these assemblies and what they contain. The Microsoft.CSharp.dll, which was added in .Net 4, contains the runtime builder that adds support for the *dynamic* keyword, while Microsoft.VisualBasic.dll contains a lot more. If you only want to write code for platforms that include the full .Net framework, such as desktop applications for Windows or ASP.Net applications for the web. But there is other platform that doesn't include the full framework, such as Windows Phone 7 or XNA for creating games that runs on the Xbox. There can also be other third-party platforms that only contain parts of the .Net framework. When you have limited space for a custom implementation one of the last things you might want to add is the large Microsoft.VisualBasic.dll assembly.

However the assembly is always referenced by default as soon as you create a VB project, but you can actually remove the reference when you compile the project. However you would need to use the command line compiler to do so. But there is a problem with removing the VB assembly and the name of that problem is the CType operator. Since CType is an operator and not a function the conversion will be done inline at compile time, or rather the compiler will try to do it inline but that is not always possible. Converting an integer to a double can easily be done during compile time but converting the generic System.Object to a less generic type is not possible until run-time. So there is a helper function in the VB assembly that does the conversion for you, if you have a reference of a assembly.

So the vbruntime switch that has existed in the command line compiler (vbc.exe) for a long time has now been extended. Instead of just excluding the runtime assembly or using another assembly as the runtime, it will now be able to include the key VB runtime as a reference into your application. This will be a great help to get VB support faster for new platforms.

The vbruntime compiler option has a new vbruntime switch that embeds core functionality from the Visual Basic Runtime Library into an assembly. Where Runtime specifies that the compiler should compile without a reference to the Visual Basic Runtime Library, or with a reference to a specific runtime library. And the switch will enable Visual Basic developers to target their applications and libraries at platforms where the full Visual Basic Runtime hasn't traditionally been available. You can use this switch to enable your Visual Basic application to run on platforms that do not contain the Visual Basic Runtime Library.

### 10.3 AUTO-IMPLEMENTED PROPERTY

In C# we can create properties for classes in simple way by just specifying the below code, this option is called auto-implemented property, as the implementation is taken care by the compiler.

```
public int Price { get; set; }
```

In Visual Basic we don't have such simple option to create properties, hence to create a simple class we have to write the below code..

```
Class Client
  Private Code As String
  Public Property Code() As String
    Get
      Return Code
    End Get
    Set(ByVal value As String)
      Code = value
    End Set
  End Property

  Private Name As String
  Public Property Name() As String
    Get
      Return Name
    End Get
    Set(ByVal value As String)
      Name = value
    End Set
  End Property

  Private CreditLimit As Single = 2000
  Public Property CreditLimit() As Single
    Get
      Return CreditLimit
    End Get
    Set(ByVal value As Single)
      CreditLimit = value
    End Set
  End Property
End Class
```

But in Visual Basic 2010 with auto-implemented properties, a property, including a default value, can be declared in a single line. We can write the below code instead of the above..

```
Class Client
```

```
    Public Property Code As String
```

```
    Public Property Name As String
```

```
    Public Property CreditLimit As Single = 2000
```

```
End Class
```

*Auto-implemented properties* enable you to quickly specify a property of a class without having to write code to Get and Set the property. When you write code for an auto-implemented property, the Visual Basic compiler automatically creates a private field to store the property variable in addition to creating the associated Get and Set procedures. An auto-implemented property is equivalent to a property for which the property value is stored in a private field. The following code example shows an auto-implemented property.

```
    Property Prop2 As String = "Empty"
```

The following code example shows the equivalent code for the previous auto-implemented property example.

```
    Private _Prop2 As String = "Empty"
```

```
    Property Prop2 As String
```

```
        Get
```

```
            Return _Prop2
```

```
        End Get
```

```
        Set(ByVal value As String)
```

```
            _Prop2 = value
```

```
        End Set
```

```
    End Property
```

### 10.3.1 Backing Field

When you declare an auto-implemented property, Visual Basic automatically creates a hidden private field called the *backing field* to contain the property value. The backing field name is the auto-implemented property name preceded by an underscore (\_). For example, if you declare an auto-implemented property named CITY, the backing field is named \_CITY. If you include a member of your class that is also named \_CITY, you produce a naming conflict and Visual Basic reports a compiler error.

The backing field also has the following characteristics:

- The access modifier for the backing field is always Private, even when the property itself has a different access level, such as Public.
- If the property is marked as Shared, the backing field also is shared.

- Attributes specified for the property do not apply to the backing field.
- The backing field can be accessed from code within the class and from debugging tools such as the Watch window. However, the backing field does not show in an IntelliSense word completion list.

### 10.3.2 Initializing an Auto Implemented Property

Any expression that can be used to initialize a field is valid for initializing an auto-implemented property. When you initialize an auto-implemented property, the expression is evaluated and passed to the Set procedure for the property. The following code examples show some auto-implemented properties that include initial values.

```
Property FirstName As String = "Amit"  
Property PartNo As Integer = 54203  
Property Orders As New List(Of Order)(300)
```

You cannot initialize an auto-implemented property that is a member of an Interface, or one that is marked `MustOverride`.

When you declare an auto-implemented property as a member of a Structure, you can only initialize the auto-implemented property if it is marked as `Shared`.

When you declare an auto-implemented property as an array, you cannot specify explicit array bounds. However, you can supply a value by using an array initializer, as shown in the following examples.

```
Property Marks As Integer() = {95, 70}  
Property Temperatures As Integer() = New Integer() {66, 53, 81}
```

### 10.3.3 Property Definitions That Require Standard Syntax

Auto-implemented properties are convenient and support many programming scenarios. However, there are situations in which you cannot use an auto-implemented property and must instead use standard, or *expanded*, property syntax.

You have to use expanded property-definition syntax if you want to do any one of the following:

- Add code to the Get or Set procedure of a property, such as code to validate incoming values in the Set procedure. For example, you might want to verify that a string that represents a telephone number contains the required number of numerals before setting the property value.
- Specify different accessibility for the Get and Set procedure. For example, you might want to make the Set procedure `Private` and the Get procedure `Public`.
- Create properties that are `WriteOnly` or `ReadOnly`.
- Use parameterized properties (including `Default` properties). You must declare an expanded property in order to specify a parameter for the property, or to specify additional parameters for the Set procedure.

- Place an attribute on the backing field, or change the access level of the backing field.
- Provide XML comments for the backing field.

### 10.3.4 Expanding an Auto-Implemented Property

If you have to convert an auto-implemented property to an expanded property that contains a Get or Set procedure, the Visual Basic Code Editor can automatically generate the Get and Set procedures and End Property statement for the property. The code is generated if you put the cursor on a blank line following the Property statement, type a G (for Get) or an S (for Set) and press ENTER. The Visual Basic Code Editor automatically generates the Get or Set procedure for read-only and write-only properties when you press ENTER at the end of a Property statement.

## 10.4 COLLECTION INITIALIZERS

*Collection initializers* provide a shortened syntax that enables you to create a collection and populate it with an initial set of values. Collection initializers are useful when you are creating a collection from a set of known values, for example, a list of menu options or categories, an initial set of numeric values, a static list of strings such as day or month names, or geographic locations such as a list of states that is used for validation.

### Syntax

You identify a collection initializer by using the From keyword followed by braces ({}). This is similar to the array literal syntax that is described in Arrays in Visual Basic. A collection initializer consists of a list of comma-separated values that are enclosed in braces ({}), preceded by the From keyword, as shown in the following code.

```
Dim names As New List(Of String) From { "Amit " , "Namit " , "Sumit " }
```

When you create a collection by using a collection initializer, each value that is supplied in the collection initializer is passed to the appropriate Add method of the collection. For example, if you create a List(Of T) by using a collection initializer, each string value in the collection initializer is passed to the Add method. If you want to create a collection by using a collection initializer, the specified type must be valid collection type. Examples of valid collection types include classes that implement the IEnumerable(Of T) interface or inherit the CollectionBase class. The specified type must also expose an Add method that meets the following criteria.

- The Add method must be available from the scope in which the collection initializer is being called. The Add method does not have to be public if you are using the collection initializer in a scenario where non-public methods of the collection can be accessed.
- The Add method must be an instance member or Shared member of the collection class, or an extension method.
- An Add method must exist that can be matched based on overload resolution rules, to the types that are supplied in the collection initializer.

When you use a collection initializer to create a collection, the Visual Basic compiler searches for an Add method of the collection type for which the parameters for the Add method match the types of the values in the collection initializer. This Add method is used to populate the collection with the values from the collection initializer. You cannot combine both a collection initializer and an object initializer to initialize the same collection object. You can use object initializers to initialize objects in a collection initializer.

## 10.5 IMPLICIT LINE CONTINUATION SUPPORT

All programming languages use some character to denote the end of a statement. C-flavored languages (like C#) use the semicolon to mark the end of a statement, whereas the Basic programming language has long used the carriage return. VB assumes that each statement is on one line of code. If your line of code is too long then you could use the underscore character to tell the compiler that the line of code continues onto the next line. Thus, when a statement in VB has been split up across multiple lines, you had to use a line-continuation underscore character (`_`) to indicate that the statement wasn't complete.

For example, with VB 2008 the below LINQ query needs to append a “`_`” at the end of each line to indicate that the query is not complete yet:

```
Public Function ChangePassword(ByVal Username As String, _
    ByVal OldPassword As String, _
    ByVal NewPassword As String) As Boolean
    ...
End Function
```

The VB 2010 compiler and code editor now adds support for what is called “implicit line continuation support” – which means that it is smarter about auto-detecting line continuation scenarios, and as a result no longer needs you to explicitly indicate that the statement continues in many, many scenarios. This means that with VB 2010 we can now write the above code with no “`_`” at all:

```
Public Function ChangePassword(ByVal Username As String,
    ByVal OldPassword As String,
    ByVal NewPassword As String) As Boolean
    ...
End Function
```

The implicit line continuation feature also works well when editing XML Literals within VB

## 10.6 MULTILINE LAMBDA EXPRESSIONS AND SUBROUTINES

A *lambda expression* is a function or subroutine without a name that can be used wherever a

delegate is valid. Lambda expressions can be functions or subroutines and can be single-line or multi-line. You can pass values from the current scope to a lambda expression.

### Lambda expression syntax

The syntax of a lambda expression resembles that of a standard function or subroutine. The differences are as follows:

- A lambda expression does not have a name.
- Lambda expressions cannot have modifiers, such as Overloads or Overrides.
- Single-line lambda functions do not use an As clause to designate the return type. Instead, the type is inferred from the value that the body of the lambda expression evaluates to. For example, if the body of the lambda expression is `cust.City = "London"`, its return type is Boolean.
- In multi-line lambda functions, you can either specify a return type by using an As clause, or omit the As clause so that the return type is inferred. When the As clause is omitted for a multi-line lambda function, the return type is inferred to be the dominant type from all the Return statements in the multi-line lambda function. The *dominant type* is a unique type that all other types supplied to the Return statement can widen to. If this unique type cannot be determined, the dominant type is the unique type that all other types supplied to the Return statement can narrow to. If neither of these unique types can be The body of a single-line function must be an expression that returns a value, not a statement. There is no Return statement for single-line functions. The value returned by the single-line function is the value of the expression in the body of the function.
- The body of a single-line subroutine must be single-line statement.
- Single-line functions and subroutines do not include an End Function or End Sub statement.
- You can specify the data type of a lambda expression parameter by using the As keyword, or the data type of the parameter can be inferred. Either all parameters must have specified data types or all must be inferred.
- Optional and Paramarray parameters are not permitted.
- Generic parameters are not permitted.
- Determined, the dominant type is Object.

#### 10.6.1 To create a single-line lambda expression subroutine

1. In any situation where a delegate type could be used, type the keyword `Sub`, as shown in the following example.  

```
Dim add = Sub
```
2. In parentheses, directly after `Sub`, type the parameters of the subroutine. Notice that you do not specify a name after `Sub`.

```
Dim add = Sub (msg As String)
```

3. Following the parameter list, type a single statement as the body of the subroutine.

```
Dim writeMessage = Sub(msg As String) Console.WriteLine(msg)
```

4. You call the lambda expression by passing in a string argument.

```
writeMessage( "Hello" )
```

### 10.6.2 To create a multiline lambda expression function

1. In any situation where a delegate type could be used, type the keyword `Function`, as shown in the following example.

```
Dim add = Function
```

2. In parentheses, directly after `Function`, type the parameters of the function. Notice that you do not specify a name after `Function`.

```
Dim add = Function (index As Integer)
```

3. Press `ENTER`. The `End Function` statement is automatically added.

4. Within the body of the function, add the following code to create an expression and return the value. You do not use an `As` clause to specify the return type.

```
Dim getColumn = Function(index As Integer)
```

```
    Select Case index
```

```
        Case 0
```

```
            Return "FirstName"
```

```
        Case 1
```

```
            Return "LastName"
```

```
        Case 2
```

```
            Return "CompanyName"
```

```
        Case Else
```

```
            Return "LastName"
```

```
    End Select
```

```
End Function
```

You call the lambda expression by passing in an integer argument.

```
Dim getColumn = getColumn(0)
```

### 10.6.3 To create a multiline lambda expression subroutine

1. In any situation where a delegate type could be used, type the keyword `Sub`, as shown in the following example:



```
Dim add = Sub
```

2. In parentheses, directly after Sub, type the parameters of the subroutine. Notice that you do not specify a name after Sub.

```
Dim add = Sub(msg As String)
```

3. Press ENTER. The End Sub statement is automatically added.

4. Within the body of the function, add the following code to execute when the subroutine is invoked

```
Dim writeToLog = Sub(msg As String)
```

```
Dim log As New EventLog()
```

```
log.Source = "Project"
```

```
log.WriteEntry(msg)
```

```
log.Close()
```

```
End Sub
```

You call the lambda expression by passing in a string argument.

```
writeToLog( "Project started." )
```

A lambda expression shares its context with the scope within which it is defined. It has the same access rights as any code written in the containing scope. This includes access to member variables, functions and subs and parameters and local variables in the containing scope.

Access to local variables and parameters in the containing scope can extend beyond the lifetime of that scope. As long as a delegate referring to a lambda expression is not available to garbage collection, access to the variables in the original environment is retained. In the following example, variable target is local to makeTheGame, the method in which the lambda expression playTheGame is defined. Note that the returned lambda expression, assigned to takeAGuess in Main, still has access to the local variable target.

## 10.7 TYPE EQUIVALENCE SUPPORT

You can now deploy an application that has embedded type information instead of type information that is imported from a Primary Interop Assembly (PIA). With embedded type information, your application can use types in a runtime without requiring a reference to the runtime assembly. If various versions of the runtime assembly are published, the application that contains the embedded type information can work with the various versions without having to be recompiled.

The /link option enables you to deploy an application that has embedded type information. The application can then use types in a runtime assembly that implement the embedded type information without requiring a reference to the runtime assembly. /link causes the compiler to make Component Object model (COM) type information in the specified assemblies available to the project that you are currently compiling.

Syntax

/link:fileList or /l:fileList

Where fileList is required. It is comma-delimited list of assembly file names. If the file name contains a space, enclose the name in quotation marks.

Using the /link option is especially useful when you are working with COM interop. You can embed COM types so that your application no longer requires a primary interop assembly (PIA) on the target computer. The /link option instructs the compiler to embed the COM type information from the referenced interop assembly into the resulting compiled code. The COM type is identified by the CLSID (GUID) value. As a result, your application can run on a target computer that has installed the same COM types with the same CLSID values. The /link option embeds only interfaces, structures, and delegates. Embedding COM classes is not supported.

## 10.8 DYNAMIC SUPPORT

Dynamic objects expose members such as properties and methods at run time, instead of in at compile time. This enables you to create objects to work with structures that do not match a static type or format. For example, you can use a dynamic object to reference the HTML Document Object Model (DOM), which can contain any combination of valid HTML markup elements and attributes. Because each HTML document is unique, the members for a particular HTML document are determined at run time. A common method to reference an attribute of an HTML element is to pass the name of the attribute to the `GetProperty` method of the element.

Dynamic objects provide convenient access to dynamic languages such as IronPython and IronRuby. You can use a dynamic object to refer to a dynamic script that is interpreted at run time. You reference a dynamic object by using late binding. In C#, you specify the type of a late-bound object as `dynamic`. In Visual Basic, you specify the type of a late-bound object as `Object`.

You can create custom dynamic objects by using the classes in the *System.Dynamic* namespace. You can also create your own type that inherits the *DynamicObject* class. You can then override the members of the *DynamicObject* class to provide run-time dynamic functionality.

The *DynamicObject* class enables you to define which operations can be performed on dynamic objects and how to perform those operations. For example, you can define what happens when you try to get or set an object property, call a method, or perform standard mathematical operations such as addition and multiplication.

You cannot directly create an instance of the *DynamicObject* class. To implement the dynamic behavior, you may want to inherit from the *DynamicObject* class and override necessary methods. For example, if you need only operations for setting and getting properties, you can override just the `TrySetMember` and `TryGetMember` methods.

The following code example demonstrates how to create an instance of a class that is derived from the *DynamicObject* class.

```
Public Class SampleDynamicObject
```

Inherits DynamicObject

...

```
Dim sampleObject As Object = New SampleDynamicObject()
```

You can also add your own members to classes derived from the DynamicObject class. .

The DynamicObject class implements the DLR interface IDynamicMetaObjectProvider, which enables you to share instances of the DynamicObject class between languages that support the DLR interoperability model. For example, you can create an instance of the DynamicObject class in C# and then pass it to an IronPython function.

## 10.9 COVARIANCE AND CONTRAVARIANCE

In C# and Visual Basic, covariance and contravariance enable implicit reference conversion for array types, delegate types, and generic type arguments. Covariance preserves assignment compatibility and contravariance reverses it.

*Covariance* enables you to use a more derived type than that specified by the generic parameter, whereas *contravariance* enables you to use a less derived type. This allows for implicit conversion of classes that implement variant interfaces and provides more flexibility for matching method signatures with variant delegate types. You can create variant interfaces and delegates by using the new In and Out language keywords. The .NET Framework also introduces variance support for several existing generic interfaces and delegates, including the IEnumerable(Of T) interface and the Func(Of TResult) and Action(Of T) delegates. For more information, see Covariance and Contravariance (C# and Visual Basic).

The following code demonstrates the difference between assignment compatibility, covariance, and contravariance.

‘Assignment compatibility.

```
Dim str As String = “test”
```

```
Dim obj As Object = str
```

‘Covariance.

```
Dim strings As IEnumerable(Of String) = New List(Of String)()
```

```
Dim objects As IEnumerable(Of Object) = strings ‘ An object that is instantiated
```

‘Contravariance.

```
Dim actObject As Action(Of Object) = AddressOf SetObject
```

```
Dim actString As Action(Of String) = actObject
```

## 10.10 NAVIGATE TO

You can use the **Navigate To** feature to search for a symbol or file in source code. You can search for keywords that are contained in a symbol by using Camel casing and underscore characters to divide the symbol into keywords. You can use **Object Browser**, **Navigate To**, **Find Symbol**, or **Go to Definition** to search for objects, definitions, or references (symbols) in a solution.

In the **Object Browser**, you can type a search string to filter the names of the symbols that are displayed in the objects pane for the current browsing scope. For example, the string `MyObject` would return “`MyObject`,” “`MyObjectTest`” and “`CMyObject`.”

You can use the **Navigate To** feature to search for a symbol or file in the source code.

### Searching for Symbols By Using Object Browser

When you type a search string in the **Search** box in **Object Browser**, just the current browsing scope is searched. Use the **Browse** list to select a browsing scope.

To search for symbols by using a search string in Object Browser

1. In **Object Browser**, in the **Browse** list, select a browsing scope.
2. In the **Search** box, type all or part of a symbol name to search for, or select one from the drop-down list.
3. Click **Search**.

The objects pane displays just those symbol names in the browsing scope that include the search string. The string is highlighted in every match.

To clear the results

- In **Object Browser**, click the **Clear Search** button on the toolbar.

The objects pane now displays all the objects in the current browsing scope.

### Searching for Symbols By Using Navigate To

**Navigate To** lets you find and navigate to a specific location in the solution, or explore elements in the solution. It helps you pick a good set of matching results from a query.

To search for symbols or files in Navigate To

1. On the **Edit** menu, click **Navigate To**.
2. In the upper box, type a search string.

Notice that results are displayed in the lower box as you type, and change as you type more. The search results may include symbol definitions and file names in the solution, but does not include namespaces or local variables.

A search string can have multiple search terms, which must be separated by spaces. If a search term has an uppercase letter, the search for that term is case-sensitive; otherwise, the search is case-insensitive. File names are always case-insensitive for the first characters of the file name.

You can search for keywords that are contained in a symbol by using Camel casing and underscore characters to divide the symbol into keywords. For example, to find an “AddOrderHeader” and “Update customer” symbol, you could search the combinations, as shown in the following table.

Symbol Name	Associated Keywords	Matching Search String Examples
AddOrderHeader	Add, Order, Header	“add”, “order”, “header”, “order add”, “AOH”, “a o”, “add ord”, “OrderH”
update_customer	update, customer	“update”, “customer”, “customer upd”, “update c”

The **Navigate To** syntax does not support special logic or special characters such as these:

- Wildcard matching
- Boolean logic operators, including and, or, &, |
- Regular expressions

## 10.11 NEW COMMAND-LINE OPTION

The command-line option causes the compiler to accept only syntax that is valid in the specified version of Visual Basic. `/langversion` causes the compiler to accept only syntax that is included in the specified Visual Basic language version.

### Syntax:

```
/langversion:version
```

Where *version* is required. The language version to be used during the compilation. Accepted values are 9, 9.0, 10, and 10.0.

### Example

```
vbc /langversion:9.0 sample.vb
```

The `/langversion` option specifies what syntax the compiler accepts. For example, if you specify that the language version is 9.0, the compiler generates errors for syntax that is valid only in version 10.0 and later.

You can use this option when you develop applications that target different versions of the .NET Framework. For example, if you are targeting .NET Framework 3.5, you could use this option to ensure that you do not use syntax from language version 10.0.

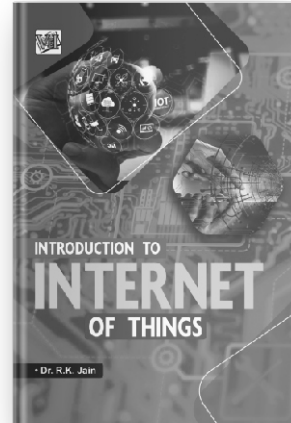
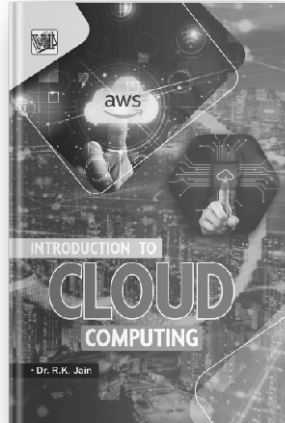
## LET US REVISE

- ✓ VB2010 includes the features of Visual basic and C#.
- ✓ The `/vbruntime` compiler option has a new `/vbruntime` switch that embeds core functionality from the Visual Basic Runtime Library into an assembly.

- ✓ *Auto-implemented properties* provide a shortened syntax that enables you to quickly specify a property of a class without having to write code to Get and Set the property.
- ✓ *Collection initializers* provide a shortened syntax that enables you to create a collection and populate it with an initial set of values.
- ✓ *implicit line continuation* enables you to continue a statement on the next consecutive line without using the underscore character (\_).
- ✓ The `/langversion` command-line option causes the compiler to accept only syntax that is valid in the specified version of Visual Basic.
- ✓ A *lambda expression* is a function or subroutine without a name that can be used wherever a delegate is valid.
- ✓ Visual Basic binds to objects from dynamic languages such as IronPython and IronRuby.
- ✓ *Covariance* enables you to use a more derived type than that specified by the generic parameter.
- ✓ *contravariance* enables you to allow implicit conversion of classes that implement variant interfaces and provides more flexibility for matching method signatures with variant delegate types.
- ✓ **Navigate To** feature is used to search for a symbol or file in source code.

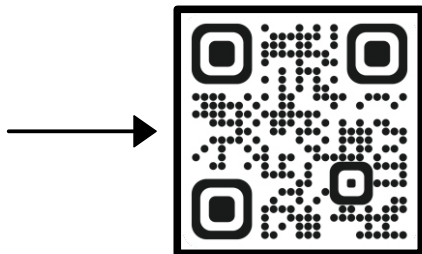
# SPECIAL BONUS!

Want These 3 Bonus Books for free?



Get FREE, unlimited access to these and all of our new books by joining our community!

SCAN w/ your camera TO JOIN!



OR Visit  
[freebie.kartbucket.com](http://freebie.kartbucket.com)